National Technical University of Athens

School of Computer and Electrical Engineering

PATTERN RECOGNITION

Semester Project

Cognitively motivated Deep Learning: Neural Production Systems

Ευάγγελος Δοροβάτας *(03119171)* *vdorovatas@hotmail.gr*
Ορέστης Κωνσταντaρόπουλος *(03119068)* *orestiskonsta@gmail.com*
Ειρήνη Μηλιώρη *(03119071)* *eirhnhmr@gmail.com*
Γεώργιος Τσέλιγκας *(03119431)* *tseligasgio@gmail.com*
Αγγελική Τσινούκα *(03119401)* *el19401@mail.ntua.gr*

TL;DR: *Section 2 summarizes our contribution to the work of Goyal et al. In Sections 3 and 4, we present the experiments. Section 5 provides a critical view of the idea of the paper, while in Section 6 the experimental and overall conclusions are discussed.*

1 Paper Presentation

In the paper we were given, Goyal et al. [1] propose Neural Production Systems (NPS), a model to represent entities within structured visual environments and express knowledge about entity dynamics and interactions. This production system consists of a set of entities and a set of rules, along with a mechanism for selecting rules to apply to subsets of the entities. The authors take inspiration from cognitive science and resurrect a classic AI approach called production systems which aims at knowledge factorization.

From a deep learning perspective, the essence of rules carried over from traditional symbolic systems lies in their operation on variables bound to entities in the world. Each production rule is represented by a distinct Multilayer Perceptron (MLP) with query-key attention mechanisms, specifying rule-entity binding and triggering conditions. The use of abstract, modular, and sparse production rules allows for efficient modeling of interactions among variables in a graph, fostering dynamic parameter sharing among these interactions.

The application of NPS involves multiple steps, starting with parsing an input image into slot-based entities. Subsequent steps include rule selection and application, where attention mechanisms are employed. These steps ensure the systematic application of rules to primary and contextual slots, promoting sparse interactions. Notably, each rule application considers only one contextual slot for modifying a primary slot, contrasting with other methods like Graph Neural Networks (GNNs), which consider all slots for modification. In the final step of rule application, the selected rule is applied to the primary slot based on the rule and the current contents of both primary and contextual slots, leading to a state change through residual addition.

2 Aim of this work

Our objective is to replicate several experiments detailed in the paper, incorporating certain modifications, to validate and assess the resultant outcomes. In addition, our aim is to investigate the conditions and underlying assumptions to which the suggested model (idea) is best suited. We conducted multiple experiments, where we implemented, tested, and enhanced the NPS algorithm across various settings. All these experiments and augmentations are accessible in our GitHub repository. In particular, we set out to explore different aspects and capabilities of the model by investigating both the parallel (PNPS) and sequential (SNPS) variants.

In the SNPS setting, the focus shifts to examining the extent to which the proposed model can learn more general rules and assessing its ability to integrate them into compositional rules (by sequential rule application), subsequently translating into more complex actions performed by the entities. Moreover, we aim to evaluate whether the model can learn the correct order of rule applications. This becomes particularly crucial in settings where the order holds significance (like arithmetic operations) and is not applicable in scenarios such as MNIST transformations (where the order of rotation and translation does not affect the result).

In the PNPS setting, the emphasis lies in challenging the model's capability of understanding the nature of interactions among entities and selecting correct pairs of primary and secondary slots by increasing the number of slots and interactions. Furthermore, we extend our investigation to input points: $$\boldsymbol{X} = [(x_i, y_i), (x_j, y_j)]$$

output points: $$Y = [(\hat{x_i}, \hat{y_i}), (\hat{x_j}, \hat{y_j})]$$

where, $$Y = [(x_i, y_i), (x_j, y_j - y_i)])$$

input to the model:
{ (SLOT1), (SLOT2) } $$[(x_i, y_i, \hat{x_i}, \hat{y_i}), (x_j, y_j, \hat{x_j}, \hat{y_j})]$$

Figure 1: Setup of the arithmetic operation on 2D coordinates

comprehend the role and impact of the Null Rule (indicating no state change). This involves incorporating within our datasets both entities that undergo a state change through interactions and entities that do not interact with others, thereby maintaining their states. We hypothesize that if the model is able to understand the nature of the interactions between entities, it will assign the Null Rule to entities that do not change their state while applying relevant rules (best-matched) to those engaging in interactions.

3 Experiments

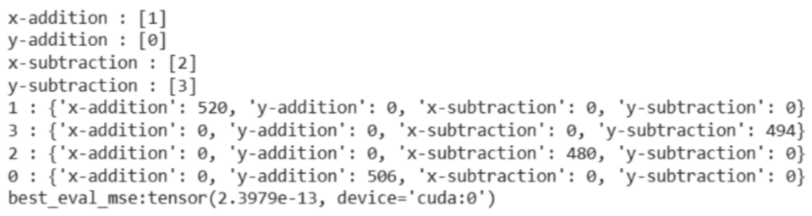
We explore the learning ability of our Neural Production System through a series of experiments. These experiments focus on well-defined, discrete operations, allowing us to evaluate how accurately NPS can recover and represent distinct rules inherent in the data. Starting with straightforward scenarios, we aim to understand NPS's learning capabilities in the context of discrete operations. These initial experiments serve as a foundation for our exploration. Moving forward, we introduce a more complex environment with visually rich setting featuring abstract physical rules. This poses a challenge for NPS to factorize knowledge into separate rules, testing its scalability and adaptability to intricate settings.

3.1 Reproduction

3.1.1 Arithmetic

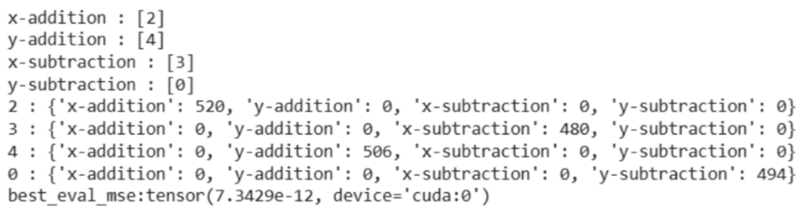
In the paper, it is proposed to implement an arithmetic experiment on 2D coordinates to evaluate the performance of the NPS. The experiment rules are addition or subtraction to either the x or the y coordinate. Given two input points and the expected output, the NPS model should be able to infer the rule that has been applied. The setup of the experiment is shown in Figure 1.

Executing the experiment for 300 epochs and 4 and 5 rules we get the results that are shown in Figure 2. We observe that the model is learning excellently the different rules as the model picks the right rule to use for each operation. In the case of 5 rules, we can see in Figure 2 that the model uses only 4 out of 5 rules (as it can be seen in the rows' indices only 0,2,3,4 rules are chosen).



```
x-addition : [1]
y-addition : [0]
x-subtraction : [2]
y-subtraction : [3]
1 : {'x-addition': 520, 'y-addition': 0, 'x-subtraction': 0, 'y-subtraction': 0}
3 : {'x-addition': 0, 'y-addition': 0, 'x-subtraction': 0, 'y-subtraction': 494}
2 : {'x-addition': 0, 'y-addition': 0, 'x-subtraction': 480, 'y-subtraction': 0}
0 : {'x-addition': 0, 'y-addition': 506, 'x-subtraction': 0, 'y-subtraction': 0}
best_eval_mse:tensor(2.3979e-13, device='cuda:0')
```

(a) 4 operation rules



```
x-addition : [2]
y-addition : [4]
x-subtraction : [3]
y-subtraction : [0]
2 : {'x-addition': 520, 'y-addition': 0, 'x-subtraction': 0, 'y-subtraction': 0}
3 : {'x-addition': 0, 'y-addition': 0, 'x-subtraction': 480, 'y-subtraction': 0}
4 : {'x-addition': 0, 'y-addition': 506, 'x-subtraction': 0, 'y-subtraction': 0}
0 : {'x-addition': 0, 'y-addition': 0, 'x-subtraction': 0, 'y-subtraction': 494}
best_eval_mse:tensor(7.3429e-12, device='cuda:0')
```

(b) 5 operation rules

Figure 2: Results of arithmetic operation on 2D coordinates for 300 epochs and different number of operations. The first four rows are the mapping of the model's rules to the actual operations performed. The following four represent the number of the correct operations performed for each i-rule of the model

	Rule 0	**Rule 1**	**Rule 3**	**Rule 4**
Translate up	0	3696	0	0
Translate down	3785	0	0	0
Rotate right	0	0	0	3720
Rotate left	0	0	3799	0

Table 1: The frequency of each rule's application in the specified operations.

3.1.2 MNIST

In this simple task we use the MNIST dataset and we generate data with four transformations: Translate Up, Translate Down, Rotate Right, and Rotate Left. We also use four rules corresponding to each transformation. We feed the NPS module with two entities after been encoded in two slots. For the first entity (input image) we apply the convolutional Encoder of Table 2 to extract a d-dimensional vector and the second entity (one-hot operation vector) is mapped to a d-dimensional vector using learnable weights. Then, using an attention mechanism the NPS match the transformation embedding to the corresponding MLP rule. The MLP of the selected rule is applied to the encoded representation of the image and then it is passed through the Decoder of Table 2 to output the transformed image.

We train the model for 100 epochs and we use a batch size of 50 for training. We observe that the NPS effectively learns discrete rules for each transformation. It is also noteworthy that our model, even from the second epoch, has managed to separate the rules as shown in Table 1. We also present a demonstration of this process in Figure 3.

	Type	**Channel**	**Activation**	**Stride**
	Conv2D [4 x 4]	16	ELU	2
Encoder	Conv2D [4 x 4]	32	ELU	2
	Conv2D [4 x 4]	64	ELU	2
	Linear	100	ELU	-
	Linear	4096	ReLU	-
	Interpolate (scale factor = 2)	-	-	-
Decoder	Conv2D [4x4]	32	ReLU	1
	Interpolate (scale factor = 2)	-	-	-
	Conv2D [4x4]	16	ReLU	1
	Interpolate (scale factor = 2)	-	-	-
	Conv2D [3x3]	1	ReLU	1

Table 2: The architecture of the Encoder and Decoder for the MNIST Transformation task.

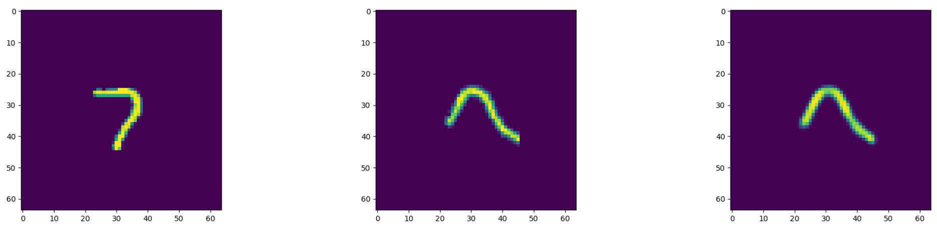


Figure 3: Input Image, Target Image, NPS output

3.2 Extensions

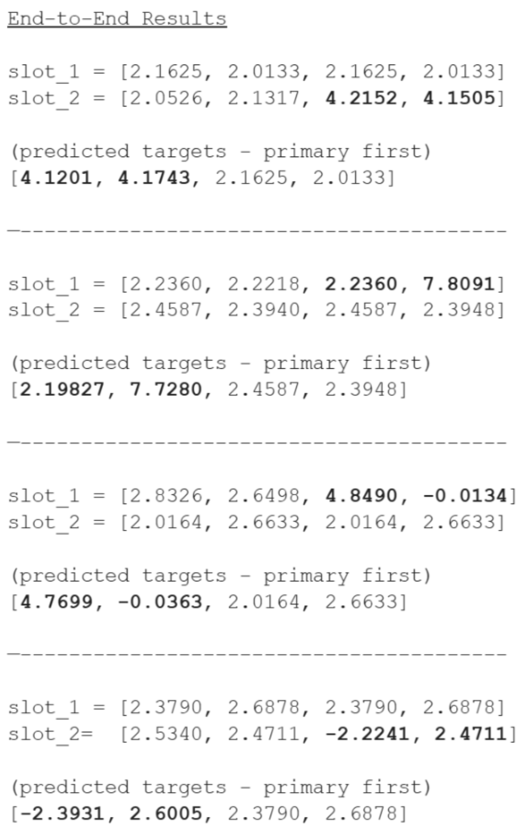
3.2.1 Synthetic

The paper repository implements a synthetic experiment similar to the arithmetic, with scalar points and 3 operations(add, multiply, subtract). The model gets as input the one hot encoding of the operation and the two scalars before and after it. We extend the provided implementation that operates on scalars to implement the set of 2D point operations (X/Y add/subtract). The model learns to separate them fully and achieves an evaluation MSE around 0.005 (equal to that of the scalar case).

3.2.2 Arithmetic

1. An extension to the arithmetic experiment discussed in the paper is the use of the sequential NPS. We want to evaluate the performance of NPS in finding the operations needed from input to output. Similar to the simple arithmetic experiment, the input consists of two 4 dimensional slots (initial-final state of the two 2D points). The difference here is that the primary point (the one with modified final state) has undergone two transformations through arithmetic operations. We aim to investigate the capability of sequential NPS in comprehending and identifying the presence and role of the four basic operations (x-addition, y-addition, x-subtraction, y-subtraction) and leveraging them in a sequential fashion (the order does not matter here) to produce the desired target. To perform this experiment, we extend the dataset having two operations between the coordinates.

In more detail, we choose two implementation techniques to train the model. In the first one, we perform two steps internally in the NPS model and we pass as output the final state produced. The results are shown in Figure 4. The Train Loss is 0.0869 and the Best Mean Square Error in evaluation is 0.0006.



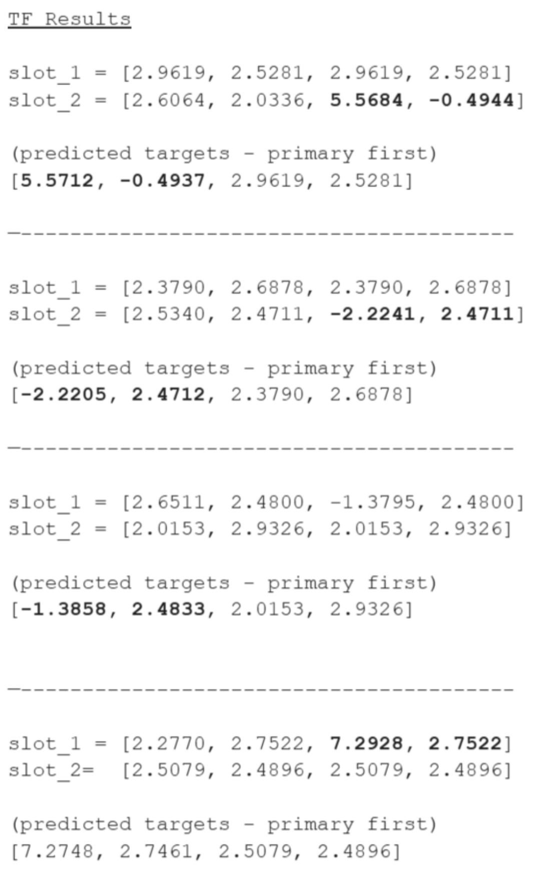
```
End-to-End Results

slot_1 = [2.1625, 2.0133, 2.1625, 2.0133]
slot_2 = [2.0526, 2.1317, 4.2152, 4.1505]

(predicted targets - primary first)
[4.1201, 4.1743, 2.1625, 2.0133]

_________________________________________

slot_1 = [2.2360, 2.2218, 2.2360, 7.8091]
slot_2 = [2.4587, 2.3940, 2.4587, 2.3948]

(predicted targets - primary first)
[2.19827, 7.7280, 2.4587, 2.3948]

_________________________________________

slot_1 = [2.8326, 2.6498, 4.8490, -0.0134]
slot_2 = [2.0164, 2.6633, 2.0164, 2.6633]

(predicted targets - primary first)
[4.7699, -0.0363, 2.0164, 2.6633]

_________________________________________

slot_1 = [2.3790, 2.6878, 2.3790, 2.6878]
slot_2=  [2.5340, 2.4711, -2.2241, 2.4711]

(predicted targets - primary first)
[-2.3931, 2.6005, 2.3790, 2.6878]
```

Figure 4: Arithmetic SNPS end-to-end training

Training end-to-end yields fair results. However, a challenge arises due to the sequential nature of the problem, making it inherently difficult for the model to disentangle the rules between the steps. Consequently, we choose to complement these results with another implementation inspired by teacher forcing (commonly used in problems with a sequential nature). The idea is that in the second step the NPS takes as input the desired intermediate states of the points (the states that would result if the first rule application was perfect). This technique enhances training speed and results in overall improved performance, as illustrated in Figure 5. This, of course, requires access to the intermediate states of the slots which may not be always assured. Here, The Train Loss is 7.2567e-04 and the Best Mean Square Error in evaluation is 1.4664e-05.

Finally, we show some cases of the full inference of the model (step 1 and 2 separately) in Figure 6 and Figure 7. It is apparent that the rules have been learned correctly and the algorithm in each step chooses to apply one of them based on the slots' current states.

2. We propose another extension of the arithmetic experiment to examine the Parallel NPS

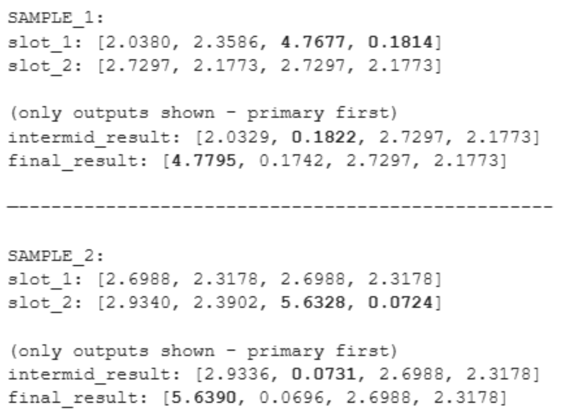


```
TF Results

slot_1 = [2.9619, 2.5281, 2.9619, 2.5281]
slot_2 = [2.6064, 2.0336, 5.5684, -0.4944]

(predicted targets - primary first)
[5.5712, -0.4937, 2.9619, 2.5281]

—-------------------------------------

slot_1 = [2.3790, 2.6878, 2.3790, 2.6878]
slot_2 = [2.5340, 2.4711, -2.2241, 2.4711]

(predicted targets - primary first)
[-2.2205, 2.4712, 2.3790, 2.6878]

—-------------------------------------

slot_1 = [2.6511, 2.4800, -1.3795, 2.4800]
slot_2 = [2.0153, 2.9326, 2.0153, 2.9326]

(predicted targets - primary first)
[-1.3858, 2.4833, 2.0153, 2.9326]


—-------------------------------------

slot_1 = [2.2770, 2.7522, 7.2928, 2.7522]
slot_2=  [2.5079, 2.4896, 2.5079, 2.4896]

(predicted targets - primary first)
[7.2748, 2.7461, 2.5079, 2.4896]
```

Figure 5: Arithmetic SNPS step-by-step training

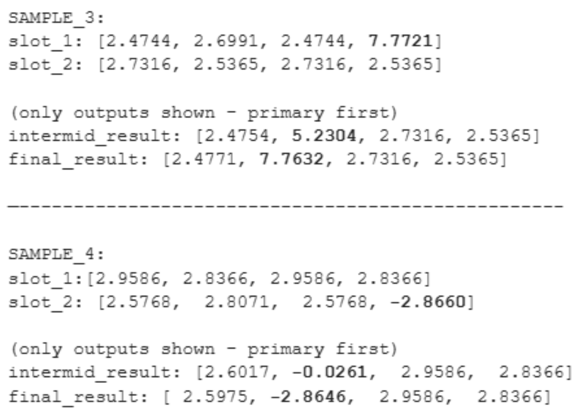
algorithm. In particular, we create a dataset with three 2D points, where each point's final state (output coordinates) are the result of an operation (again, x-addition, y-addition, x-subtraction, y-subtraction) with one of the other two points (chosen randomly). The goal of this experiment is to investigate the capability of parallel NPS to understand and discern the four basic operations, as well as to model dense interactions (in the sense that each entity interacts with another, contrast to the previous experiments). For the implementation, we follow the algorithmic scheme proposed in the paper: each slot picks a rule and a contextual slot to update its state. From Table 3 we can see that the model's training is successful and the rules are correctly distinguished. We provide some samples showing the model's inference in Figure 8.

3. As we presented in the second section, one of our aims was to investigate the role of the Null Rule, which translates in no change in a slot's state. We implemented the logic presented in the original paper and extended the dataset (of the sequential Arithmetic experiment) to incorporate instances with no interactions. However, in practice, the model struggled to accurately learn the basic rules while simultaneously learning when to apply the Null Rule. The exploration/exploitation dilemma, which we discuss later in more detail, is strong here.



```
SAMPLE_1:
slot_1: [2.0380, 2.3586, 4.7677, 0.1814]
slot_2: [2.7297, 2.1773, 2.7297, 2.1773]

(only outputs shown - primary first)
intermid_result: [2.0329, 0.1822, 2.7297, 2.1773]
final_result: [4.7795, 0.1742, 2.7297, 2.1773]

-----------------------------------------------

SAMPLE_2:
slot_1: [2.6988, 2.3178, 2.6988, 2.3178]
slot_2: [2.9340, 2.3902, 5.6328, 0.0724]

(only outputs shown - primary first)
intermid_result: [2.9336, 0.0731, 2.6988, 2.3178]
final_result: [5.6390, 0.0696, 2.6988, 2.3178]
```

Figure 6: Step 1 and 2 of arithmetic SNPS



```
SAMPLE_3:
slot_1: [2.4744, 2.6991, 2.4744, 7.7721]
slot_2: [2.7316, 2.5365, 2.7316, 2.5365]

(only outputs shown - primary first)
intermid_result: [2.4754, 5.2304, 2.7316, 2.5365]
final_result: [2.4771, 7.7632, 2.7316, 2.5365]

-----------------------------------------------

SAMPLE_4:
slot_1:[2.9586, 2.8366, 2.9586, 2.8366]
slot_2: [2.5768,  2.8071,  2.5768, -2.8660]

(only outputs shown - primary first)
intermid_result: [2.6017, -0.0261,  2.9586,  2.8366]
final_result: [ 2.5975, -2.8646,  2.9586,  2.8366]
```

Figure 7: Step 1 and 2 of arithmetic SNPS

Briefly, the algorithm at each training step needs to decide if a wrong produced output (final state) is due to an incorrect rule choice or because of insufficient training of the chosen rule (in particular, the rule's MLP in charge of modifying the states). This makes the optimization procedure hard and in the particular setup results in some atypical scenarios (based on the dataset) where depending on the choice of some hyperparameters the Null Rule dominates the rule selection process or nearly vanishes. We hypothesize that the particular task (due to its arithmetic/continuous nature) may not be suitable for this idea.

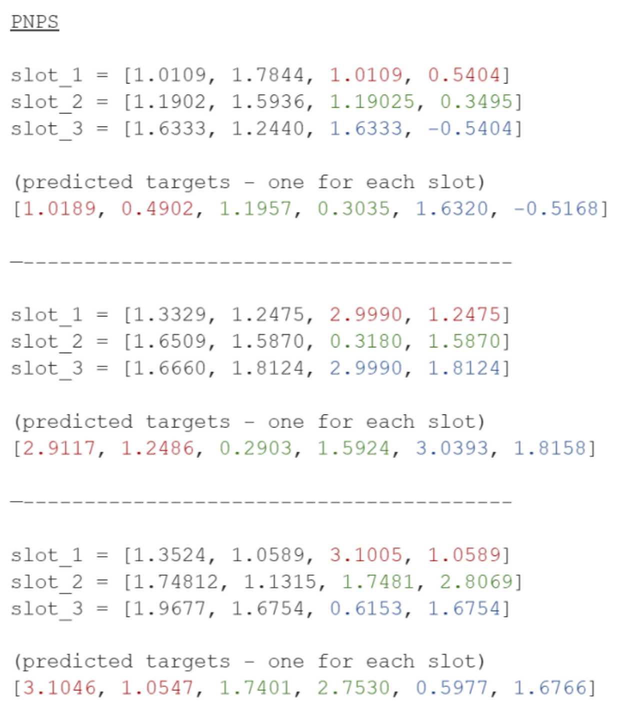
3.2.3 MNIST

1. Exploring New Transformations:

   We evaluate rule networks in a challenging setting with over four transformations. In this way, we examine the ability of rule network and its attention mechanism to select the adequate rule in a more computationally challenging setting.

	Rule 0	Rule 1	Rule 3	Rule 4
X Addition	659	0	0	1
Y addition	0	633	1	0
X Subtraction	0	2	650	0
Y Subtraction	5	0	3	646

Table 3: The frequency of each rule's application arithmetic PNPS



```
PNPS

slot_1 = [1.0109, 1.7844, 1.0109, 0.5404]
slot_2 = [1.1902, 1.5936, 1.19025, 0.3495]
slot_3 = [1.6333, 1.2440, 1.6333, -0.5404]

(predicted targets - one for each slot)
[1.0189, 0.4902, 1.1957, 0.3035, 1.6320, -0.5168]

—---------------------------------------

slot_1 = [1.3329, 1.2475, 2.9990, 1.2475]
slot_2 = [1.6509, 1.5870, 0.3180, 1.5870]
slot_3 = [1.6660, 1.8124, 2.9990, 1.8124]

(predicted targets - one for each slot)
[2.9117, 1.2486, 0.2903, 1.5924, 3.0393, 1.8158]

—---------------------------------------

slot_1 = [1.3524, 1.0589, 3.1005, 1.0589]
slot_2 = [1.74812, 1.1315, 1.7481, 2.8069]
slot_3 = [1.9677, 1.6754, 0.6153, 1.6754]

(predicted targets - one for each slot)
[3.1046, 1.0547, 1.7401, 2.7530, 0.5977, 1.6766]
```

Figure 8: Produced samples of PNPS

At first, we increase the numbers of transformations on MNIST digits to five. We use the same parameters for the training process and we observe that the model learns to separate the rules after been trained for 36 epochs over 2 needed for four transformations. Table 4

Then, we further challenge the model to examine whether it can learn rules for performing six transformations. The model now needs more than 50 epochs to learn 6 different rules for the 6 operations and to use them correctly. In the 50th epoch of training, NPS can only select 4 rules correctly, while for the other two operations it applies a combination of them to the image as show in Table 5.

We can conclude that increasing the number and the complexity of novel transformations applied to MNIST digits results in greater difficulty for the model to accurately distinguish the rules for each operation and apply them, necessitating additional training.

To address the previously mentioned issue, we incorporate an exploration mechanism to compel the model to explore all potential rule-based Multi-Layer Perceptrons (MLPs) during the initial

	Rule 0	**Rule 1**	**Rule 3**	**Rule 4**	**Rule 5**
Translate down	0	3034	0	0	0
Translate up	3028	0	0	0	0
Translate left	0	0	0	0	3018
Horizontal flip	0	0	0	2969	0
Rotate right	0	0	2951	0	0

Table 4: The frequency of each rule's application in the specified operations.

	Rule 0	**Rule 1**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
Translate down	0	2546	0	0	0	0
Translate up	0	0	2499	0	0	0
Translate left	2459	0	0	0	0	0
Translate right	0	0	0	2569	0	0
Rotate right	0	0	0	0	2491	0
Rotate left	0	0	2527	0	0	0

Table 5: The frequency of each rule's application in the specified operations. The translate up and rotate left transform appear at the same rule indicating the model's weakness to separate the rules at the 50th epoch and the need for more epochs training.

stages of training, rather than simply exploiting some and risking getting stuck in a local minimum.

In greater detail, the model tends to train a rule-based MLP that generates a combination of two transformations found in the training set. As training progresses, the NPS attention mechanism, instead of exploring another untrained rule, opts to exploit this intermediate rule, resulting in only mediocre performance.

Hence, we introduce an exploration phase in NPS, during which the model randomly selects rules for each entity with a decreasing probability epsilon. Empirical findings suggest that consistently applying the same rule permutation in each training batch yields superior results.

In table Table 6, we demonstrate that NPS can now rapidly differentiate between six rule-transformations.

2. Less, More General Rules for MNIST:

   We reduce MNIST experiment rules to two and test their ability to incorporate four dataset transformations. We use designed tokens for inference, showcasing NPS's capability to learn

	Rule 0	**Rule 1**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
Translate down	0	0	0	0	2503	0
Translate up	0	2536	0	0	0	0
Translate left	0	0	0	0	0	2488
Translate right	2540	0	0	0	0	0
Rotate right	0	0	2519	0	0	0
Rotate left	0	0	0	2414	0	0

Table 6: After only 6 epochs NPS has managed to distinguish the 6 transformations leveraging the exploration mechanism.

	Rule 0	**Rule 1**
Translate up	5007	0
Translate down	0	4946
Rotate right	0	5038
Rotate left	5009	0

Table 7: This table illustrates the distribution of rules showcasing the frequency of each rule's application in the specified operations. Rule 0 denotes the combination of translating up and rotating left, while Rule 1 corresponds to translating down and rotating right. We train the model for 50 epochs.

specific and general rules. The results are shown in Table 7, where it is clear that NPS learns a rule for rotating right and translating down transformations and a second rule for the rests.

3. Composite Transformations Assessment:

   We apply the SNPS algorithm to the MNIST experiment with $k = 2$ aiming to assess the system's ability to apply composite image transformations. In each step, an order-independent transformation is applied.

   The operation vector now comprises the concatenation of two four-dimensional one-hot vectors, each indicating the transformation at each step. We decided to merge the encoded image embedding with the encoded operation vector, treating this new vector as the NPS entity.

   Utilizing the attention mechanism, we select a rule at each step, expecting the rule network to incorporate second-step information into the entity. This allows the model to choose different rules for the second step. The model appears capable of generating transformed images as intended. However, including the operation vector in the input of the rule MLP is ill-advised due to the MLP's susceptibility to overfitting.

   We log the confusion matrix for both true rule combinations and the selected rule combinations. It appears that the model consistently chooses a single rule combination, albeit with different rules at each step. An example of a composite transformation application is depicted in Figure 9.

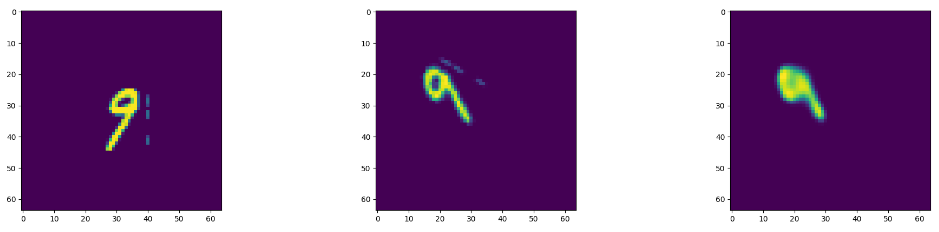


Figure 9: Input Image, Target Image, NPS output for translate up - rotate left transformation

To address the issue of overfitting, we've implemented a new approach. We now apply the rule MLP only to the image part of the NPS entity after concatenating a small vector that indicates the step of SNPS procedure. This adaptation allows the network to streamline its processing, employing a single rule combination for the four transformation combinations that constitute the identity transformation. Furthermore, it effectively discerns between the remaining transformation pairs, allocating a distinct rule combination to each, with the exception of two pairs sharing the same rule combination. Results are shown in Figure 11.

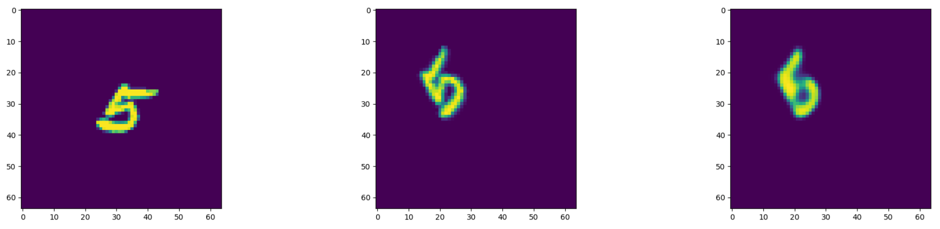


Figure 10: Input Image, Target Image, NPS output for translate up - rotate left transformation

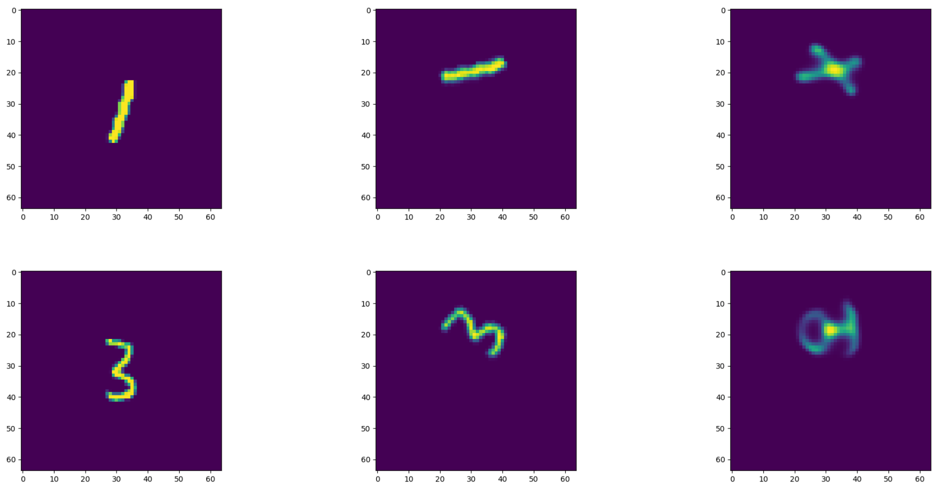


Figure 11: Input Image, Target Image, NPS output for rotate right - translate up and rotate left - translate up transformations. The network chooses the same rule MLP combination for both

4. Double Digit MNIST:

   - Expanding the complexity of the task, we propose an experiment where two MNIST images representing digits a and b are used as input. The resulting output image can represent either of the input digits or a double-digit number ab or ba. Using two rules, our expectation is for the model to identify the primary digit and either leave its image unchanged or concatenate the contextual digit's image from the left.

     We use again a four-dimensional one hot vector as an operation vector, concatenate it to both digit entities and choose rule and primary entity simultaneously with one attention mechanism. We apply the rule MLP to the image part of the entities. The model distinguishes the rules perfectly and makes single and double digits as it is shown in Figure 12.
   - We enhance the existing setup by introducing a third digit image as input to NPS. This augmentation enables the model to select from among the three digits and decide whether

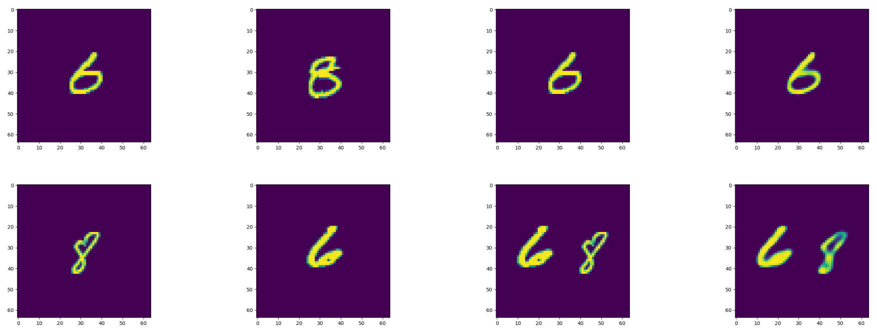


Figure 12: Input Images, Target Image, NPS output for only first and second first rules of double-digit MNIST

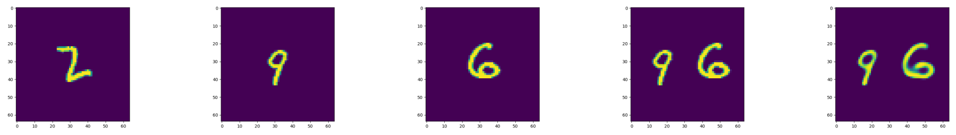


Figure 13: Input Images, Target Image, NPS output for second third rule of double-digit MNIST with three digits as input

to concatenate the adjacent digit from the right or retain it unchanged. The operational vector is a nine-dimensional one-hot encoding representing all potential outcomes, encompassing six different double-digit combinations and three distinct single-digit instances.

While the configuration remains largely unchanged, we now introduce a second attention mechanism to evaluate the model's proficiency in identifying the contextual entity. Once more, the results demonstrate flawless performance as shown in Figure 12.

- We've transitioned the experiment to the sequential framework of SNPS. While the input is still the three digit-images and the rule MLPs are two, the output image can now range from a single digit to any possible combination of double or triple digits achievable through concatenating the input digit-images.

  Our process involves concatenating the operation vector to each image, employing two attention mechanisms to select the rule, primary, and contextual entities, and then applying the chosen rule to the image component of the entities. Subsequently, we reintroduce the resulting entities as input to the NPS.

  In terms of the loss function, we compare all three decoded entities to the three image targets. However, the performance is unsatisfactory as the model solely utilizes one rule without gaining any meaningful insights. We defer the integration of the previously mentioned exploration mechanism to future work.

5. Customizing Transformations:

   We extend the range of transformations to rotations of random angles and translations of random pixels. We pass learnable embeddings of the parameters of each transformation to the rule networks and expect the model to learn general rules that rotate and translate the images with respect to these embeddings. The objective is to enhance the adaptability of rule

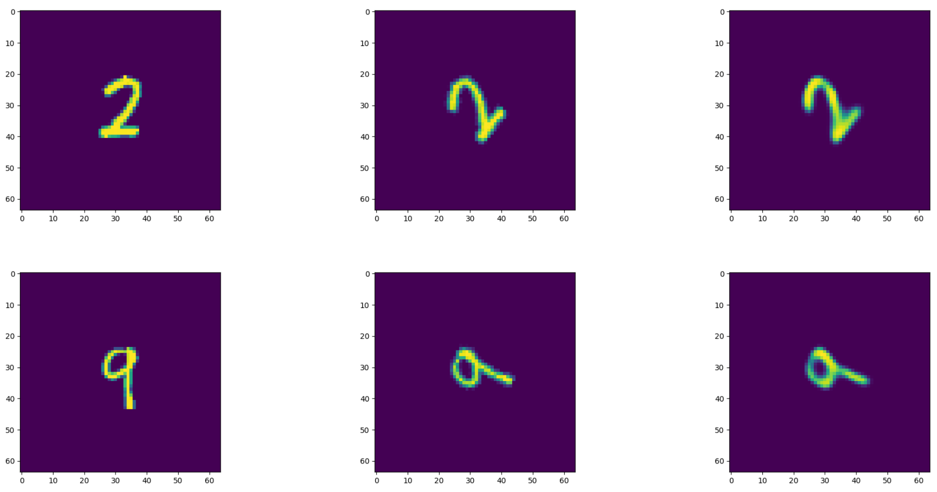


Figure 14: Input Images, Target Image, NPS output for rotate left 60 and 80 degrees for Customizing Transformation experiment

networks to a broader set of image transformations.

In terms of implementation, we adhere to the original framework where two entities are utilized: one for the image and one for the operation vector. Within this framework, we concurrently select a rule and a primary entity. For the selected rule MLP, we input the primary entity concatenated with a vector indicating solely the magnitude of the transformation to be executed. NPS effectively discerns between the four rules and constructs adjustable rule MLPs that can be tuned using a numerical parameter 14.

4 Advanced Experiments

In this section, we describe advanced experiments, which aim at learning an accurate world model and understanding the rules governing the environment. Our objective is to reproduce the experiments and validate the results presented in the paper.

4.1 Physics Environment

In this experiment, a simplified representation of the physical world is simulated, and the aim is to train the model to discover the governing rules, essentially learning an accurate world model. The simulation comprises multiple blocks, each characterized by unique and unknown weights. The underlying dynamics involve sparse interactions between two blocks at a time, where heavier blocks push lighter blocks on their path. This setting introduces an acyclic causal graph between different blocks (shown below). Each unique weight is associated with a unique color. For the model to acquire an accurate world model, it needs to learn the correct mapping from colors to weights. In doing so, the model gains the capability to generalize to shapes it has not encountered previously, inferring their weights based on their color.

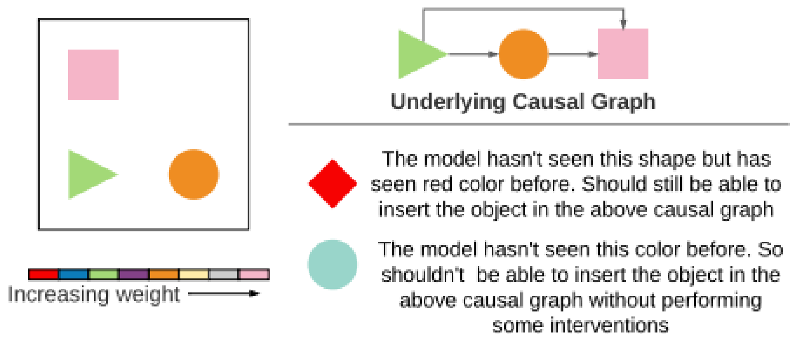

Underlying Causal Graph
The model hasn't seen this shape but has seen red color before. Should still be able to insert the object in the above causal graph
The model hasn't seen this color before. So shouldn't be able to insert the object in the above causal graph without performing some interventions
Increasing weight

Figure 15: Demonstration of the physics environment

For the data collection process, an agent performs actions in the environment collecting image frames (along with the action executed). Each intervention involves moving a block in any of the 4 available directions (left, right, up, and down). If the moved block A with weight W_A comes into contact with another block B with weight W_B, block B will get pushed if $W_B < W_A$ (else B will remain still). The sparsity of interactions in this environment (only two blocks at a time may interact and the others remain unaffected) makes NPS a suitable model whereas using an alternative model that involves dense interactions (like a GNN) may be wasteful.

At each timestep t, the input image is passed through a convolutional encoder to perform object extraction and the produced representation that is separated into M slots $V^t_{1...M}$(which represent entities) is transferred to the latent space by an MLP encoder. We concatenate each of the M slots with the action representation a^t and we pass it to our model. NPS acts as a transition model. The objective is to perform an accurate prediction of $V^{t+k}_{1...M}$ based on image frame x^t (from where we extract $V^t_{1...M}$) and a sequence of actions $a^t, a^{t+1}, ..., a^{t+k}$. The ground truth $\overline{V}^{t+k}_{1...M}$ is obtained by passing the input frame x^{t+k} through the convolutional encoder and then again the MLP encoder. The equations that describe in detail the setup are shown below:

- $\hat{\boldsymbol{x}}^t = \text{Encoder}(\boldsymbol{x}^t)$
- $\boldsymbol{V}^t_{1...M} = \text{Slot Attention}(\hat{\boldsymbol{x}}^t)$
- $\boldsymbol{V}^t_i = \text{Concatenate}(\boldsymbol{V}^t_i, \boldsymbol{a}^t)\ \forall \text{i} \in \{1, \ldots, M\}$
- $\boldsymbol{V}^{t+1}_{1...M} = \text{NPS}(\boldsymbol{V}^t_{1...M})$

Figure 16: Physics Experiment Equations

For evaluation we use the metrics Hits at Rank 1 (H@1) and Mean Reciprocal Rank (MRR) introduced in [2], that measure the proximity of the produced result to the target result.

For training, we train the model for 100 epochs on a dataset of 600 episodes of episode length 30 (30 random interventions by a block pushing agent per episode). Evaluation is performed on a test dataset of 1000 episodes of length 10. We test the performance of the sequential model (with number of rule application steps k=1,3,5,7) on action sequences of length 1,5,10. The results can be seen on Figure 17 and Figure 18. We observe that performance is increasing with the number of rule application steps, as expected, and dropping with the action depth of the transition. SNPS models with more than one rule application steps seem to have a considerable advantage over the 1 step SNPS, which is not performing well at all in deeper transitions. We did not expect to see such a poor performance on 1 step SNPS, but we could attribute that to the small size of the training dataset (due to restricted RAM we used 600 episodes of length 30 in contrast to 1000 episodes of length 100 used in [1]).

As a conclusion we can validate the statement of [1], that NPS is better suited than the GNN, to the physics environment transition predictions. Our results validate or improve the results of [1], and outperform the GNN results of [2], in all cases except of the case with 1 rule selection step and 10 actions transition depth.

The superiority of NPS over GNN is attributed, as mentioned in [1], to the sparsity of rule applications (only 1 rule is applied in every step, and only 1/2 objects change state in every action) and also the contextual sparsity of each rule (1 contextual slot per NPS rule, in contrast to the contextual density of GNN transitions), which is better suited for our block pushing environment, where at most 2 block interact at each time.

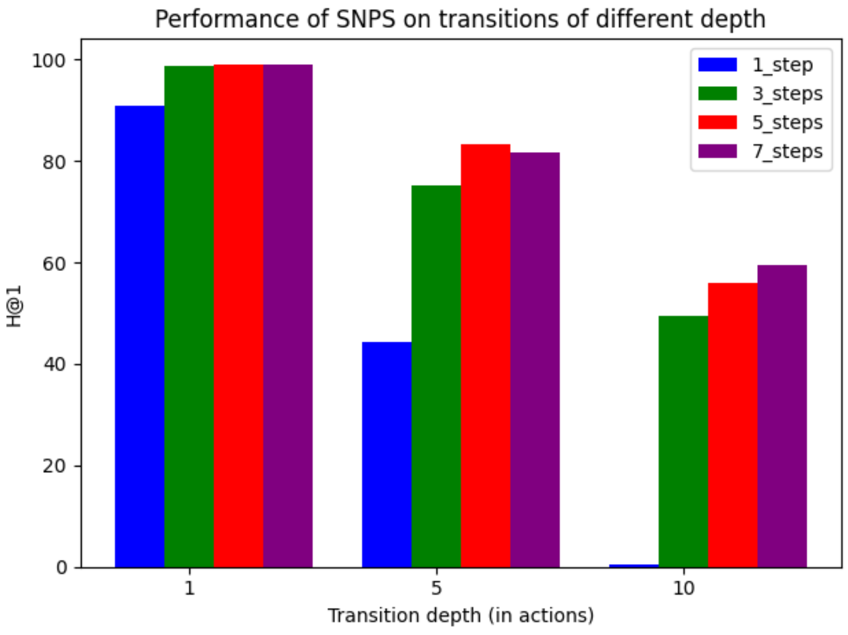


Figure 17: H@1 metric for transitions of action depth 1,5 and 10. SNPS models use 1,3,5 and 7 rule application steps.

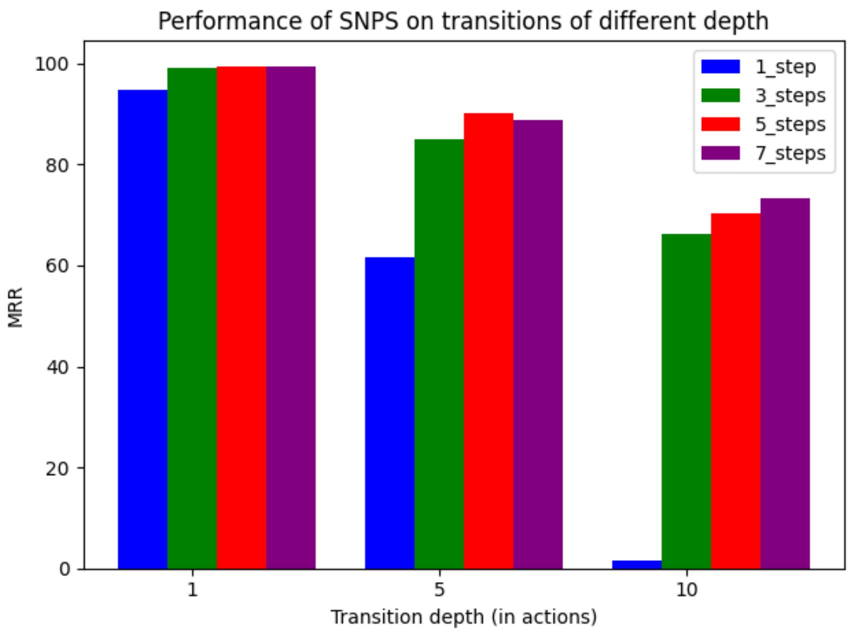


Figure 18: MRR metric for transitions of action depth 1,5 and 10. SNPS models use 1,3,5 and 7 rule application steps.

5 A critique to the general idea, future directions

- In SNPS K is not learnable. The authors opt to predefine and manually tune the number of sequential rule applications based on experimental requirements. The question arises: can K be learnable? Adding a null rule to SNPS might get the model to learn automatically the adequate number of sequences. To ensure convergence, strategies such as enforcing small K values through regularization or setting a maximum possible K could be implemented.
- A key idea introduced in the work we examine is contextual sparsity. NPS instead of modeling dense interactions between all entities using a GNN as previous works do, focuses on interactions between two slots and applies a learnable rule after prioritizing one of the slots. Yet, the question persists: do all interactions strictly involve two entities? The proposed solution is a sequential rule application, suggesting that the interaction of three actions can be modeled sequentially. However, this approach does not completely resolve the issue, as slots are influenced by rule applications in previous stages, and the order of application may impact the final representation.
- The PNPS vs SNPS matter is also left to be resolved empirically. The authors do not propose a way to learn automatically which framework to follow and let the user choose according to the sparsity of the application. Ideally, a framework that composes the advantages of both algorithms could solve this problem.

6 Conclusion

In this study, we extensively analyzed and experimented with Neural Production Systems (NPS), primarily designed for modeling entities' interactions within structured vision environments. Therefore, tasks like the physics environment experiment align well with the model's intended purpose and as the results of this experiment showcase, the model can effectively understand the underlying dynamics of interactions between entities in such environments. However, experimenting with tasks like the Arithmetic Operation or MNIST Transformation, can reveal interesting properties of the algorithm, help test its capabilities and investigate how successful the model is in modularizing knowledge into units/rules, which is its primary purpose.

Moving to the conclusions drawn from the experiments, NPS demonstrates an ability to comprehend the structure of its task at hand. It successfully disentangles the basic rules underlying each task, whether dealing with basic operations or transformations. Notably, as the complexity of the task increases, such as involving more rules, entities, or composite actions, the model adapts and recognizes the correct rules. This highlights the model's ability of understanding the core principles of the particular environment/task, even when the it becomes richer and more complex.

An additional advantage of the algorithm lies in its versatility across different environments, thanks to its two variants: SNPS and PNPS. SNPS is adept at modeling environments with sparse interactions, where the probability of an entity changing its state is low. Another nice property is that it can perform compositions by combining the basic learned rules of an environment and resulting in complex actions. On the other hand, PNPS efficiently models environments where the majority of entities change states frequently. Some issues regarding these variants where discussed in the previous section.

Despite the presented strengths of the algorithm, there are some technical challenge(s) worth noting. As the model is being trained there is an underlying optimization dilemma. A wrong result (produced state) can be generated either because the selected rule was not the correct one or because the MLP associated with this rule (which was the correct) is not yet trained sufficiently to output the correct state result. This poses training difficulties and may impact the model's ability to exhibit the desired behavior (i.e. the learned rules not being the true ones), while it may also result in slow training.

Throughout our experiments, we consistently encountered this challenge. We observed that the complexity of the task directly impacted the model's ability to effectively learn both attentive mechanisms for rule selection and distinct rules simultaneously through standard backpropagation of the loss gradient. In our MNIST experiment involving six transformation rules, we addressed this challenge by introducing an exploration phase. During this phase, we combined the attention mechanism with a random rule selection mechanism encouraging the training of the rule MLPs during the initial stages of training and thereby accelerating the overall training process. Consequently, we successfully achieved full differentiation between the six rules, a task previously deemed unattainable with the provided NPS algorithm.

We posit that the development of a more systematic and theoretically grounded approach to rule selection, capable of effectively navigating the exploration-exploitation dilemma, could significantly enhance NPS results and is of great interest for future research.

7 Reference

[1] Neural Production Systems: Learning Rule-Governed Visual Dynamics, Anirudh Goyal and Aniket Didolkar and Nan Rosemary Ke and Charles Blundell and Philippe Beaudoin and Nicolas Heess and Michael Mozer and Yoshua Bengio, 2022, 2103.01937, arXiv, cs.AI

[2] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. arXiv preprint arXiv:1911.12247, 2019.