# Causal schema network in model-based reinforcement learning

Andrey Gorodetskiy[1], Alexandra Shlychkova[1], and Aleksandr I. Panov[1,2]

[1] Moscow Institute of Physics and Technology
[2] Artificial Intelligence Research Institute, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences
gorodetskiyandrew@gmail.com
aleksandrashlychkova@gmail.com
panov.ai@mipt.ru

**Abstract.** One of the unresolved problems of machine learning is the inefficiency of transfer learning. One of the mechanisms that are used to solve this problem in the area of Reinforcement Learning is a model-based approach. In the paper we are expanding the schema networks method which allows to extract the logical relationships between objects and actions from the environment data. We present algorithms for training a Causal Schema Network, predicting environmental conditions using a circuit network, as well as an algorithm for selecting actions to achieve the best reward. The results are compared with the results of other implementations of the Schema Network network and Deep Q-Network.

**Keywords:** Schema Network · Causal Network · Reinforcement learning · Model-based reinforcement learning

## 1 Introduction

For an artificial agent acting in real-world conditions, it is necessary to generalize the experience gained in order to not learn from scratch after a slight change of the environment. A human does not relearn the policy of interaction with a familiar object, but only slightly corrects it, when object's characteristics are changed. For this, logical relationships between objects and their characteristics are used at different levels of generalization. For example, in the Atari game *Breakout*, the colors of the bricks do not matter and the natural agent does not change the policy when they change. For an artificial agent, such a generalization is possible using some universal model-based learning algorithm. In this paper, we propose a new approach to the learning of universal models for reinforcement learning in game environments - Causal Schema Network, which is an extension of the early work of Schema Network [1].

A Schema Network is an object-oriented model of the environment that consists of binary schemas [1]. In this architecture the agent receives a frame from the environment, then this frame is divided into a grid of square cells (e.g. pixels). Schemas reflect the interconnection of objects in a certain window of view (several neighboring cells): each schema predicts the reward or the presence or absence of an object of some type in the current cell using information of presence of objects in the corresponding neighboring grid cells in current time.

Due to the binary structure of the schemes, it is possible to represent predictions as a directed graph. A node in this graph is an indicator of the receipt of a reward, the performance of some action or the presence of some object in a certain cell of the grid at some point in time. The edge is drawn in the graph if the value at corresponding position of the corresponding schema is 1. In fact, the edges indicate the presence of a causal relationship between the events "an object of this type is present in the cell" and "received reward." An agent can find a node with a positive reward reachable from the current state of the environment and plan actions to get it.

Since the graph has fairly generalized properties, the trained Schema network can be used in conjunction with the classifier for environments with similar interaction dynamics, which provides advantages in transfer learning.

## 2 Related work

Representation of logical interconnection often helps to increase an efficiency of transfer learning. Various methods are used to represent the logical relationships of objects: in the [1] Schema Network, these are specially introduced binary Schemas with binary logic. In Logical Tensor Networks [7], it is proposed to use real logic. To further apply the obtained relationships for planning, you can provide them as additional data for a neural network. For example, in [6] schemas are passed to a neural network. Authors in [8] add logical relationships to the input of a neural network using logical tensor network. Another approach is to build a dependency graph and search for a reachable state with a positive garbage.

The usage of the Schema Network[1] in reinforcement learning consists of two main points: training a network to predict environmental state and construct a prediction graph with the subsequent search for the best reachable node.

Consider the learning process. The Schema Network uses object-oriented approach described in [3]. Knowing the types of environment objects, a Schema Network is able to identify the logical connections between them. A similar approach was used in Interaction Network [2], for which, however, no planning algorithms were developed to achieve a reward. If we consider the problem of prediction the type of cells (which correspond to the nodes in the graph), then it may be viewed as a classification task. For that problems Convolutional neural networks are used as in [4]. However, this approach requires a priori knowledge about the structure of the graph, while the Schema Network allows to obtain knowledge of the connection of objects automatically from the environment.

A dependency graph is constructed from the trained Schema Network. Finding the reachable state of the environment in which a reward is received can be considered as a degenerate estimate of the posterior maximum and solved using the max-product belief propagation [5].

## 3 Causal Schema Network model

Causal Schema Network (CSN model) learns dynamics of the environment and can be used to predict next states of the environment. At some point of view, CSN represents both transition and reward functions of the environment.

Model gets an image from the environment, which represents an agent's current observation. This image is parsed into entities. Each entity has the same set of properties represented by binary variables called here attributes.

In our implementation there are $M$ different types of objects in the environment. Entity is a pixel of the image, $j$-th attribute of the entity is $True$ if object of $j$-th type crosses the pixel. Each image contains $N$ pixels, hence the number of entities is also $N$. At the time $t$ algorithm gets from environment matrix $s_t$ of shape $(N \times M)$, which represents information about a particular type of object each pixel belongs to. Rows of $s_t$ corresponds to entities and columns to attributes. This matrix is added to frame stack of size 2.

Then in a number of steps we build matrix $X_t$, called augmented matrix. First, $s_t$ is augmented horizontally, by adding to $i$-th entity, that is located in $i$-th row, attributes of its $R-1$ spatial neighbours. This augmentation results in $z_t$ matrix of shape $(N \times MR)$. Performing this operation on each element of the frame stack, finally we construct $X_t$ as horizontal concatenation of $(z_{t-1}, z_t, A_t)$, where $A_t$ is a matrix with same rows, each of which equals to one-hot encoded action $a_t$, that was taken at time $t$.

Then schemas are learned. All schemas are represented by

$$W = (W_i : i = 1..M),$$

where $W_i$ is a matrix of schemas for $j$-th attribute prediction. $w_i^j$ is $j$ schema from $W_i$. $W$ are used for prediction of next state $Y_t$ - matrix of attributes. After planning agent gets action $a_t$ at time $t$. $\{X_u\}_{u=0}^{u=t}$ is transformed to replay memory $(X, Y)$, where lines of $X$ - unique lines of matrices $\{X_u\}_{u=0}^{u=t}$. If $x$ is $j^{th}$ line of $X$ in the moment $t$ then $j$-th line if $Y$ is attributes of entity, which is in the center of $w$.

Schemas represent rules of the environment. Each schema computes next state from certain attributes of entities from previous states using operation AND:

$$Schema\colon (Attributes \cup Actions)^k \to Attributes \cup Rewards$$

If several schemes predict same attribute result is computed with operation OR (see fig.3).

Data preprocessing is performed as follows:

1. Input is picture from the environment in the moment $t$ contains $M$ different types of objects.
2. One-hot encoded type of object is assigned to pixel if that object crosses that pixel
3. Concatenated one-hot encoded vector is line of matrix $X_t$ (see fig.3).

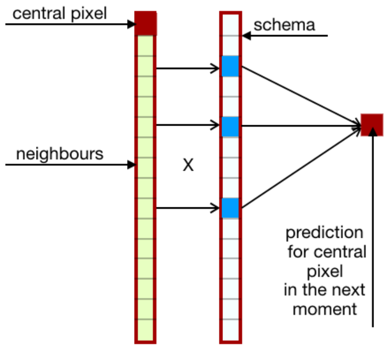

**Fig. 1.** Instantiation graph of schemas using all attributes

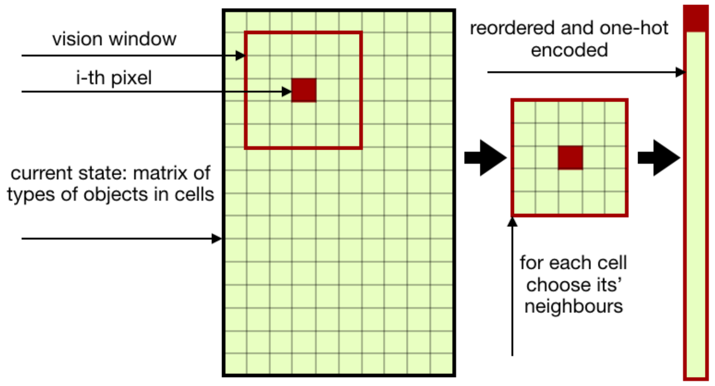
vision window
i-th pixel
current state: matrix of types of objects in cells
reordered and one-hot encoded
for each cell choose its' neighbours

**Fig. 2.** Transformation of data from the environment into learning format

4. $X'_t$ is concatenated $X_t$, $X_{t-1}$ and one-hot index of action (Pic.3).
5. After performing an action agent gets $Y$ from the environment which agent learns to predict.
6. Add to experience replay unique pairs of lines from $X'_t$, $Y_t$ with the same index.
7. Experience replay = $(X, Y)$ .

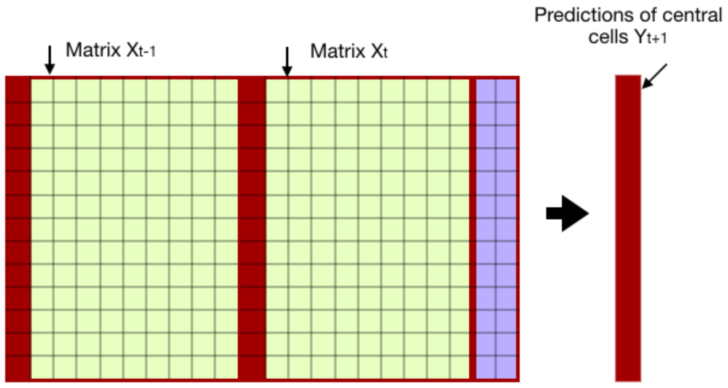

**Fig. 3.** Getting prediction from data

## 4 Learning new schemas

The learning process consists of several stages. Firstly, check the correctness of the existing schemes: try to predict the current state using existing shames. If a certain schema produces a false positive result, then it is incorrect and needs to be deleted. Further, for each type of object we will try to obtain new schemes. If current Schema Net predicts all data for the attribute correctly then no new schemes needed. Otherwise choose one random sample with positive target from data that is not predicted by the current Schema Net and put it in the set *solved*. Then find such schema that does not give a false positive result throughout the entire previous history, predicts 1 for vectors in *solved* and, at the same time, predicts 1 for the maximum number of other vectors from previous history that are not predicted yet. Add all vectors for which new schema predicts 1 to set *solved*. After that find schema with minimal norm which predicts 1 for vectors in *solved* and does not give a false positive result throughout the entire previous history. Depending on the minimization algorithm, the result of an optimization may need to be binarized to be a schema vector. Then we apply a logical filter: some heuristic that allows to make the learning process more effective.

$W_i$ is matrix of schemes in the moment $t$ for prediction of $i^{th}$ attribute. At the moment 0 prediction is $False$. Schema is correct if it does not produce a false positive result. All incorrect schemes are deleted.

```
Input  : X - matrix NT * MR, Y - matrix M * NT)
Output: w - schema
if $\overline{\overline{X}W_i} \neq Y_i$ then
    find $x \in X$, such that $\overline{\overline{x}W_i} \neq Y_i|_x$
    solved = $\{x\}$
else
    return
$w = argmin \sum_{n:Y_n=1} (1 - x_n)w$ and $(1 - x_n)w > 1|y_n = 0$ and
 $(1 - x)w = 0|x \in solved$
$solved = \{x|\overline{\overline{x}W_i} \neq Y_i|_x$ and $\overline{\overline{x}w} = Y_i|_x\}$
$w_{new} = argmin_w|w|_1$ and $(1 - x_n)w > 1|Y_n = 0)$ and $(1 - x)w = 0|x \in solved)$
return $w_{new}$
```

The scheme is binarized as follows: $w = |w| > \alpha$, where $\alpha$ is hyperparameter.

## 5 Planning algorithm

The purpose of planning is to find a sequence of actions that will lead the agent to positive reward.

The input to the planner is the frame stack of size 2 consisting of environment state matrices. The planning process consists of several stages:

1. Potential reachability analysis;
2. Target queuing;
3. Finding the path to the target.

Using the schema vectors obtained at the training stage, the future states of the environment are predicted for $T$ time ticks ahead by a chain of matrix multiplications. Let $s_t^j$ be the $j$-th column of the state matrix $s_t$. Then:

$$s_{t+1}^j = \overline{\overline{X_t}W_j}\vec{1},$$

where $X_t$ is the augmented matrix and $W_j$ is the $j$-th matrix of the schema weights.

When building an augmented matrix, we consider that the agent performs all possible actions at each time step. This allows us to get all the possible options for the next state $s_t$ superimposed on each other in the same matrix. For positively predicted attributes, concrete instances of the corresponding schemas on the factor graph are instantiated.

Having predictions for the future environment states on $T$ ticks ahead, the desired target nodes of the factor graph are selected, upon reaching which the agent will receive a positive reward. Such nodes are added to the target queue.

$$\begin{aligned} q_t &= \text{sorted by time potentially reachable positive reward nodes} \\ &= [r^+_{closest} \dots r^+_{farthest}] \end{aligned}$$

It is required to find the minimal consistent configuration of the $G$ graph with the activated target node. In this configuration, the values of the attribute and reward nodes show their actual reachability. The action nodes $a_t$ represent the actions that must be taken to reach the target node. To find such a configuration, the next node is selected from the target queue and passed to the backtrace_-node(v) function, which returns a value indicating whether such a configuration is found.

```
**Algorithm:** plan_actions
**Input:** $q$ - queue of target nodes
**Output:** actions - sequence of actions to reach the nearest target node (if
         such sequence is found)
while $q$ is not empty do
    target $\leftarrow$ next element from $q$;
    is_success = backtrace_node(target);
    if is_success then
        actions $\leftarrow \{a_i \in G : i \in [1..T]\}$;
        break;
end
```

The `is_reachable` attribute of the graph nodes stores the actual reachability of the node subject to currently selected actions, or `None` if the reachability is not known.

When searching for a path to activate the node at time $t$, schemas are tried in the following order:

1. action independent
2. coinciding with the current constraint on the action at time $t$, if any
3. others

After training stage, some schema vectors may depend on actions, while in the dynamics of the environment there is no such dependence. This occurs because during training events correlated, but did not have a causal relationship.

For correct planning, it is necessary to find a consistent configuration of the graph constructed by prediction on such vectors. It may happen that due to incorrect dependencies it will be impossible to satisfy all of them, and none of the reward nodes will be reachable. These reward nodes turned out to be active in the prediction because all actions nodes were considered active.

```
Algorithm: backtrace_node
Input   : $v$ - target node
          $S_v$ - set of schemas that activate node $v$
Output: actual node reachability, planned actions

$v$.is_reachable ← False;
for $s \in S_{node}$ do
    if $s.is\_reachable$ is None then
        backtrace_schema($s$);
    if $s.is\_reachable$ then
        $v$.is_reachable ← True;
        break;
end
```

```
Algorithm: backtrace_schema
Input   : $s$ - target schema
          $V_s$ - set of nodes, $s$ is conditioned on
Output: actual $s$ reachability

$s$.is_reachable ← True;
for $v \in V_s$ do
    if $v.is\_reachable$ is None then
        backtrace_node(v);
    if not $v.is\_reachable$ then
        $s$.is_reachable ← False;
        break;
end
```

## 5.1 Replanning

Modified `backtrace_node` algorithm can handle these conflicts. We maintain an array of joint constraints on the active action node for each time tick. During graph traversal, we either satisfy these constraints or replan nodes committed to them if there is no other path to the target. The modified algorithm is described as follows:

1. try to activate the node with an action-independent schema
2. if there is no constraint on the current tick, try to activate the node with any schema
3. try to select a schema that satisfies the constraint on the current tick
4. replan all vertices that require current constraint
   – find all actions that being constraints would not prevent the activation of any conflicting node
   – sequentially start replanning path to each conflicting node using the action acceptable by all
   – if all nodes have been replanned successfully, change the constraints at all layers affected by replanning

During the replanning process, a new conflict situation may arise. Then new replanning process should be recursively started.

## 6 Experiments

The model was evaluated on the Atari Breakout game. The goal of the game is to knock down bricks with a ball, substituting a moving platform under it. There are no random factors in the environment.

The action space consists of the following actions: do not move, move left, move right. As an observation, the agent receives an RGB image and information about a particular type of object each pixel belonging to. Rewards are distributed as follows: +1 for knocking down a brick, -1 for dropping a ball past the platform, 0 in other cases.

In this environment, the agent is able to show an adequate game on ∼20 hand-crafted schema vectors. During training, however, vectors do not cover samples in the replay buffer so effectively, and their number was limited to 220 units.

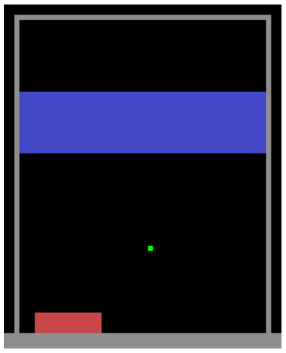

**Fig. 4.** Atari Breakout

The episode was limited to 512 frames. For this amount of frames, one can hit the ball about 7 times without losing it. Figure 5 shows that the agent starts to play successfully after 3 episodes. A distinctive feature of the schema network is the efficient transfer

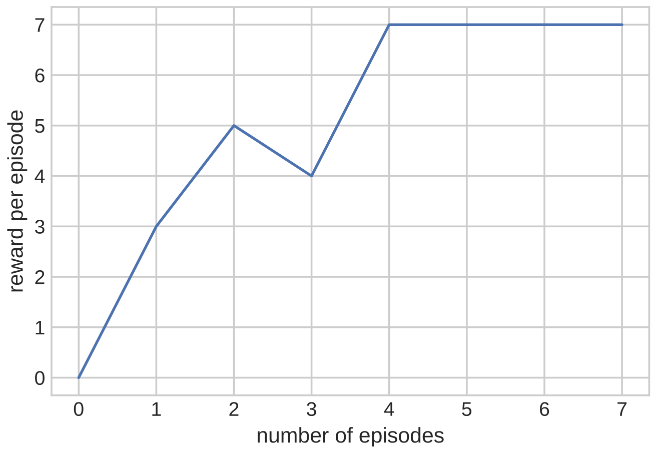

**Fig. 5.** Model evaluation on a standard Breakout

of the trained model to environments with similar dynamics. However, during our experiment of transferring the trained model to the same environment with two balls, some problems of the approach were revealed. When bouncing, the ball can fly off in different directions, depending on the part of the paddle into which it hit and paddle's velocity. At the same time, several balls appear in the predicted future states of the environment, flying in different directions. The model cannot distinguish between balls from different realizations of the future, and considers the nearby balls from different realizations to be the flight path of another ball, predicting its appearance in the next state. This leads to the generation of many virtual balls in the predicted states.

Evaluation on the environment with two balls without additional training showed average score of 7, meaning that agent plays effectively with one ball and catches the second in half of cases. Agent with hard-coded schema vectors showed perfect score.

## 7 Conclusion

In this paper, we proposed an original implementation of the universal logical model of environment dynamics for model-based reinforcement learning. Our approach, which we called Causal Schema Network, is a modification and extension of Schema Network for RL. We described in detail the algorithmic implementation of the proposed method and conducted basic experimental studies on the *Breakout* environment. Code of the model can be obtained in the repository: github.com/cog-isa/schema-rl.

## References

1. Ken Kansky, Tom Silver, David A. Mely, Mohamed Eldawy, Miguel Lazzaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, Dileep George: Schema networks:Zero-shot transfer with a generative causal model of intuitive physics.In ICML., (2017).
2. 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).
3. 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).
4. Kipf, T. N., and Welling, M.: Semi-supervised classification with graph convolutional networks. In ICLR, (2017).
5. Hagai. Attias.: Planning by probabilistic inference. In AISTATS, (2003).
6. Sam Toyer, Felipe Trevizan, Sylvie Thieubaux, Lexing Xie: Action Schema Networks: Generalised Policies with Deep Learning. (2018).
7. Luciano Serafini, Artur d'Avila Garcez: Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge. (2016).
8. Samy Badreddine , Michael Spranger: Injecting Prior Knowledge for Transfer Learning into Reinforcement Learning Algorithms using Logic Tensor Networks (2019).