Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics

Anonymous Authors[1]

Abstract

The recent adaptation of deep neural network-based methods to reinforcement learning and planning domains has yielded remarkable progress on individual tasks. Nonetheless, progress on task-to-task transfer remains limited. In pursuit of efficient and robust generalization, we introduce the Schema Network, an object-oriented generative physics simulator capable of disentangling multiple causes of events and reasoning backward through causes to achieve goals. The richly structured architecture of the Schema Network can learn the dynamics of an environment directly from data. We compare Schema Networks with Asynchronous Advantage Actor-Critic and Progressive Networks on a suite of Breakout variations, reporting results on training efficiency and zero-shot generalization, consistently demonstrating faster, more robust learning and better transfer. We argue that generalizing from limited data and learning causal relationships are essential abilities on the path toward generally intelligent systems.

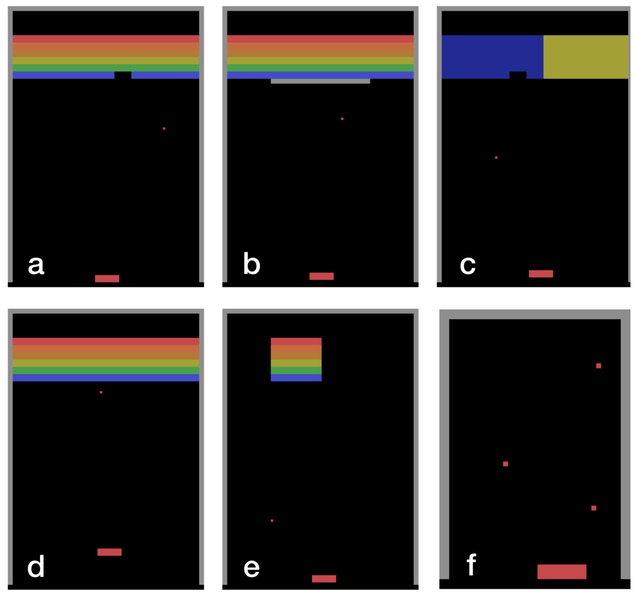


Figure 1. Variations of Breakout. From top left: standard version, middle wall, half negative bricks, offset paddle, random target, and juggling. After training on the standard version, Schema Networks are able to generalize to the other variations without any additional training.

1. Introduction

A longstanding ambition of research in artificial intelligence is to efficiently generalize experience in one scenario to other similar scenarios. Such generalization is essential for an embodied agent working to accomplish a variety of goals in a changing world. Despite remarkable progress on individual tasks like Atari 2600 games (Mnih et al., 2015; Van Hasselt et al., 2016; Mnih et al., 2016) and Go (Silver et al., 2016), the ability of state-of-the-art models to *transfer* learning from one environment to the next remains limited. For instance, consider the variations of Breakout illustrated in Fig. 1. In these environments the positions of objects are perturbed, but the object movements and sources of reward remain the same. While humans have no trouble generalizing experience on the basic Breakout to its variations, deep neural network-based models are easily fooled (Taylor & Stone, 2009; Rusu et al., 2016).

The model-free approach of deep reinforcement learning (Deep RL) such as the Deep-Q Network and its descendants is inherently hindered by the same feature that makes it desirable for single-scenario tasks: it makes no assumptions about the structure of the domain. Recent work has suggested how to overcome this deficiency by utilizing *object-based representations* (Diuk et al., 2008; Usunier et al., 2016). Such a representation is motivated by the well-acknowledged Gestalt principle, which states that the ability to perceive objects as a bounded figure in front of an unbounded background is fundamental to all perception

*Equal contribution [1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

(Weiten, 2012). Battaglia et al. (2016) and Chang et al. (2016) go further, defining hardcoded *relations* between objects as part of the input.

While object-based and relational representations have shown great promise alone, they stop short of modeling *causality* – the ability to reason about previous observations and explain away alternative causes. A causal model is essential for regression planning, in which an agent works backward from a desired future state to produce a plan (Anderson, 1990). Reasoning backward and allowing for multiple causation requires a framework like Probabilistic Graphical Models (PGMs), which natively supports explaining away (Koller & Friedman, 2009).

Here, we introduce Schema Networks – a generative model for object-oriented reinforcement learning and planning. Schema networks incorporate key desiderata for the flexible and compositional transfer of learned prior knowledge to new settings[1]. 1) Knowledge is represented using "schemas" – causal graphical model fragments involving entities, their attributes, and learnable interactions among entities; 2) In a new setting, the appropriate knowledge fragments are automatically instantiated to guide action selection; and 3) The representation deals with uncertainty, multiple-causation and explaining away, and stochasticity in a principled way. We describe the representational framework and learning algorithms and demonstrate how action policies can be generated by treating planning as inference. We evaluate the end-to-end system on Breakout variations and compare against Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) and Progressive Networks (PNs) (Rusu et al., 2016), the latter of which extends A3C explicitly to handle transfer. We show that the rich structure of the Schema Network enables efficient and robust generalization beyond the Deep RL models.

2. Related Work

The field of reinforcement learning has witnessed significant progress with the recent adaptation of deep learning methods to traditional frameworks like Q-learning. Since the introduction of the Deep Q-network (DQN) (Mnih et al., 2015), which uses experience replay to achieve human-level performance on a set of Atari 2600 games, several innovations have enabled faster convergence and better performance with less memory. The asynchronous methods introduced by Mnih et al. (2016) exploit multiple agents acting in copies of the same environment, combining their experiences into one model. As the Asynchronous Advantage Actor-Critic (A3C) is the best among these methods, we use it as our primary point of comparison.

Model-free deep RL models like A3C are unable to substantially generalize beyond their training experience (Jaderberg et al., 2016; Rusu et al., 2016). To address this limitation, recent work has attempted to introduce more structure into the neural network-based models. The Interaction Network (Battaglia et al., 2016) and the Neural Physics Engine (Chang et al., 2016) use object-level and pairwise relational representations to learn models of intuitive physics. The primary advantage of these models is their amenability to gradient-based methods, though such techniques might be applied to Schema Networks as well. Schema Networks offer two key advantages: latent physical properties and relations (schemas) need not be hardcoded; and planning can make use of backward search, since the model can distinguish different causes.

Schema Networks build upon the ideas of the Object-Oriented Markov Decision Process (OO-MDP) introduced by Diuk et al. (2008) (see also (Scholz et al., 2014)). Related frameworks include relational and first-order logical MDPs (Guestrin et al., 2003a). These various formalisms, which harken back to classical AI's roots in symbolic reasoning, are designed to enable robust generalization. Recent work by Garnelo et al. (2016) on "deep symbolic reinforcement learning" makes this connection explicit, marrying first-order logic with deep RL. This effort is similar in spirit to our work with Schema Networks, but like Interaction Networks and Neural Physic Engines, it remains limited without a mechanism for backward planning (also referred to as regression planning).

Generalization is the ability to transfer experience from one scenario to other similar scenarios, or, ideally, dissimilar scenarios that exhibit repeatable structure and sub-structure (Taylor & Stone, 2009). Schema networks achieve this by building an inductive model of the world after observing a limited and biased sample of it. It is also possible to attempt task-to-task transfer by directly using the learned model from one task to begin learning another, without attempting to generalize. This strategy is taken by Rusu et al. (2016) in their work on Progressive Networks (PNs). A PN is constructed by successively training copies of A3C on each task of interest. With each new task, the existing network is frozen, another copy of A3C is added, and lateral connections between the frozen network and the new copy are established to facilitate transfer of features learned during previous tasks. The obvious limitation of PNs is that the number of network parameters must grow quadratically with the number of tasks. However, even if this growth rate was improved, the PN would still be unable to generalize in the manner of Schema Networks to create one coherent model of all experiences.

Schema Networks are built on the technical foundations

[1]We borrow the term "schema" from Drescher (1991), whose schema mechanism inspired the early development of our model.

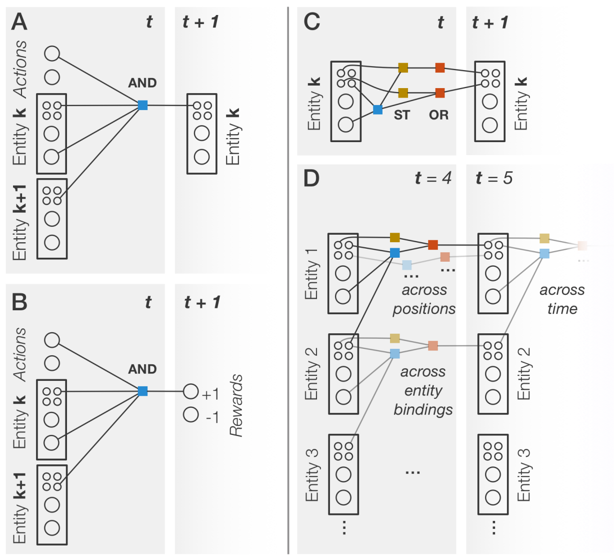


Figure 2. **Construction and architecture of a Schema Network.** A schema is a template for a factor that predicts future reward (A) or the value of an entity-attribute (B) based on entity states and actions taken in the present. Self-transitions (C) predict the value of entity-attributes in the absence of other factors. Self-transitions allow continuous or categorical variables to be represented by a set of binary variables (depicted as smaller nodes). The grounded schema factors and self-transitions are combined to create a Schema Network (D), which gives a generative model of the MDP transition and reward distributions.

of probabilistic graphical models (PGMs), which provide not only an expressive and powerful modeling language, but also a rich toolbox of inference techniques, including the ability to learn the structure of models. More importantly, reasoning with uncertainty and explaining away are naturally supported by PGMs. We direct the readers to (Koller & Friedman, 2009) and (Jordan, 1998) for a thorough overview of PGMs. In particular, early work on factored MDPs established how PGMs can be applied in RL and planning settings (Guestrin et al., 2003b).

3. Schema Networks

3.1. MDPs and Notation

The traditional formalism for the Reinforcement Learning problem is the Markov Decision Process (MDP). An MDP M is a five-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $T(s^{(t+1)}|s^{(t)}, a^{(t)})$ is the probability of transitioning from state $s^{(t)} \in \mathcal{S}$ to $s^{(t+1)} \in \mathcal{S}$ after action $a^{(t)} \in \mathcal{A}$, $R(r^{(t+1)}|s^{(t)}, a^{(t)})$ is the probability of receiving reward $r^{(t+1)} \in \mathbb{R}$ after executing action $a^{(t)}$ while in state $s^{(t)}$, and $\gamma \in [0, 1]$ is the rate at which future rewards are exponentially discounted.

3.2. Model Definition

A Schema Network is a structured generative model of an MDP. We first describe the architecture of the model informally. An image input is parsed into a list of *entities*, which may be thought of as instances of objects in the sense of OO-MDPs (Diuk et al., 2008). All entities share the same collection of *attributes*. We refer to a specific attribute of a specific entity as an *entity-attribute*, which is represented as a binary variable to indicate the presence of that attribute for an entity. An *entity state* is an assignment of states to all attributes of the entity, and the complete model state is the set of all entity states.

A *grounded schema* is a binary variable associated with a particular entity-attribute in the next timestep, conditioned on the present values of other entity-attributes. Each entity-attribute that conditions the distribution is associated with 0 or 1, and the event that the entity-attribute assumes this value is called a *precondition* of the grounded schema. When the preconditions of a grounded schema are satisfied, we say that the schema is active, and it predicts the activation of its associated entity-attribute. Grounded schemas may also predict *rewards* and may be conditioned on *actions*, both of which are represented as binary variables. For instance, a grounded schema might define a distribution over Entity 1's "position" attribute at time 5, conditioned on Entity 2's "position" attribute at time 4 and the action "UP" at time 4. Grounded schemas are instantiated from *ungrounded schemas*, which behave like templates for grounded schemas to be instantiated at different times and in different combinations of entities. For example, an ungrounded schema could predict the "position" attribute of Entity x at time $t+1$ conditioned on the "position" of Entity y at time t and the action "UP" at time t; this ungrounded schema could be instantiated at time $t = 4$ with $x = 1$ and $y = 2$ to create the grounded schema described above. In the case of attributes like "position" that are inherently continuous or categorical, several binary variables may be used to discretely approximate the distribution (see the smaller nodes in Figure 2). A ***Schema Network*** is a factor graph that corresponds to the instantiation of a set of ungrounded schemas for a particular set of entities over some window of time. See Figure 2 for a high-level illustration of the Schema Network architecture.

We now formalize the Schema Network factor graph. For simplicity, suppose the number of entities and the number of attributes are fixed at N and M respectively. Let E_i refer to the i^{th} entity and let $\alpha_{i,j}^{(t)}$ refer to the j^{th} attribute of the i^{th} entity at time t. We use the notation $E_i^{(t)} = (\alpha_{i,1}^{(t)}, ..., \alpha_{i,M}^{(t)})$ to refer to the state of the i^{th} entity at time t. The complete state of the MDP modeled by the network at time t is then $s^{(t)} = (E_1^{(t)}, ..., E_N^{(t)})$. Actions and rewards are also represented with sets of binary variables, denoted $a^{(t)}$ and $r^{(t+1)}$ respectively. A Schema Network for time t will contain the variables in $s^{(t)}$, $a^{(t)}$, $s^{(t+1)}$, and $r^{(t+1)}$.

Let ϕ^k denote the variable for grounded schema k. ϕ^k is bound to a specific entity-attribute $\alpha_{i,j}$, and activates it when the schema is active. Multiple grounded schemas can try to predict the same attribute, and those are combined through an OR gate. For binary variables $v_1, ..., v_n$, let $\text{AND}(v_1, ..., v_n) = \prod_{i=1}^{n} P(v_i = 1)$, and $\text{OR}(v_1, ..., v_n) = 1 - \prod_{i=1}^{n}(1 - P(v_i = 1))$. A grounded schema is connected to its precondition entity-attributes with an AND factor, written as $\phi^k = \text{AND}(\alpha_{i_1,j_1}, ..., \alpha_{i_H,j_H}, a)$ for H entity-attribute preconditions and an optional action a. There is no restriction on how many entities or attributes from a single entity can be preconditions of a grounded schema.

An ungrounded schema (or template) is represented as $\Phi_l(E_{x_1}, ..., E_{x_H}) = \text{AND}(\alpha_{x_1,y_1}, \alpha_{x_1,y_2} ..., \alpha_{x_H,y_H})$, where x_h determines the relative entity index of the h-th precondition and y_h determines which attribute variable is the precondition. The ungrounded schema is a template that can be bound to multiple specific entities and locations, thus generating grounded schemas.

A subset of attributes corresponds to discrete positions. These attributes are treated differently from all others, whose semantic meanings are unknown to the model. When a schema predicts a movement to a new position, we must inform the previously active position attribute to be inactive unless there is another schema that predicts it to remain active. We introduce a *self-transition* variable to represent the probability that a position attribute will remain active in the next time step when no schema predicts a change from that position. We compute the self-transition variable as $\Lambda_{i,j} = \text{AND}(\neg\phi^1, ..., \neg\phi^k, s_{i,j})$ for entity i and position attribute j, where the set $\phi^1 ... \phi^k$ includes all schemas that predict the future position of the same entity i and include $s_{i,j}$ as a precondition.

With these terms defined, we may now compute the transition function, which can be factorized as $T(s^{(t+1)}|s^{(t)}, a^{(t)}) = \prod_{i=1}^{N} \prod_{j=1}^{M} T_{i,j}(s_{i,j}^{(t+1)}|s^{(t)}, a^{(t)})$. An entity-attribute is active at the next time step if either a schema predicts it to be active or if its self-transition variable is active:

$$T_{i,j}(s_{i,j}^{(t+1)}|s^{(t)}) = \text{OR}(\phi^{k_1}, ..., \phi^{k_Q}, \Lambda_{i,j}) \quad (1)$$

Where $k_1 ... k_Q$ are the indices of all grounded schemas that predict $s_{i,j}$.

3.3. Construction of Entities and Attributes

In practice we assume that a vision system is responsible for detecting and tracking entities in an image. It is therefore largely up to the vision system to determine what constitutes an entity. Essentially any trackable image feature could be an entity, which most typically includes objects, their boundaries, and their surfaces. Recent work has demonstrated one possible method for unsupervised entity construction using autoencoders (Garnelo et al., 2016). Depending on the task, Schema Networks could learn to reason flexibly at different levels of representation. For example, using entities from surfaces might be most relevant for predicting collisions, while using one entity per object might be most relevant for predicting whether it can be controlled by an action. The experiments in this paper utilize surface entities, described further in Section 5.

Similarly, entity attributes can be provided by the vision system, and these attributes typically include: color/appearance, surface/edge orientation, object category, or part-of an object category (e.g. front-left tire). For simplicity we here restrict the entities to have fully observable attributes, but in general they could have latent attributes such as "bounciness" or "magnetism".

3.4. Connections to Existing Models

Schema Networks are closely related to Object-Oriented MDPs (OO-MDPs) (Diuk et al., 2008) and Relational MDPs (R-MDPs) (Guestrin et al., 2003a). However, neither OO-MDPs nor R-MDPs define a transition function with an explicit OR of possible causes, and traditionally transition functions have not been learned in these models. In contrast, Schema Networks provide an explicit OR to reason about multiple causation, which enables regression planning. Additionally, the structure of Schema Networks is amenable to efficient learning.

Schema Networks are also related to the recently proposed Interaction Network (IN) (Battaglia et al., 2016) and Neural Physics Engine (NPE) (Chang et al., 2016). At a high level, INs, NPEs, and Schema Networks are much alike – objects are to entities as relations are to schemas. However, neither INs nor NPEs are generative and hence do not support regression planning from a goal through causal chains. Because Schema Networks are generative models, they support more flexible inference and search strategies for planning. Additionally, the learned structures in Schema Networks are amenable to human interpretation, explicitly factorizing different causes, making prediction errors easier to relate to the learned model parameters.

4. Learning and Planning in Schema Networks

In this section we describe how to train Schema Networks (i.e., learn its structure) from interactions with an environment, as well as how they can be used to perform planning. Planning is needed not only at test time to maximize reward, but also to improve exploration during the training procedure.

4.1. Training Procedure

Given a series of actions, rewards and images, we represent each possible action and reward with a binary variable, and we convert each image into a set of entity states S. The number of entities is allowed to vary between adjacent frames, accounting for objects appearing or moving out of view. For each entity we record the attributes of the entities at each position within a local neighborhood. Empty positions in the neighborhood are represented by setting all its attributes (other than position) to zero. This collection of attributes can then be converted into a fixed-length binary feature vector for a given neighborhood radius. This data is aggregated across all frames and provided to the schema learning algorithm described in Section 4.2.

While gathering data, actions are chosen by planning using the schemas that have been learned so far. This planning algorithm is described in Section 4.3. We use an ε-greedy approach to encourage exploration, taking a random action at each timestep with small probability. We found no need to perform any additional policy learning, and after convergence predictions were accurate enough to allow for successful planning. As shown in Section 5, since learning only involves understanding the dynamics of the game, transfer learning is simplified and there is no need for policy adaptation.

4.2. Schema learning

Structure learning in graphical models is a well studied topic in machine learning (Koller & Friedman, 2009; Jordan, 1998). To learn the structure of the Schema Network, we cast the problem as a supervised learning problem over a discrete space of parameterizations (the schemas), and then apply a greedy algorithm that solves a sequence of LP relaxations. See Jaakkola et al. (2010) for further work on applying LP relaxations to structure learning.

Let us arrange the observed inputs to the Schema Network as a binary matrix $X \in \{0,1\}^{N \times D}$, with one observation per row. Similarly, let binary vector $y \in \{0,1\}^N$ represent the observed binary outputs corresponding to the previous inputs. Both X and y are collected during the environment exploration. The output of the Schema Network is an estimation of y and can be expressed as

$$\hat{y} = f_W(X) = \overline{\overline{X}W}\vec{1}$$

where all the involved variables are binary and operations follow Boolean logic: addition corresponds to ORing, and overlining to negation. $W \in \{0,1\}^{D \times M}$ is a binary matrix, with each column representing one (ungrounded) schema. The variables set to 1 in each schema represent an existing connection between that schema and an input condition (see Fig. 2). The outputs of each individual schema are ORed to produce the final prediction.

We would like to minimize the prediction error of Schema Networks while keeping them as simple as possible. A suitable objective function is

$$\min_{W \in \{0,1\}^{D \times M}} \frac{1}{N}|y - f_W(X)|_1 + C|W|_1, \qquad (2)$$

where the first term computes the prediction error, the second term estimates the complexity and parameter C controls the trade-off between both. This is an NP-hard problem for which we cannot hope to find an exact solution, except for very small environments.

We consider a greedy solution in which linear programming (LP) relaxations are used to find each new schema. Starting from the empty set, we greedily add schemas (columns to W) that have perfect precision and increase recall for the prediction of y (See Algorithm 1 in the Supplementary). In each successive iteration, only the input-output pairs for which the current schema network is predicting an output of zero are passed. This procedure monotonically decreases the prediction error of the overall schema network, while increasing its complexity. The process stops when we hit some predefined complexity limit. In our implementation, the greedy schema selection produces very sparse schemas, and we simply set a limit to the number of schemas to add. For this algorithm to work, no contradictions can exist in the input data (such as the same input appearing twice with different labels). Such contradictions might appear in stochastic environments, and would not be artifacts in real environments, so we preprocess the input data to remove them.

4.3. Planning as Probabilistic Inference

The full Schema Network graph (Fig. 2) provides a probabilistic model for the set of rewards that will be achieved by a sequence of actions. Finding the sequence of actions that will result in a given set of rewards becomes then a MAP inference problem. This problem can be addressed approximately using max-product belief propagation (MPBP) (Attias, 2003). Another option is variational inference. Cheng et al. (2013) use variational inference for planning, but also end up resorting to MPBP to optimize the variational free energy functional. We will follow the first approach.

Without lack of generality, we will consider the present time step to be $t = 0$. The state, action and reward variables for $t \leq 0$ are observed, and we will consider inference over the unobserved variables in a look-ahead window of size[2] T, $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T-1}$. Since the Schema Network is built exclusively of *compatibility factors* that can take values 0 or 1, any variable assignment is either impossible or equally probable under the joint distribution of the graph. Thus, if we want to know if there exists any global assignment that activates a binary variable (say, variable $r_{(+)}^{(t)}$ signaling positive reward at some future time $t > 0$), we should look at the max-marginal $\tilde{p}(r_{(+)}^{(t)} = 1)$. It will be 0 if no global assignment compatible with both the SN and existing observations can lead to activate the reward, or 1 if it is feasible. Similarly, we will be interested in the max-marginal $\tilde{p}(r_{(-)}^{(t)} = 0)$, i.e., whether it is feasible to find a configuration that avoids a negative reward.

At a high-level, planning proceeds as follows: Identify feasible desirable states (activating positive rewards and deactivating negative rewards), clamp their value to our desires by adding a unary potential to the factor graph, and then find the MAP configuration of the resulting graph. The MAP configuration contains the values of the action variables that are required to reach our goal of activating/deactivating a variable. We can also look at S to see how the model "imagines" the evolution of the entities until they reach their goal state. Then we perform the actions found by the planner and repeat. We now explain each of these stages in more detail.

Potential feasibility analysis First we run a feasibility analysis. To this end, a forward pass MPBP from time 0 to time T is performed. This provides a (coarse) approximation to the desired max-marginals for every variable. Because the SN graph is loopy, MPBP is not exact and the forward pass can be too optimistic, announcing the feasibility of states that are unfeasible[3]. Actual feasibility will be verified later, at the backtracking stage.

Choosing a positive reward goal state We will choose the potentially feasible positive reward that happens sooner within our explored window, clamp its state to 1 and backtrack (see below) to find the set of actions that lead to it. If backtracking fails, we will repeat for the remaining potentially feasible positive rewards.

Avoiding negative rewards Keeping the selected positive reward variable clamped to 1 (if it was found in the previous step), we now repeat the same procedure on the negative rewards. Among the negative rewards that have been found as potentially feasible to turn off, we clamp to zero as many negative rewards as we can find a jointly satisfying backtrack. If no positive reward was feasible, we backtrack from the earliest predicted negative reward.

Backtracking This step is akin to Viterbi backtracking, a message passing backward pass that finds a satisfying configuration. Unlike the HMM for which the Viterbi algorithm was designed, our model is loopy, so a standard backward pass is not enough to find a satisfying configuration (although can help to find good candidates). We combine the standard backward pass with a depth-first search algorithm to find a satisfying configuration.

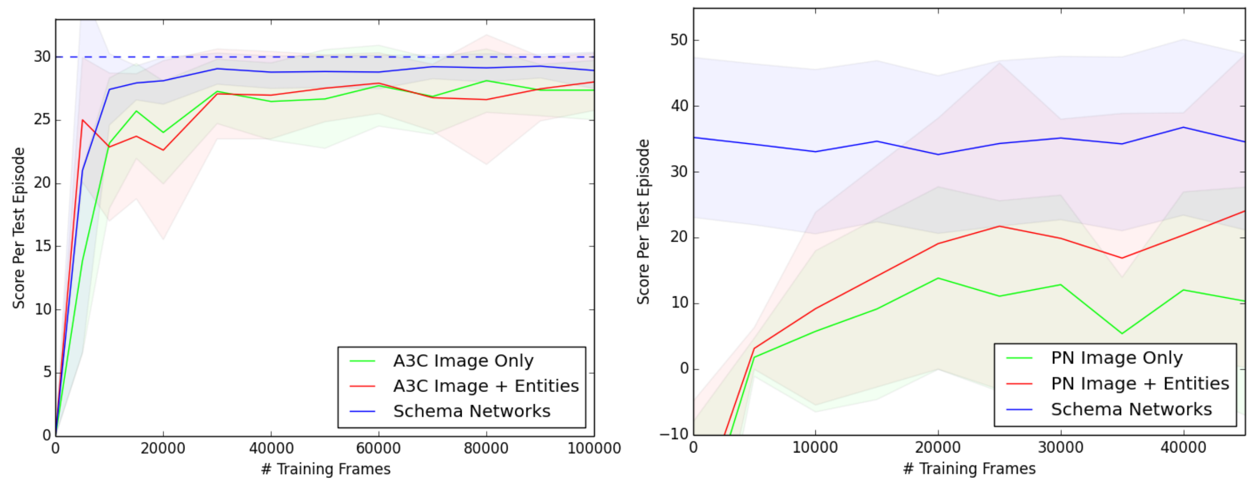
5. Experiments

We compared the performance of Schema Networks, A3C, and PNs (Progressive Networks) on several variations of the game Breakout. The chosen variations all share similar dynamics, but the layouts change, requiring different policies to achieve high scores. A diverse set of concepts must be learned to correctly predict object movements and rewards. For example, when predicting why rewards occur, the model must disentangle possible causes to discover that reward depends on the color of a brick but is independent of the ball's velocity and position where it was hit. While these causal relationships are straightforward for humans to recover, we have yet to see any existing approach for learning a generative model that can recover all of these dynamics without supervision or curriculum and transfer them effectively.

Because Schema Networks rely on an input of entity states instead of raw images, we attempted to provide the same information to A3C and PNs by augmenting the three color channels of the image with 34 additional channels. Four of these channels indicated the shape to which each pixel belongs, including shapes for bricks, balls, walls, and obstacles. Another 30 channels indicated the positions of parts of the paddle, where each part consisted of a single pixel. To reduce training time, we did not provide A3C and PN with part channels for objects other than the paddle, since these are not required to learn the dynamics or predict scores. However, Schema Networks were provided separate entities for each part (pixel) of each object, and each entity contained 78 attributes corresponding to the available part labels (21 for bricks, 30 for the paddle, 25 for obsta-

[2] In contrast with MDPs, the reward is discounted with a rolling square window instead of an exponentially weighted one.

[3] To illustrate the problem, consider the case in which it is feasible for an entity to move at time t to position A or position B (but obviously not both) and then some reward is conditioned on that type of entity being in both positions: A single forward pass will not handle the entanglement properly and will incorrectly report that such reward is also feasible.



(a) Mini Breakout Learning Rate

(b) Middle Wall Learning Rate

Figure 3. Comparison of learning rates. (a) Schema Networks and A3C were trained for 100k frames in Mini Breakout. (b) PNs and Schema Networks were pretrained on 100K frames of Standard Breakout, and then training continued on 45K additional frames of the Middle Wall variation. We show performance as a function of training frames for both models. Note that Schema Networks are ignoring all the additional training data, since all the required schemas were learned during pretraining. For Schema Networks, zero-shot transfer learning is happening.

	Standard Breakout	Offset Paddle	Middle Wall	Random Target	Juggling
A3C Image Only	N/A	0.60 ± 20.05	9.55 ± 17.44	6.83 ± 5.02	-39.35 ± 14.57
A3C Image + Entities	N/A	11.10 ± 17.44	8.00 ± 14.61	6.88 ± 6.19	-17.52 ± 17.39
Schema Networks	36.33 ± 6.17	41.42 ± 6.29	35.22 ± 12.23	21.38 ± 5.02	-0.11 ± 0.34

Table 1. **Zero-Shot Average Score per Episode** Average of the 2 best out of 5 training attempts for A3C, and average of 5 training attempts for Schema Networks. A3C was trained on 200k frames of Standard Breakout (hence its zero-shot scores for Standard Breakout are unknown) while Schema Networks were trained on 100k frames of Mini Breakout. Episodes were limited to 2500 frames for all variations. In every case the average Schema Network scores are better than the best A3C scores by more than one standard deviation.

cles, 1 for walls, and 1 for the ball). Only one of these part attributes was active per entity. In this way, Schema Networks did not treat any object differently, and they were forced to learn that some part attributes, like bricks or obstacles, were irrelevant for predicting the ball's movement. We intentionally provided A3C and PN only with the relevant part information (for the paddle) while ignoring irrelevant information (for other other objects), to give them a strict advantage over the input to the Schema Networks.

5.1. Transfer Learning

This experiment examines how effectively Schema Networks and PNs are able to learn a new Breakout variation after pretraining, which examines how well the two models can transfer existing knowledge to a new task. Fig. 3a shows the learning rates during 100k frames of training on Mini Breakout. In a second experiment, we pretrained on Large Breakout for 100k frames and continued training on the Middle Wall variation, shown in Fig. 1b. Fig. 3b shows that PNs require significant time to learn in this new environment, while Schema Networks do not learn anything new because the dynamics are the same.

5.2. Zero-Shot Generalization

While the moving obstacles variation required additional training to adapt to the new dynamics, many Breakout variations can be constructed that all involve the same dynamics. If a model correctly learns the dynamics from one variation, in theory the others could be played perfectly by planning using the learned model. Rather than comparing transfer with additional training using PNs, in these variations we can compare zero-shot generalization by training A3C only on Standard Breakout and Schema Networks only on Mini Breakout. Fig. 1b-e shows some of these vari-

	Half Negative Bricks
A3C Image Only	5.95 ± 5.53
A3C Image + Entities	-1.75 ± 6.93
Schema Networks	6.34 ± 4.53

Table 2. **Average Score per Episode on Half Negative Bricks** A3C was trained on 200k frames of Random Negative Bricks, and Schema Networks were trained on 100k frames of Mini Random Negative Bricks, both to convergence. Testing episodes were limited to 1000 frames.

ations. Here are brief descriptions:

- **Offset Paddle** (Fig. 1d): The paddle is shifted upward by a few pixels.
- **Middle Wall** (Fig. 1b): A wall is placed in the middle of the screen, requiring the agent to aim around it to hit the bricks.
- **Random Target** (Fig. 1e): A group of bricks is destoyed when the ball hits any of them and then reappears in a new random position, requiring the agent to delibarately aim at the group.
- **Juggling** (Fig. 1f, enlarged from actual environment to see the balls): Without any bricks, three balls are launched in such a way that a perfect policy could juggle them without dropping any.

Table 1 shows the average scores per episode in each Breakout variation. These results show that A3C has failed to recognize the common dynamics and adapt its policy accordingly. This comes as no surprise, as the policy it has learned for Standard Breakout is no longer applicable in these variations. Simply adding an offset to the paddle is sufficient to confuse A3C, which has not learned the causal nature of controlling the paddle with actions and controlling the ball with the paddle. The Middle Wall and Random Target variations illustrate that Schema Networks are aiming to deliberately cause positive rewards from ball-brick collisions, while A3C struggles to adapt its policy accordingly. The Juggling variation is particularly challenging, since it is not clear which ball to respond to unless the model understands that the lowest downward-moving ball is the most imminent cause of a negative reward. By transferring the correct causal dynamics from Mini Breakout, Schema Networks outperform A3C in all variations.

5.3. Testing for Learned Causes

To better evaluate whether these models are truly learning the causes of rewards, we designed one more zero-shot generalization experiment. We trained both Schema Networks and A3C on a Mini Breakout variation in which the color of a brick determines whether a positive or negative reward is received when it is destroyed. Six colors of bricks provide +1 reward, and two colors provide -1 reward. Negative bricks occurred in random positions 33% of the time during training. Then during testing, the bricks were arranged into two halves, with all positive colored bricks on one half and negative colored bricks on the other. If the causes of rewards have been correctly learned, the agent should prefer to aim for the positive half whenever possible. As Table 1 shows, Schema Networks have correctly learned from random arrangements which brick colors cause which rewards, preferring to aim for the positive half during testing, while A3C demonstrates no preference for one half or the other, achieving an average score near zero.

6. Discussion and Conclusion

In this work, we have demonstrated the promise of Schema Networks with strong performance on a suite of Breakout variations. Instead of learning policies to maximize rewards, the learning object for Schema Networks is designed to *understand causality* within these environments. The fact that Schema Networks are able to achieve rewards more efficiently than state-of-the-art model-free methods like A3C is all the more notable, since high scores are a byproduct of learning an accurate model of the game.

The success of Schema Networks is derived in part from the entity-based representation of state. Our results suggest that providing Deep RL models like A3C with such a representation as input can improve both training efficiency and generalization. This finding corroborates recent attempts (Usunier et al., 2016; Garnelo et al., 2016; Chang et al., 2016; Battaglia et al., 2016) to incorporate object and relational structure into neural network-based models.

The environments considered in this work are conceptually diverse but also simplified in a number of ways with respect to the real world: states, actions, and rewards are all discretized as binary random variables; the dynamics of the environments are deterministic; and there is no uncertainty in the observed entity states. In future work we plan to address each of these limitations, adapting Schema Networks to continuous, stochastic domains.

Schema Networks have shown promise toward multi-task transfer where Deep RL struggles. This transfer is enabled by the causal understanding embedded in the networks, which in turn allows for planning in novel tasks. As progress in RL and planning continues, robust generalization from limited experience will be vital for future intelligent systems.

References

Anderson, John R. *Cognitive psychology and its implications*. WH Freeman/Times Books/Henry Holt & Co, 1990.

Attias, Hagai. Planning by probabilistic inference. In *AISTATS*, 2003.

Battaglia, Peter, Pascanu, Razvan, Lai, Matthew, Rezende, Danilo Jimenez, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pp. 4502–4510, 2016.

Chang, Michael B, Ullman, Tomer, Torralba, Antonio, and Tenenbaum, Joshua B. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

Cheng, Qiang, Liu, Qiang, Chen, Feng, and Ihler, Alexander T. Variational planning for graph-based mdps. In *Advances in Neural Information Processing Systems*, pp. 2976–2984, 2013.

Diuk, Carlos, Cohen, Andre, and Littman, Michael L. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 240–247. ACM, 2008.

Drescher, Gary L. *Made-up minds: a constructivist approach to artificial intelligence*. MIT press, 1991.

Garnelo, Marta, Arulkumaran, Kai, and Shanahan, Murray. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

Guestrin, Carlos, Koller, Daphne, Gearhart, Chris, and Kanodia, Neal. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pp. 1003–1010. Morgan Kaufmann Publishers Inc., 2003a.

Guestrin, Carlos, Koller, Daphne, Parr, Ronald, and Venkataraman, Shobha. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003b.

Jaakkola, Tommi S, Sontag, David, Globerson, Amir, Meila, Marina, et al. Learning bayesian network structure using lp relaxations. In *AISTATS*, pp. 358–365, 2010.

Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z, Silver, David, and Kavukcuoglu, Koray. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Jordan, Michael Irwin. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998.

Koller, Daphne and Friedman, Nir. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1928–1937, 2016.

Rusu, Andrei A, Rabinowitz, Neil C, Desjardins, Guillaume, Soyer, Hubert, Kirkpatrick, James, Kavukcuoglu, Koray, Pascanu, Razvan, and Hadsell, Raia. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Scholz, Jonathan, Levihn, Martin, Isbell, Charles, and Wingate, David. A physics-based model prior for object-oriented mdps. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1089–1097, 2014.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Taylor, Matthew E and Stone, Peter. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Usunier, Nicolas, Synnaeve, Gabriel, Lin, Zeming, and Chintala, Soumith. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Weiten, W. *Psychology: Themes and Variations*. PSY 113 General Psychology Series. Cengage Learning, 2012. ISBN 9781111354749. URL https://books.google.com/books?id=a4tznfeTxV8C.