Neural Production Systems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Visual environments are structured, consisting of distinct objects or *entities*. These entities have properties—visible or latent—that determine the manner in which they interact with one another. To partition images into entities, deep-learning researchers have proposed structural inductive biases such as slot-based architectures. To model interactions among entities, equivariant graph neural nets (GNNs) are used, but these are not particularly well suited to the task for two reasons. First, GNNs do not predispose interactions to be sparse, as relationships among independent entities are likely to be. Second, GNNs do not factorize knowledge about interactions in an entity-conditional manner. As an alternative, we take inspiration from cognitive science and resurrect a classic approach, *production systems*, which consist of a set of rule templates that are applied by binding placeholder variables in the rules to specific entities. Rules are scored on their match to entities, and the best fitting rules are applied to update entity properties. In a series of experiments, we demonstrate that this architecture achieves a flexible, dynamic flow of control and serves to factorize entity-specific and rule-based information. This disentangling of knowledge achieves robust future-state prediction in rich visual environments, outperforming state-of-the-art methods using GNNs, and allows for the extrapolation from simple (few object) environments to more complex environments.

1 Introduction

Despite never having taken a physics course, every child beyond a young age appreciates that pushing a plate off the dining table will cause the plate to break. The laws of physics accurately characterize the dynamics of our natural world, and although explicit knowledge of these laws is not necessary to reason, we can reason explicitly about objects interacting through these laws. Humans can verbalize knowledge in propositional expressions such as "If a plate drops from table height, it will break," and "If a video-game opponent approaches from behind and they are carrying a weapon, they are likely to attack you." Expressing propositional knowledge is not a strength of current deep learning methods for several reasons. First, propositions are discrete and independent from one another. Second, propositions must be quantified in the manner of first-order logic; for example, the video-game proposition applies to any X for which X is an opponent and has a weapon. Incorporating the ability to express and reason about propositions should improve generalization in deep learning methods because this knowledge is modular— propositions can be formulated independently of each other—and can therefore be acquired incrementally. Propositions can also be composed with each other and applied consistently to all entities that match, yielding a powerful form of systematic generalization.

The classical AI literature from the 1980s can offer deep learning researchers a valuable perspective. In this era, reasoning, planning, and prediction were handled by architectures that performed propositional inference on symbolic knowledge representations. A simple example of such an architecture is the *production system* (Laird et al., 1986; Anderson, 1987), which expresses knowledge by *condition-action rules*. The rules operate on a *working memory*: rule conditions are matched to entities in

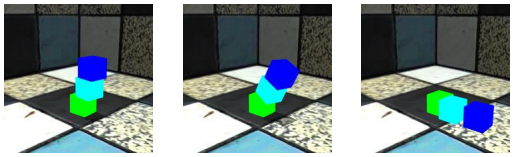
Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

working memory inspired by cognitive science, and such a match can trigger computational actions that update working memory or external actions that operate on the outside world.

Production systems were typically used to model high-level cognition, e.g., mathematical problem solving or procedure following; perception was not the focus of these models. It was assumed that the results of perception were placed into working memory in a symbolic form that could be operated on with the rules. In this article, we revisit production systems but from a deep learning perspective which naturally integrates perceptual processing and subsequent inference for visual reasoning problems. We describe an end-to-end deep learning model that constructs object-centric representations of entities in videos, and then operates on these entities with differentiable—and thus learnable—production rules. The essence of these rules, carried over from traditional symbolic system, is that they operate on variables that are *bound*, or linked, to the entities in the world. In the deep learning implementation, each production rule is represented by a distinct MLP with query-key attention mechanisms to specify the rule-entity binding and to determine when the rule should be triggered for a given entity.

1.1 Variables and entities

What makes a rule general-purpose is that it incorporates placeholder *variables* that can be bound to arbitrary *values* or—the term we prefer in this article—*entities*. This notion of binding is familiar in functional programming languages, where these variables are called arguments. Analogously, the use of variables in the production rules we describe enable a model to reason about any set of entities that satisfy the selection criteria of the rule.



Rule 1	Rule 2
0.84	0.16
0.62	0.38
0.29	0.71

Figure 1: Here we show the rule selection statistics from the proposed model for all entities in the shapes stack dataset across all examples. Each example contains 3 entities as shown above. Each cell in the table shows the probability with which the given rule is triggered for the corresponding entity. We can see that the bottom-most entity triggers rule 2 most of the time while the other 2 entities trigger rule 1 most often. This is quite intuitive as, for most examples, the bottom-most entity remains static and does not move at all while the upper entities fall. Therefore, rule 2 captures information which is relevant to static entities, while rule 1 captures physical rules relevant to the interactions and motion of the upper entities.

In order for rules to operate on entities, these entities must be represented explicitly. That is, the visual world needs to be parsed in a task-relevant manner, e.g., distinguishing the sprites in a video game or the vehicles and pedestrians approaching an autonomous vehicle. Only in the past few years have deep learning vision researchers developed methods for object-centric representation (Le Roux et al., 2011; Eslami et al., 2016; Greff et al., 2016; Raposo et al., 2017; Van Steenkiste et al., 2018; Kosiorek et al., 2018; Engelcke et al., 2019; Burgess et al., 2019; Greff et al., 2019; Locatello et al., 2020a; Ahmed et al., 2020; Goyal et al., 2019; Zablotskaia et al., 2020; Rahaman et al., 2020; Du et al., 2020; Ding et al., 2020; Goyal et al., 2020; Ke et al., 2021). These methods differ in details but share the notion of a fixed number of *slots*, also known as *object files*, each encapsulating information about a single object. Importantly, the slots are interchangeable, meaning that it doesn't matter if a scene with an apple and an orange encodes the apple in slot 1 and orange in slot 2 or vice-versa.

A model of visual reasoning must not only be able to represent entities but must also express knowledge about entity dynamics and interactions. To ensure *systematic* predictions, a model must be capable of applying knowledge to an entity regardless of the slot it is in and must be capable of applying the same knowledge to multiple instances of an entity. Several distinct approaches exist in the literature. The predominant approach uses graph neural networks to model slot-to-slot interactions (Scarselli et al., 2008; Bronstein et al., 2017; Watters et al., 2017; Van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Tacchetti et al., 2018). To ensure systematicity, the GNN must share parameters among the edges. In a recent article, Goyal et al. (2020) developed a more general framework in which parameters are shared but slots can dynamically select which parameters to use in a state-dependent manner. Each set of parameters is referred to as a *schema*, and slots use a query-key attention mechanism to select which schema to apply at each time step. Multiple slots can select the same schema. In both GNNs and SCOFF, modeling dynamics involves each slot interacting with

each other slot. In the work we describe in this article, we replace the direct slot-to-slot interactions with rules, which mediate sparse interactions among slots.

Through our experiments, we show that factorization of knowledge into rules provides a strong inductive bias for learning interaction dynamics among entities in rich physical environments. Representing entity dynamics using NPS leads to impressive performance gains over commonly used models such as GNNs in a wide variety of physical environments. We also find that the distinct rules learned by the proposed model are quite intuitive and interpretable as shown in Figure 1. The figure shows rule selection statistics of the proposed Neural Production System model for the shapes stack dataset (Groth et al., 2018) when using 2 rules. The shapes stack dataset consists of 3 entities that are stacked into a tower and fall under the influence of gravity.

2 Neural Production System: Slots and Sparse Rules

The Neural Production System (NPS), illustrated in Figure 2, provides an architectural backbone that supports the detection and inference of entity (object) representations in an input sequence, and the underlying rules which govern the interactions between these entities in time and space. The input sequence indexed by time step t, $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^t, \ldots, \boldsymbol{x}^T\}$, for instance the frames in a video, are processed by a neural encoder (Burgess et al., 2019; Greff et al., 2019; Goyal et al., 2019, 2020) applied to each $\boldsymbol{x}^t$, to obtain a set of M entity representations $\{\boldsymbol{V}_1^t, \ldots, \ldots, \boldsymbol{V}_M^t\}$, one for each of the M slots. These representations describe an entity and are updated based on both the previous state, $\boldsymbol{V}^{t-1}$ and the current input, $\boldsymbol{x}^t$.

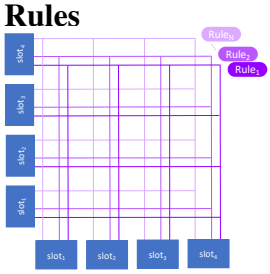


Figure 2: **Rule and slot combinatorics.** Condition-action rules specify how entities interact. Slots maintain the time-varying state of an entity. Every rule is matched to every pair of slots. Through key-value attention, a goodness of match is determined, and a rule is selected along with its binding to slots.

NPS consists of N separately encoded rules, $\{\boldsymbol{R}_1, \boldsymbol{R}_2, .., \boldsymbol{R}_N\}$. Each rule consists of two components, $\boldsymbol{R}_i = (\vec{\boldsymbol{R}}_i, MLP_i)$, where $\vec{\boldsymbol{R}}_i$ is a learned rule embedding vector, which can be thought of as a template defining the condition for when a rule applies; and MLP_i, which determines the action taken by a rule. Both $\vec{\boldsymbol{R}}_i$ and the parameters of MLP_i are learned along with the other parameters of the model using back-propagation on an objective optimized end-to-end.

In the general form of the model, each slot selects a rule that will be applied to it to change its state. This can potentially be performed several times, with possibly different rules applied at each step. Rule selection is done using an attention mechanism described in detail below. Each rule specifies conditions and actions on a pair of slots. Therefore, while modifying the state of a slot using a rule, it can take the state of another slot into account. The slot which is being modified is called the primary slot and other is called the contextual slot. The contextual slot is also selected using an attention mechanism described in detail below.

2.1 Computational Steps in NPS

In this section, we give a detailed description of the rule selection and application procedure for the slots. First, we will formalize the definitions of a few terms that we will use to explain our method. We use the term **primary slot** to refer to slot $\boldsymbol{V}_p$ whose state gets modified by a rule $\boldsymbol{R}_r$. We use the term **contextual slot** to refer to the slot $\boldsymbol{V}_c$ that the rule $\boldsymbol{R}_r$ takes into account while modifying the state of the primary slot $\boldsymbol{V}_p$.

Notation. We consider a set of $\boldsymbol{N}$ rules $\{\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_N\}$ and a set of $\boldsymbol{T}$ input frames $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^T\}$. Each frame $\boldsymbol{x}^t$ is encoded into a set of $\boldsymbol{M}$ slots $\{\boldsymbol{V}_1^t, \boldsymbol{V}_2^t, \ldots, \boldsymbol{V}_M^t\}$. In the following discussion, we omit the index over $\boldsymbol{t}$ for simplicity.

Step 1. is external to NPS and involves parsing an input image, $\boldsymbol{x}^t$, into slot-based entities conditioned on the previous state of the slot-based entities. Any of the methods proposed in the literature to obtain a slot-wise representation of entities can be used (Burgess et al., 2019; Greff et al., 2019; Goyal et al., 2019, 2020). The next three steps constitute the rule selection and application procedure.

Step 2. For each primary slot $\boldsymbol{V}_p$, we attend to a rule $\boldsymbol{R}_r$ to be applied. Here, the queries come from the primary slot: $\boldsymbol{q}_p = \boldsymbol{V}_p W^q$, and the keys come from the rules: $\boldsymbol{k}_i = R_i W^k \quad \forall \boldsymbol{i} \in \{1, \ldots, \boldsymbol{N}\}$. The rule is selected using a straight-through Gumbel softmax (Jang et al., 2016) to achieve a learnable

hard decision: $\boldsymbol{r} = \text{argmax}_i(\boldsymbol{q}_p \boldsymbol{k}_i + \gamma)$, where $\gamma \sim \text{Gumbel}(0, 1)$. This competition is a noisy version of rule matching and prioritization in traditional production systems.

Step 3. For a given primary slot $\boldsymbol{V}_p$ and selected rule $\boldsymbol{R}_r$, a contextual slot $\boldsymbol{V}_c$ is selected using another attention mechanism. In this case the query comes from the primary slot: $\boldsymbol{q}_p = \boldsymbol{V}_p W^q$, and the keys from all the slots: $\boldsymbol{k}_j = \boldsymbol{V}_j W^q \quad \forall \boldsymbol{j} \in \{1, \ldots, \boldsymbol{M}\}$. The selection takes place using a straight-through Gumbel softmax similar to step 2: $\boldsymbol{c} = \text{argmax}_j(\boldsymbol{q}_p \boldsymbol{k}_j + \gamma)$, where $\gamma \sim \text{Gumbel}(0, 1)$. Note that each rule application is sparse since it takes into account only 1 contextual slot for modifying a primary slot, while other methods like GNNs take into account all slots for modifying a primary slot.

Step 4. Rule Application: the selected rule $\boldsymbol{R}_r$ is applied to the primary slot $\boldsymbol{V}_p$ based on the rule and the current contents of the primary and contextual slots. The rule-specific MLP_r, takes as input the concatenated representation of the state of the primary and contextual slots, $\boldsymbol{V}_p$ and $\boldsymbol{V}_c$, and produces an output, which is then used to change the state of the primary slot $\boldsymbol{V}_p$ by residual addition.

2.2 Rule Application: Sequential vs Parallel Rule Application

In the previous section, we have described how each rule application only considers another contextual slot for the given primary slot i.e., **contextual sparsity**. We can also consider **application sparsity**, wherein we use the rules to update the states of only a subset of the slots. In this scenario, only the selected slots would be *primary slots*. This setting will be helpful when there is an entity in an environment that is stationary, or it is following its own default dynamics unaffected by other entities. Therefore, it does not need to consider other entities to update its state. We explore two scenarios for enabling application sparsity.

Parallel Rule Application. Each of the M slots selects a rule to potentially change its state. To enable sparse changes, we provide an extra **Null Rule** in addition to the available N rules. If a slot picks the null rule in step 2 of the above procedure, we do not update its state.

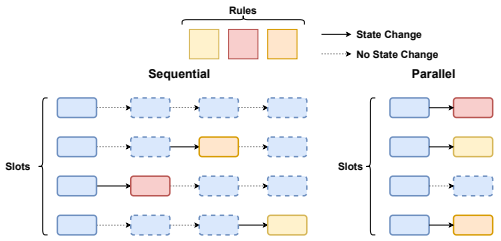


Figure 3: This figure demonstrates the sequential and parallel rule application.

Sequential Rule Application. In this setting, only one slot gets updated in each rule application step. Therefore, only one slot is selected as the primary slot. This can be facilitated by modifying step 2 above to select one {primary slot, rule} pair among $\boldsymbol{NM}$ {rule, slot} pairs. The queries come from each slot: $\boldsymbol{q}_j = \boldsymbol{V}_j W^q \quad \forall \boldsymbol{j} \in \{1, \ldots, \boldsymbol{M}\}$, the keys come from the rules: $\boldsymbol{k}_i = R_i W^k \quad \forall \boldsymbol{i} \in \{1, \ldots, \boldsymbol{N}\}$. The straight-through Gumbel softmax selects one (primary slot, rule) pair: $\boldsymbol{p}, \boldsymbol{r} = \text{argmax}_{i,j}(\boldsymbol{q}_p \boldsymbol{k}_i + \gamma)$, where $\gamma \sim \text{Gumbel}(0, 1)$. In the sequential regime, we allow the rule application procedure (step 2, 3, 4 above) to be performed multiple times iteratively in K rule application stages for each time-step t.

A pictorial demonstration of both rule application regimes can be found in Figure 3. We provide detailed algorithms for the sequential and parallel regimes in Appendix.

3 Related Work

Key-Value Attention. Key-value attention (Bahdanau et al., 2014) defines the backbone of updates to the slots in the proposed model. This form of attention is widely used in Transformer models (Vaswani et al., 2017). Key-value attention selects an input value based on the match of a query vector to a key vector associated with each value. To allow easier learnability, selection is soft and computes a convex combination of all the values. Rather than only computing the attention once, the multi-head dot product attention mechanism (MHDPA) runs through the scaled dot-product attention multiple times in *parallel*. There is an important difference with NPS: in MHDPA, one can treat different heads as different rule applications. Each head (or rule) considers *all* the other entities as relevant arguments as compared to the sparse selection of arguments in NPS.

Sparse and Dense Interactions. GNNs model pairwise interactions between all the slots hence they can be seen as capturing *dense* interactions (Scarselli et al., 2008; Bronstein et al., 2017; Watters et al., 2017; Van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Tacchetti et al., 2018). Instead, verbalizable interactions in the real world are sparse (Bengio, 2017): the immediate effect

of an action is only on a small subset of entities. In NPS, a selected rule only updates the state of a subset of the slots.

In NPS, one can view the computational graph as a dynamically constructed GNN resulting from applying dynamically selected rules, where the states of the slots are represented on the different nodes of the graph, and different rules dynamically instantiate an hyper-edge between a set of slots (the primary slot and the contextual slot). It is important to emphasize that the topology of the graph induced in NPS is dynamic, while in most GNNs the topology is fixed. Through a thorough set of experiments, we show that learning sparse and dynamic interactions using NPS indeed works better for the problems we consider than learning dense interactions using GNNs. We show that NPS outperforms state-of-the-art GNN-based architectures such as C-SWM Kipf et al. (2019) and OP3 Veerapaneni et al. (2019) while learning action conditioned world models.

4 Experiments

We demonstrate the effectiveness of NPS on multiple tasks and compare to a comprehensive set of baselines. To show that NPS can learn intuitive rules from the data generating distribution, we design a couple of simple toy experiments with well-defined discrete operations. Results show that NPS can accurately recover each operation defined by the data and learn to represent each operation using a separate rule. We then move to a much more complicated and visually rich setting with abstract physical rules and show that factorization of knowledge into rules as offered by NPS does scale up to such settings. We study and comparison the parallel and sequential rule application procedures and try to understand the settings which favour each. We then evaluate the benefits of reusable, dynamic and sparse interactions as offered by NPS in a wide variety of physical environments by comparing it against baselines with dense interactions like GNNs. We conduct ablation studies to assess the contribution of different components of NPS. Here we briefly outline the tasks considered and direct the reader to the Appendix for full details on each task and details on hyperparameter settings.

4.1 Learning intuitive rules with NPS: Toy Simulations

We designed a couple of simple tasks with well-defined discrete rules to show that NPS can learn intuitive and interpretable rules. We also show the efficiency and effectiveness of the selection procedure (step 2 and step 3 in section 2.1) by comparing against a baseline with many more parameters. Both tasks require a single modification of only one of the available entities, therefore the use of sequential or parallel rule application would not make a difference here since parallel rule application in which all-but-one slots select the null rule is similar to sequential rule application with 1 rule application step. To simplify the presentation, we describe the setup for both tasks using the sequential rule application procedure.

Table 1: This table shows the segregation of rules for the MNIST Transformation task. Each cell indicates the number of times the corresponding rule is used for the given operation. We can see that NPS automatically and perfectly learns a separate rule for each operation.

	Rule 1	Rule 2	Rule 3	Rule 4
Translate Down	5039	0	0	0
Translate Up	0	4950	0	0
Rotate Right	0	0	5030	0
Rotate Left	0	0	0	4981

MNIST Transformation. We test whether NPS can learn simple rules for performing transformations on MNIST digits. We generate data with four transformations: {Translate Up, Translate Down, Rotate Right, Rotate Left}. We feed the input image ($\boldsymbol{X}$) and the transformation ($\boldsymbol{o}$) to be performed as a one-hot vector to the model. The detailed setup is described in Appendix. For this task, we evaluate whether NPS can learn to use a unique rule for each transformation.

We use 4 rules corresponding to the 4 transformations with the hope that the correct transformations are recovered. Indeed, we observe that **NPS successfully learns to represent each transformation using a separate rule** as shown in Table 1. Our model achieves an MSE of 0.02. A visualization of the outputs from our model can be found in Appendix C.

Coordinate Arithmetic Task. The model is to perform arithmetic operations on 2D coordinates. Given (X_0, Y_0) and (X_1, Y_1), we can apply the following operations: {**X Addition**: $(X_r, Y_r) = (X_0 + X_1, Y_0)$, **X Subtraction**: $(X_r, Y_r) = (X_0 - X_1, Y_0)$, **Y Addition**: $(X_r, Y_r) = (X_0, Y_0 + Y_1)$, **Y Subtraction**: $X_r, Y_r = (X_0, Y_0 - Y_1)$}, where (X_r, Y_r) is the resultant coordinate.

In this task, the model is given 2 input coordinates $X = [(x_i, y_i), (x_j, y_j)]$ and the expected output coordinates $Y = [(\hat{x_i}, \hat{y_i}), (\hat{x_j}, \hat{y_j})]$. The model is supposed to infer the correct rule

to produce the correct output. The correct output is obtained by performing a randomly selected transformation on a randomly selected coordinate in X (primary coordinate), taking another randomly selected coordinate from X (contextual coordinate) into account. The detailed setup is described in Appendix D. We use an NPS model with 4 rules for this task, with the the selection procedure in step 2 and step 3 of algorithm 1 to select the primary coordinate, contextual coordinate, and the rule. For the baseline we replace the selection procedure in NPS (i.e. step 2 and step 3 in algorithm 1) with a routing MLP similar to Fedus et al. (2021). This routing MLP has 3 heads (one each for selecting the primary coordinate, contextual coordinate, and the rule). The baseline has 4 times more parameters than NPS. The final output is produced by the rule MLP which does not have access to the correct output, hence the model cannot simply copy the correct output to produce the actual output. Unlike the MNIST transformation task, we do not provide the operation to be performed as a one-hot vector input to the model, therefore it needs to infer the available operations from the data demonstrations.

We show the segregation of rules for NPS and the baseline in Figure 4. **We can see that NPS learns to use a unique rule for each operation while the baseline struggles to disentangle the underlying operations properly**. **NPS also outperforms the baseline in terms of MSE** achieving an MSE of $0.01_{\pm 0.001}$ while the baseline achieves an MSE of $0.04_{\pm 0.008}$. To further confirm that NPS learns all the available operations correctly from raw data demonstrations, we use an NPS model with 5 rules. **We expect that in this case NPS should utilize only 4 rules since the data describes only 4 unique operations and indeed we observe that NPS ends up mostly utilizing 4 of the available 5 rules** as shown in Table 2.

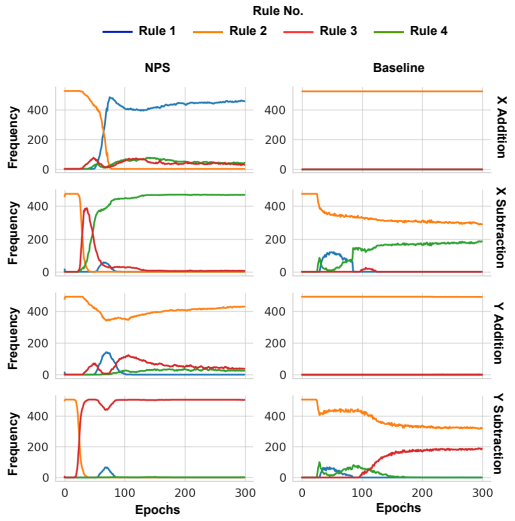


Figure 4: **Coordinate Arithmetic Task.** Here, we compare NPS to the baseline model in terms of segregation of rules as the training progresses. X-axis shows the epochs and Y-axis shows the frequency with which Rule i is used for the given operation. We can see that NPS disentangles the operations perfectly as training progresses with a unique rule specializing to every operation while the baseline model fails to do so.

4.2 Parallel vs Sequential Rule Application

We compare the parallel and sequential rule application procedures, to understand the settings that favour one or the other, over two tasks: (1) Bouncing Balls, (2) Shapes Stack. We use the term PNPS to refer to parallel rule application and SNPS to refer to sequential rule application.

Table 2: This table shows segregation of rules when we use NPS with 5 rules but the data generation distributions describes only 4 possible operations. We can see that only 4 rules get majorly utilized thus confirming that NPS successfully recovers all possible operations described by the data.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
X Addition	360	99	45	13	0
X Subtraction	0	482	0	1	0
Y Subtraction	0	39	453	2	0
Y Addition	0	57	15	99	335

Shapes Stack. We use the shapes stack dataset introduced by Groth et al. (2018). This dataset consists of objects stacked on top of each other as shown in Figure 1. These objects fall under the influence of gravity. For our experiments, we follow the same setup as Qi et al. (2021). Given the first frame, the model is tasked with predicting the object bounding boxes for the next $\boldsymbol{t}$ future frames. Qi et al. (2021) introduce a Region Proposal Interaction Network (RPIN) to solve this task, using a convolution version of the Interaction Network from Battaglia et al. (2016) and called Convolutional Interaction Network (CIN). We replace the CIN in RPIN with NPS. We consider two evaluation settings: (1) the number of rollout timesteps $\boldsymbol{t}$ is as seen during training, $\boldsymbol{T}_{train}$; (2) the number of rollout timesteps $\boldsymbol{t}$ is larger than that seen during training.

We present our results on the shapes stack dataset in Table 3. We can see that both PNPS and SNPS outperform the baseline RPIN in the transfer setting, for results on the test setting (i.e. $\boldsymbol{t} \in [0, \boldsymbol{T}_{train}]$) we ask the reader to refer to appendix. We can see that PNPS outperforms SNPS. We attribute this to

the reduced *application sparsity* with PNPS, i.e., it is more likely that the state of a slot gets updated in PNPS as compared to SNPS. For instance, consider an NPS model with $\boldsymbol{N}$ uniformly chosen rules and $\boldsymbol{M}$ slots. The probability that the state of a slot gets updated in PNPS is $P_{PNPS} = \boldsymbol{N} - 1/\boldsymbol{N}$ (since 1 rule is the null rule), while the same probability for SNPS is $P_{SPNPS} = 1/\boldsymbol{M}$ (since only 1 slot gets updated per rule application step).

For this task, we run both PNPS and SNPS for $\boldsymbol{N} = \{1, 2, 4, 6\}$ rules and $\boldsymbol{M} = 3$. For any given $\boldsymbol{N}$, we can see that $P_{PNPS} > P_{SNPS}$. Even when we have multiple rule application steps in SNPS, it might end up selecting the same slot to be updated in more than one of these steps. We report the best performance obtained for PNPS and SNPS across all $\boldsymbol{N}$, which is $\boldsymbol{N} = 2 + 1$ Null Rule for PNPS and $\boldsymbol{N} = 4$ for SNPS, in Table 3. Shapes stack is a dataset that would prefer a model with less application sparsity since all the objects are tightly bound to each other (objects are placed on top of each other), therefore all objects spend the majority of their time interacting with the objects directly above or below them. We attribute the higher performance of PNPS compared to RPIN to the higher contextual sparsity of PNPS. Each example in the shapes stack task consists of 3 objects. Even though the blocks are tightly bound to each other, each block is only affected by the objects it is in direct contact with. For example, the top-most object is only affected by the object directly below it. The contextual sparsity offered PNPS is a strong inductive bias to model such sparse interactions while RPIN models all pairwise interactions between the objects. Figure 1 shows an illustration of the PNPS model for the shapes stack dataset. In the figure, *Rule 2* actually refers to the Null Rule, while *Rule 1* refers to all the other non-null rules. The bottom-most block picks the Null Rule most times, as the bottom-most block generally does not move.

Model Name	Transfer
RPIN (Qi et al. (2021))	$6.82_{\pm 0.93}$
PNPS	$5.97_{\pm 0.82}$
SNPS	$6.25_{\pm 0.84}$

Table 3: Prediction error of the compared models on the shapes stack dataset (lower is better) for the transfer setting. The *Transfer* setting refers to $\boldsymbol{t} \in [\boldsymbol{T}_{train}, 2 * \boldsymbol{T}_{train}]$. We can see that both SNPS and PNPS outperform the RPIN baseline. Results averaged across 5 random seeds.

Bouncing Balls. We consider a bouncing-balls environment in which multiple balls move with billiard-ball dynamics. We validate our model on a colored version of this dataset. This is a next-step prediction task in which the model is tasked with predicting the final binary mask of each ball. We compare the following methods: (a) SCOFF (Goyal et al., 2020): factorization of knowledge in terms of slots (object properties) and schemata, the latter capturing object dynamics; (b) SCOFF++: we extend SCOFF by using the idea of iterative competition as proposed in slot attention (SA) (Locatello et al., 2020a); SCOFF + PNPS/SNPS: We replace pairwise slot-to-slot interaction in SCOFF++ with parallel or sequential rule application. For comparing different methods, we use the Adjusted Rand Index or ARI (Rand, 1971). To investigate how the factorization in the form of rules allows for extrapolating knowledge from fewer to more objects, we increase the number of objects from 4 during training to 6-8 during testing.

Model Name	Test	Transfer
SCOFF	0.28	0.15
SCOFF++	0.8437	0.2632
PNPS (10 Rules+1 Null Rule)	0.7813	0.1997
SNPS (10 Rules)	0.8518	0.3553

Table 4: Here we show the ARI achieved by the models on the bouncing balls dataset (higher is better). We can see that SNPS outperforms SCOFF and SCOFF++ while PNPS has a poor performance in this task. Results average across 2 seeds.

We present the results of our experiments in Table 4. Contrary to the shapes stack task, we see that SNPS outperforms PNPS for the bouncing balls task. The balls are not tightly bound together into a single tower as in the shapes stack. Most of the time, a single ball follows its own dynamics, only occasionally interacting with another ball. Rules in NPS capture interaction dynamics between entities, hence they would only be required to change the state of an entity when it interacts with another entity. In the case of bouncing balls, this interaction takes place through a collision between multiple balls. Since for a single ball, such collisions are rare, SNPS, which has higher application sparsity (less probability of modifying the state of an entity), performs better as compared to PNPS (lower application sparsity). Also note that, SNPS has the ability to compose multiple rules together by virtue of having multiple rule application stages.

Given the analysis in this section, we can conclude that PNPS is expected to work better when interactions among entities are more frequent while SNPS is expected to work better when interactions are rare and most of the time, each entity follows its own dynamics. Note that, for both SNPS and

PNPS, the rule application considers only 1 other entity as context. Therefore, both approaches have equal *contextual sparsity* while the baselines that we consider (SCOFF and RPIN) capture dense pairwise interactions. We discuss the benefits of *contextual sparsity* in more detail in the next section. More details regarding our setup for the above experiments can be found in Appendix.

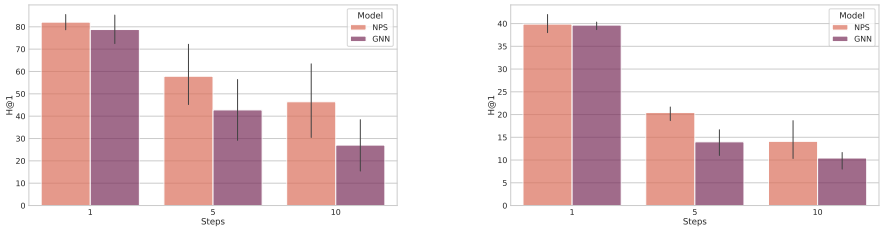
4.3 Benefits of Sparse Interactions Offered by NPS

In this section, we compare NPS to GNNs across a wide variety of physical settings. GNNs offer dense pairwise interactions and modify the state of each of the entities at every step while NPS modifies the state of only a subset of entities per step and uses only one other entity as context while modifying the state of an entity. In this section, we consider two types of tasks: (1) Learning Action Conditioned World Models (2) Physical Reasoning. We use SNPS for all these experiments since in the environments that we consider here, interactions among entities are rare.

Learning Action-Conditioned World Models. Here, we present empirical results to support the discussion in section 3. For learning action-conditioned world models, we follow the same experimental setup as Kipf et al. (2019). Therefore, all the tasks in this section are next-K step ($K = \{1, 5, 10\}$) prediction tasks, given the intermediate actions, and with the predictions being performed in the latent space. We use the Hits at Rank 1 (H@1) metrics described by Kipf et al. (2019) for evaluation. H@1 is 1 for a particular example if the predicted state representation is nearest to the encoded true observation and 0 otherwise. We report the average of this score over the test set (higher is better).

Physics Environment. The physics environment (Ke et al., 2021) simulates a simple physical world. It consists of blocks of unique but unknown weights. The dynamics for the interaction between blocks is that the movement of heavier blocks pushes lighter blocks on their path. This rule creates an acyclic causal graph between the blocks. For an accurate world model, the learner needs to infer the correct weights through demonstrations. Interactions in this environment are sparse and only involve two blocks at a time, therefore we expect NPS to outperform dense architectures like GNNs. This environment is demonstrated in Appendix Fig 9.

We follow the same setup as Kipf et al. (2019). We use their C-SWM model as baseline. For the proposed model, we only replace the GNN from C-SWM by NPS. GNNs generally share parameters across edges, but in NPS each rule has separate parameters. For a fair comparison to GNN, we use an NPS model with 1 rule. Note that this setting is still different from GNNs as in GNNs at each step every slot is updated by instantiating edges between all pairs of slots, while in NPS an edge is dynamically instantiated between a single pair of slots and only the state of the selected slot (i.e., primary slot) gets updated.



(a) **Physics Env** (b) **Atari Games**

Figure 5: **Action-Conditioned World Models**, with number of future steps to be predicted for the world-model on the horizontal axes. (a) Here we show a comparison between GNNs and the proposed NPS on the physics environment using the H@1 metric (higher is better). (b) Comparison of average H@1 scores across 5 Atari games for the proposed model NPS and GNN.

The results of our experiments are presented in Figure 5(a). We can see that NPS outperforms GNNs for all rollouts. Multi-step settings are more difficult to model as errors may get compounded over time steps. The sparsity of NPS (only a single slot affected per step) reduces compounding of errors and enhances symmetry-breaking in the assignment of transformations to rules, while in the case of GNNs, since all entities are affected per step, there is a higher possibility of errors getting compounded. We can see that even with a single rule, we significantly outperform GNNs thus proving the effectiveness of dynamically instantiating edges between entities.

Atari Games. We also test the proposed model in the more complicated setting of Atari. Atari games also have sparse interactions between entities. For instance, in Pong, any interaction involves only 2 entities: (1) paddle and ball or (2) ball and the wall. Therefore, we expect sparse interactions captured by NPS to outperform GNNs here as well.

We follow the same setup as for the physics environment described in the previous section. We present the results for the Atari experiments in Figure 5(b), showing the average H@1 score across 5 games: Pong, Space Invaders, Freeway, Breakout, and QBert. As expected, we can see that the proposed model achieves a higher score than the GNN-based C-SWM. The results for the Atari experiments reinforce the claim that NPS is especially good at learning sparse interactions.

Model	MOTA ↑	MOTP ↑	Mostly Detected ↑	Mostly Tracked ↑	Match ↑	Miss ↓	ID Switches ↓	False Positives ↓
OP3	$89.1_{\pm 5.1}$	$78.4_{\pm 2.4}$	$92.4_{\pm 4.0}$	$91.8_{\pm 3.8}$	$95.9_{\pm 2.2}$	$3.7_{\pm 2.2}$	$0.4_{\pm 0.0}$	$6.8_{\pm 2.9}$
NPS	$90.72_{\pm 5.15}$	$79.91_{\pm 0.1}$	$94.66_{\pm 0.29}$	$93.18_{\pm 0.84}$	$96.93_{\pm 0.16}$	$2.48_{\pm 0.07}$	$0.58_{\pm 0.02}$	$6.2_{\pm 3.5}$

Table 5: **Sprites-MOT**. Comparison between the proposed NPS and the baseline OP3 for various MOT (multi-object tracking) metrics on the sprites-MOT dataset (↑: higher is better, ↓: lower is better). Average over 3 random seeds.

Learning Rules for Physical Reasoning. To show the effectiveness of the proposed approach for physical reasoning tasks, we evaluate NPS on another dataset: Sprites-MOT (He et al., 2018). The Sprites-MOT dataset was introduced by He et al. (2018). The dataset contains a set of moving objects of various shapes. This dataset aims to test whether a model can handle occlusions correctly. Each frame has consistent bounding boxes which may cause the objects to appear or disappear from the scene. A model which performs well should be able to track the motion of all objects irrespective of whether they are occluded or not. We follow the same setup as Weis et al. (2020). We use the OP3 model (Veerapaneni et al., 2019) as our baseline. To test the proposed model, we replace the GNN-based transition model in OP3 with the proposed NPS.

We use the same evaluation protocol as followed by Weis et al. (2020) which is based on the MOT (Multi-object tracking) challenge (Milan et al., 2016). The results for this task are presented in Table 5. We ask the reader to refer to appendix F.1 for more details. We can see that for almost all metrics, NPS outperforms the OP3 baseline. Although this dataset does not contain physical interactions between the objects, sparse rule application should still be useful in dealing with occlusions. At any time step, only a single object is affected by occlusions i.e., it may get occluded due to another object or due to a prespecified bounding box, while the other objects follow their default dynamics. Therefore, a rule should be applied to only the object (or entity) affected (i.e., not visible) due to occlusion and may take into account any other object or entity that is responsible for the occlusion.

5 Discussion and Conclusion

Looking Backward. Production systems were one of the first AI research attempts to model cognitive behaviour and form the basis of many existing models of cognition. However, in traditional symbolic AI, both the key entities and the rules that operated on the entities were given. For AI agents such as robots trying to make sense of their environment, the only observables are low-level variables like pixels in images. To generalize well, an agent must induce high-level entities as well as discover and disentangle the rules that govern how these entities actually interact with each other. Here we have focused on perceptual inference problems and proposed NPS, a neural instantiation of production systems by introducing an important inductive bias in the architecture following the proposals of Bengio (2017); Goyal & Bengio (2020); Ke et al. (2021).

Limitations & Looking Forward. Our experiments on learning action-conditioned world models and extrapolation of knowledge in the form of learned rules in video prediction highlight the advantages brought by the factorization of knowledge into a small set of entities and sparse sequentially applied rules. Immediate future work would investigate how to take advantage of these inductive biases for more complex physical environments (Ahmed et al., 2020) and novel planning methods, which might be more sample efficient than standard ones (Schrittwieser et al., 2020). Humans seem to exploit the inductive bias in the sparsity of the rules and that reasoning about the application of these rules in an abstract space can be very efficient.

Social Impact. The authors do not foresee negative social impact of this work beyond that which could arise from general improvements in ML.

References

Ahmed, O., Träuble, F., Goyal, A., Neitz, A., Wüthrich, M., Bengio, Y., Schölkopf, B., and Bauer, S. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

Anderson, J. R. Skill acquisition: Compilation of weak-method problem situations. *Psychological review*, 94(2):192, 1987.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016. URL `http://arxiv.org/abs/1612.00222`.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bengio, Y. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

Bottou, L. and Gallinari, P. A framework for the cooperation of learning algorithms. In *Advances in neural information processing systems*, pp. 781–788, 1991.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.

Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Cai, J., Shin, R., and Song, D. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611*, 2017.

Ding, D., Hill, F., Santoro, A., and Botvinick, M. Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures. *arXiv preprint arXiv:2012.08508*, 2020.

Du, Y., Smith, K., Ulman, T., Tenenbaum, J., and Wu, J. Unsupervised discovery of 3d physical objects from video. *arXiv preprint arXiv:2007.12348*, 2020.

Engelcke, M., Kosiorek, A. R., Jones, O. P., and Posner, I. Genesis: Generative scene inference and sampling with object-centric latent representations. *arXiv preprint arXiv:1907.13052*, 2019.

Eslami, S., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. Attend, infer, repeat: Fast scene understanding with generative models. *arXiv preprint arXiv:1603.08575*, 2016.

Evans, R., Hernández-Orallo, J., Welbl, J., Kohli, P., and Sergot, M. Making sense of sensory input. *Artificial Intelligence*, pp. 103438, 2019.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021. URL `https://arxiv.org/abs/2101.03961`.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Goyal, A. and Bengio, Y. Inductive biases for deep learning of higher-level cognition. *arXiv preprint arXiv:2011.15091*, 2020.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. Recurrent independent mechanisms, 2019.

Goyal, A., Lamb, A., Gampa, P., Beaudoin, P., Levine, S., Blundell, C., Bengio, Y., and Mozer, M. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Greff, K., Rasmus, A., Berglund, M., Hao, T. H., Schmidhuber, J., and Valpola, H. Tagger: Deep unsupervised perceptual grouping. *arXiv preprint arXiv:1606.06724*, 2016.

Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

Groth, O., Fuchs, F., Posner, I., and Vedaldi, A. Shapestacks: Learning vision-based physical intuition for generalised object stacking. *CoRR*, abs/1804.08018, 2018. URL `http://arxiv.org/abs/1804.08018`.

He, Z., Li, J., Liu, D., He, H., and Barber, D. Tracking by animation: Unsupervised learning of multi-object attentive trackers. *CoRR*, abs/1809.03137, 2018. URL `http://arxiv.org/abs/1809.03137`.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., Hinton, G. E., et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Ke, N. R., Didolkar, A. R., Mittal, S., Goyal, A., Lajoie, G., Bauer, S., Rezende, D. J., Mozer, M. C., Bengio, Y., and Pal, C. Systematic evaluation of causal discovery in visual model based reinforcement learning, 2021. URL `https://openreview.net/forum?id=gp5Uzbl-9C-`.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

Kipf, T., van der Pol, E., and Welling, M. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

Kirsch, L., Kunze, J., and Barber, D. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems*, pp. 2408–2418, 2018.

Kosiorek, A., Kim, H., Teh, Y. W., and Posner, I. Sequential attend, infer, repeat: Generative modelling of moving objects. *Advances in Neural Information Processing Systems*, 31:8606–8616, 2018.

Laird, J. E., Rosenbloom, P. S., and Newell, A. Chunking in soar: The anatomy of a general learning mechanism. *Machine learning*, 1(1):11–46, 1986.

Lamb, A., Goyal, A., Słowik, A., Mozer, M., Beaudoin, P., and Bengio, Y. Neural function modules with sparse arguments: A dynamic approach to integrating information across layers. *arXiv preprint arXiv:2010.08012*, 2020.

Le Roux, N., Heess, N., Shotton, J., and Winn, J. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011.

Li, Y., Gimeno, F., Kohli, P., and Vinyals, O. Strong generalization and efficiency in neural programs. *arXiv preprint arXiv:2007.03629*, 2020.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020a.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention, 2020b.

McMillan, C., Mozer, M. C., and Smolensky, P. The connectionist scientist game: rule extraction and refinement in a neural network. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pp. 424–430, 1991.

Milan, A., Leal-Taixé, L., Reid, I. D., Roth, S., and Schindler, K. MOT16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016. URL `http://arxiv.org/abs/1603.00831`.

Neelakantan, A., Le, Q. V., and Sutskever, I. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

Qi, H., Wang, X., Pathak, D., Ma, Y., and Malik, J. Learning long-term visual dynamics with region proposal interaction networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=_X_4Akcd8Re`.

Rahaman, N., Goyal, A., Gondal, M. W., Wuthrich, M., Bauer, S., Sharma, Y., Bengio, Y., and Schölkopf, B. S2rms: Spatially structured recurrent modules. *arXiv preprint arXiv:2007.06533*, 2020.

Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Raposo, D., Santoro, A., Barrett, D., Pascanu, R., Lillicrap, T., and Battaglia, P. Discovering objects and their relations from entangled scene representations. *arXiv preprint arXiv:1702.05068*, 2017.

Reed, S. and De Freitas, N. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Ronco, E., Gollee, H., and Gawthrop, P. J. Modular neural networks and self-decomposition. *Technical Report CSC-96012*, 1997.

Rosenbaum, C., Klinger, T., and Riemer, M. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

Rosenbaum, C., Cases, I., Riemer, M., and Klinger, T. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Tacchetti, A., Song, H. F., Mediano, P. A., Zambaldi, V., Rabinowitz, N. C., Graepel, T., Botvinick, M., and Battaglia, P. W. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., and Blunsom, P. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pp. 8035–8044, 2018.

Van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2017.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J. B., and Levine, S. Entity abstraction in visual model-based reinforcement learning. *CoRR*, abs/1910.12827, 2019. URL `http://arxiv.org/abs/1910.12827`.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pp. 1439–1456. PMLR, 2020.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pp. 4539–4547, 2017.

Weis, M. A., Chitta, K., Sharma, Y., Brendel, W., Bethge, M., Geiger, A., and Ecker, A. S. Unmasking the inductive biases of unsupervised object representations for video sequences, 2020.

Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., and Savarese, S. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2018.

Zablotskaia, P., Dominici, E. A., Sigal, L., and Lehrmann, A. M. Unsupervised video decomposition using spatio-temporal iterative inference. *arXiv preprint arXiv:2006.14727*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See section 4.3
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] All datasets we used are open-sourced
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include the code for reproducing our experiments
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All datasets we use are synthetic and created using simulation hence do not include any personal or user data.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not use any dataset that includes any kind of identifying information.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]