ToyArchitecture: Unsupervised Learning of Interpretable Models of the World

Jaroslav Vítků*, Petr Dluhoš*, Joseph Davidson, Matěj Nikl, Simon Andersson, Přemysl Paška, Jan Šinkora, Petr Hlubuček, Martin Stránský, Martin Hyben, Martin Poliak, Jan Feyereisl, Marek Rosa *GoodAI*
{name.surname}@goodai.com

***Abstract*—Research in Artificial Intelligence (AI) has focused mostly on two extremes: either on small improvements in narrow AI domains, or on universal theoretical frameworks which are usually uncomputable, incompatible with theories of biological intelligence, or lack practical implementations. The goal of this work is to combine the main advantages of the two: to follow a big picture view, while providing a particular theory and its implementation. In contrast with purely theoretical approaches, the resulting architecture should be usable in realistic settings, but also form the core of a framework containing all the basic mechanisms, into which it should be easier to integrate additional required functionality.**

In this paper, we present a novel, purposely simple, and interpretable hierarchical architecture which combines multiple different mechanisms into one system: unsupervised learning of a model of the world, learning the influence of one's own actions on the world, model-based reinforcement learning, hierarchical planning and plan execution, and symbolic/sub-symbolic integration in general. The learned model is stored in the form of hierarchical representations with the following properties: 1) they are increasingly more abstract, but can retain details when needed, and 2) they are easy to manipulate in their local and symbolic-like form, thus also allowing one to observe the learning process at each level of abstraction. On all levels of the system, the representation of the data can be interpreted in both a symbolic and a sub-symbolic manner. This enables the architecture to learn efficiently using sub-symbolic methods and to employ symbolic inference.

***Index Terms*—Unsupervised, world model, knowledge reuse, hierarchy, interpretable, AGI, planning, reinforcement learning.**

I. Motivation

Despite the fact that strong AI capable of handling a diverse set of human-level tasks was envisioned decades ago, and there has been significant progress in developing AI for narrow tasks, we are still far away from having a single system which would be able to learn with efficiency and generality comparable to human beings or animals. While practical research has focused mostly on small improvements in narrow AI domains, research in the area of Artificial General Intelligence (AGI) has tended to focus on frameworks of truly general theories, like AIXI [43], Causal Entropic Forces [100], or PowerPlay [86]. These are usually uncomputable, incompatible with theories of biological intelligence, and/or lack practical implementations.

Another class of algorithm that can be mentioned encompasses systems that are usually somewhere on the edge of cognitive architectures and adaptive general problem-solving systems. Examples of such systems are: the Non-Axiomatic Reasoning System [96], Growing Recursive Self-Improvers [2], recursive data compression architecture [24], OpenCog [32], Never-Ending Language Learning [14], Ikon Flux [67], MicroPsi [3], Lida [23] and many others [48]. These systems usually have a fixed structure with adaptive parts and are in some cases able to learn from real-world data. There is often a trade-off between scalability and domain specificity, therefore they are usually outperformed by deep learning systems, which are general and highly scalable given enough data, and therefore increasingly more applicable to real-world problems.

Finally, at the end of this spectrum there are theoretical roadmaps that are envisioning promising future directions of research. These usually suggest combining deep learning with additional structures enabling, for example, more sample-efficient learning, more human-like reasoning, and other attributes [51], [62].

Our approach could be framed as something between the ones described above. It is an attempt to propose a reasonably unified AI architecture[1] which takes into account the big picture, and states the required properties right from the beginning as design constraints (as in [8]), is interpretable, and yet there is a simple mapping to deep learning systems if necessary.

In this paper, we present an initial version of the theory (and its proof-of-concept implementation) defining a unified architecture which should fill the aforementioned gap. Namely, the goals are to:

- Provide a hierarchical and decentralized architecture capable of robust learning and inference across a variety of tasks with noisy and partially-observable data.
- Produce one simple architecture which either solves, or has the potential to solve as many of the requirements

We thank Will Millership for his careful reading and thoughtful suggestions for the manuscript.

* Both authors contributed equally to this work.

[1]The term "architecture" is to be taken to mean an autonomous learning and decision system which controls an agent in a virtual/real environment.

for general intelligence as possible[2].
- Emphasize simplicity and interpretability and avoid premature optimization, so that problems and their solutions become easier to identify. Thus the name ***"Toy-Architecture"***.

This paper is structured as follows: first, we state the basic premises for a situated intelligent agent and review the important areas in which current Deep Learning (DL) methods do not perform well (Section II). Next, in Section III, we describe the properties of the class of environments in which the agent should be able to act. We try to place restrictions on those environments such that we make the problem practically solvable but do not rule out the realistic real world environments we are interested in. Section IV then transforms the expected properties of the environments into design requirements on the architecture. In Section V the functionality of the prototype architecture is explained with reference to the required properties and the formal definition in the Appendix. Section VI, presents some basic experiments on which the theoretical properties of the architecture are illustrated. Finally, Section VII compares the ToyArchitecture to existing models of AI, discusses its current limitations, and proposes avenues for future research.

II. Required Properties of the Agent

This section describes the basic requirements of an autonomous agent situated in a realistic environment, and discusses how they are addressed by current Deep Learning frameworks.

1) **Learning:** Most of the information received by an agent during its lifetime comes without any supervision or reward signal. Therefore, the architecture should learn in a primarily unsupervised way, but should support other learning types for the occasions when feedback is supplied.
2) **Situated cognition:** The architecture should be usable as a learning and decision making system by an agent which is situated in a realistic environment, so it should have abilities such as learning from non-i.i.d. and partially observable data, active learning [37], etc.
3) **Reasoning:** It should also be capable of higher-level cognitive reasoning (such as goal-directed, decentralized planning, zero shot learning, etc.). However, instead of needing to decide when to switch between symbolic/sub-symbolic reasoning, the entire system should hierarchically learn to compress high-dimensional inputs to lower-dimensional (a similar concept to the semantic pointer [10]), slower changing [99], and more structured [60] representations. At each level of the hierarchy, the same inference mechanisms should be compatible with both (simple) symbolic and sub-symbolic terms. This refers to one of the most fundamental problems in AI—chunking: how to efficiently convert raw sensory data into a structured and separate format [61], [88]. The system should be able to learn and store representations of both simple and complex concepts to that they can be efficiently reused.
4) **Biological inspiration:** The architecture should be loosely biologically plausible [33], [36], [61]. This means that principles that are believed to be employed in biological networks are preferred (for example in [56]) but not required (as in [54]). The entire system should be as uniform as possible and employ decentralized reasoning and control [19]

[2]That is according to the holistic design principles of [62], [66].

Recent progress in DL has greatly advanced the state of AI. It has demonstrated that even extremely complex mappings can be learned by propagating errors through multiple network layers. However, deep networks do not sufficiently address all the requirements stated above. The problems are in particular:

1) Networks composed of unstructured layers of neurons may be too general; therefore, gradient-based methods have to "reinvent the wheel" from the data for each task, which is very data-inefficient. Furthermore, these gradient-based methods are susceptible to problems such as vanishing gradients when training very deep networks. These drawbacks are partially addressed by transfer learning [73] and specialized differentiable modules [38], [42], [84], [85].
2) The inability to perform explaining-away efficiently, especially in feedforward networks. This starts to be partially addressed by [59], [83].
3) Deep networks might form quite different internal representations than humans do. The question is whether (and if so: how?) DL systems form conceptual representations of input data or rather learn surface statistical regularities [46]. This could be one of the reasons why it is possible to do various kinds of adversarial attacks [91], [93] on these systems.
4) The previous two points suggest that deep networks are not interpretable enough, which may be a hurdle to future progress in their development as well as pose various security risks.
5) The inability to build a model of the world based on a suitable conceptual/localist representation [4], [20], [82] in an unsupervised way leads to a limited ability to reuse learned knowledge in other tasks. This occurs especially in model-based Reinforcement Learning which, for the purposes of this paper, is more desirable than emulating model-free RL [16] owing to its sample efficiency. Solving this problem in general can lead to systems which are capable of gradual (transfer/zero-shot [30], [40]) learning.
6) Many learning algorithms require the data to be i.i.d., a requirement which is almost never satisfied in realistic environments. The learning algorithm should ideally exploit the temporal dependencies in the data. This has

been partially addressed e.g. in [11], [63].

7) One of unsolved problems of AI lies in sub-symbolic/symbolic integration [9], [48]. Most successful architectures employ either just symbolic or sub-symbolic representations. This naturally leads to the situation that sub-symbolic deep networks which operate with raw data are usually not designed with higher-level cognitive processing in mind (although there are some exceptions [15]).

Some of the mentioned problems are addressed in a promising "class" of cortex-inspired networks [13]. But these usually aim just for sensory processing [68], [71], [74], [75], [79], their ability to do sensory-motoric inference is limited [52], or they focus only on sub-parts of the whole model [34].

III. Environment Description and Implications for the Learned Model

In order to create a reasonably efficient agent, it is necessary to encode as much knowledge about the environment as possible into its prior structure—without loss of universality over the class of desired problems. In other words, we are not aiming for an artificial intelligence which is universally general in all possible hypothetical universes (which might not even be possible [101]), but rather for an efficient and multi-purpose machine tailored to a chosen class of environments.

We consider realistic environments with core properties (such as space and time) following from physical laws. The purpose of this section is to describe the assumed properties of the environment and their implications for the properties of the world model. In the following, the process which determines the environment behavior will be called the *Generator*, while the model of this process learned by the agent will be called the *(Learned) Model*.

For simplicity, we first consider a passive agent which is unable to follow goals or interact with the environment using actions. In Section V, we extend both the Model and the Generator by considering actions and reinforcement signals as well. There are multiple properties which we desire of the Generator.

A. Stationarity

The dynamics of the environment is generated by a stationary process (or a non-stationary one which is changing slowly enough for the agent to adapt its learned model to the changes).

B. Non-linearity, Continuity and Partial Observability

Real environments are typically continuous and partially observable. Their Generators can be modeled as general non-linear dynamical systems:

$$\begin{aligned} \dot{x} &= f(x,u) + w, \\ o &= g(x,u) + z, \end{aligned} \tag{1}$$

where the state transition function f and observation function g are nonlinear functions taking state variable x and inputs u as parameters, the $\dot{x}$ is the derivative of x. The function f changes the state variable, while the function g produces observations o which can be perceived by the agent. The terms z and w denote noise [26], [89]. This means that hidden states are not observed directly; rather, they have to be estimated indirectly from the observations o.

C. Non-determinism and Noise

Even though the internal evolution of realistic environments may be deterministic, they are often complex and typically have non-observable hidden states. An observation function o for these environments will thereby impart incomplete information. Additionally, the sensors of the agent are imprecise, and thus there is inherent noise (z in Eq. (1)) so the reading of o (even if it is for a fully observable world) may be flawed. We can model this uncertainty (be it for faulty sensors or non-observability) by expressing the Generator as a stochastic process.

D. Hierarchical Structure and Spatial and Temporal Locality

It is reasonable to expect that the agent will interact with an environment that has many hidden state variables and very complex functions for state-transitions and observations: f and g in Eq. (1). Learning in this setting is not a tractable task in general. Therefore, we will include additional assumptions based on properties of the real world.

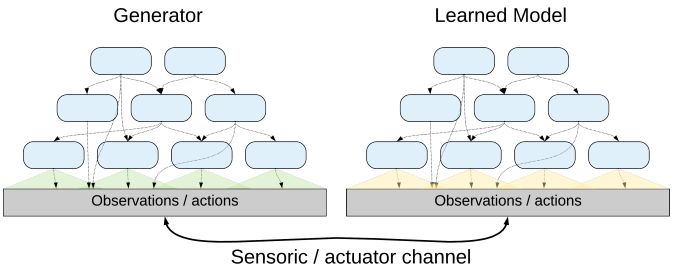


Fig. 1. The Hierarchical Generator (left), which generates spatially and temporally localized observable patterns. The Learned Model in the agent (right) should ideally correspond to the structure of the Generator. The agent's sensors and actuators are localized in the environment as in [18], [26]. Note that in many cases a single observation is a mix of effects of multiple sub-generators running in parallel.

We assume that the Generator has a predominantly hierarchical structure [25], [58], both in space and time; therefore, it can be modeled as Hierarchical Dynamic Model (HDM) [26]. We expect that the observations generated by such system are both local in space (one event influences mostly events which share

similar spatial locations) and in time (subsequent observations share more information than distant ones), as described by the following power law relations:

$$\begin{aligned} I\big(o_i(t); o_j(t)\big) &\propto \mathrm{dist}(i,j)^{-\mathrm{const}}, \\ I\big(o(t); o(t+\Delta)\big) &\propto \Delta^{-\mathrm{const}}, \end{aligned} \tag{2}$$

where $I(x;y)$ is a measure of mutual information between variables x and y, $dist()$ is a spatial distance function appropriate for the particular environment (e.g., Euclidean distance between pixels in an image), Δ is temporal distance, and $const$ is a positive constant. Note that both requirements are not strict and allow sporadic non-hierarchical interactions, interactions between small details in spatially/temporally distant events.

These relations reflect a common property of real world systems—that they have structure on all scales [25], [57]. It can serve as an inductive bias enabling the agent to learn models of environments in a much more efficient way by trying to extract information on all levels of abstraction. These assumptions also reveal an important property that data perceived and actions performed by the agent are highly non-i.i.d., which has to be taken into consideration when designing the agent.

Another important property of such a hierarchy is that at the lowest levels, most of the information (objects, or their properties) should be "place-coded" (e.g. by the fact that a sub-generator on a particular position is active/inactive), but as we ascend the hierarchy towards more abstract levels, the information should be more "rate-coded" in that we keep track of the state of particular sub-generators (e.g. their hidden states or outputs) through time [83]. This means that in higher levels, the representation should become more structured and local.

E. Decentralization and High Parallelism

The spatial locality of the environment implies that on the bottom of the Generator hierarchy, each of the sub-generators influences a spatially localized part of the environment. In realistic environments it is usually true that multiple things happen at the same time. This implies that a single observation should be a mix of results of multiple sub-generators (relatively independent sub-processes/causes) running in parallel, similar to Layered HMMs [69].

IV. Design Requirements on the Architecture

The assumptions about the Generator described in the previous section were derived from the physical properties of the real world. They serve as a set of constraints that can be taken into account when designing the architecture to model these realistic environments. Such constraints should make the learning tractable while retaining the universality of the agent. The goal is to place emphasis on the big picture and high-level interactions within parts of the architecture while still providing some functional prototype. Therefore, individual parts of the presented architecture are as simple and as interpretable as possible. Many of the implemented abilities share the same mechanisms, which results in a universal yet relatively simple system.

The sensors of any agent situated in a realistic environment have a limited spatial and temporal resolution, so the agent is in reality observing a discrete sequence of observations $O = o_1, o_2, \ldots, o_T$, each drawn from an intractable but finite vocabulary $\mathcal{A} = a_1, a_2, \ldots, a_{|\mathcal{A}|}$. Thus, it could be possible to approximate the Generator by a Hidden Markov Model (HMM) with enough states.

An approximation more suitable for the hierarchical structure of the Generator is the Hierarchical Hidden Markov Model (HHMM) [22]. It is a generalization of the HMM where each state is either a production state (a leaf node that emits an observation) or a hidden state which itself represents an HMM. The HMM generates sequences by recursive activation of one of the states in the model (vertical transition) until the production state is encountered. After this, control is handed back to the parent HMM where a horizontal transition is made. The HHMM can be converted into an HMM by concatenating the observation-emitting states and recomputing the transition probabilities. Note that this is a relatively general approach which is similar to Linear Time-Invariant (LTI) dynamical systems [81]. However, the HHMM has two substantial limitations, namely the inability to efficiently reflect (rare) non-hierarchical relationships between subparts (two neighboring sub-processes cannot directly share any information about their states) and its serial nature.

In order to efficiently address the fact that the Generator is parallel (and therefore, each observation can contain results of multiple sub-processes mixed together), the architecture has to be able to learn how to disentangle [8], [39] independent events from each other, and continue to do so on each level of the learned hierarchy.

We will show that the architecture presented in this paper overcomes both aforementioned limitations of HHMM and can efficiently approximate the Generator described in the previous sections. Namely, it can operate in continuous environments (similar to semi-HMMs [7]), but it can also automatically chunk the continuous input into semi-discrete pieces. It can process multiple concurrently independent sub-processes (an example of this is a multimodal sensor data fusion as in Layered HMMs [69]), and can handle non-linear dynamics of the environment. Finally, the architecture presented here can handle non-hierarchical interactions via top-down or lateral modulatory connections, which are often called the context [1], [13], [35], [76].

A. Hierarchical Partitioning and Consequences

Due to the fact that the interactions are largely constrained by space and time, the generating process can be seen as mostly decentralized, and it is reasonable to also create the Learned Model as a hierarchical decentralized system consisting of (almost) independent units, which we call *Experts*. In the first layer, each Expert has a spatially limited field of view—it receives sequences of local subparts of the observations from the Generator (see Fig. 1). The locality assumptions in Eq. (2) suggest that such a localized Expert should be able to model a substantial part of the information contained in its inputs without the need for information from distant parts of the hierarchy.

The outputs of Experts in one layer serve as observations for the Experts in subsequent layers, which have also only localized receptive fields but generally cover larger spatial areas, and their models span longer time scales. They try to capture the parts of the information not modelled by the lower layer Experts, in a generally more abstract and high-level form.

Each Expert models a part of the Generator observed through its receptive field using discrete states with linear and serial (as opposed to parallel) dynamics. In an ideal case, the Expert's receptive field would correspond exactly to one of the local HMMs:

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{A}\mathbf{s}(t), \\ \mathbf{o}(t) &= \mathbf{B}\mathbf{s}(t), \end{aligned} \tag{3}$$

where the $\mathbf{A}$ is a transition matrix and $\mathbf{B}$ is an observation emission matrix.

But in reality, one Expert can see observations from multiple neighboring Generator HMMs, it might not see all of the observations and does not know about the sporadic non-hierarchical connections, so the optimal partitioning of the observations and the exact number of states for each Expert is not known a priori and in general cannot be determined. Therefore, the architecture starts as a universal hierarchical topology of Experts and adapts based on the particular data it is observing. Although all the parameters of the topology and the Experts could be made learnable from data (e.g. the number of Experts, their topology, the parameters of each Expert), we decided to fix some of them (e.g. the topology) or set them as hyperparameters (e.g. the parameters of each Expert). Therefore, the current version of the architecture uses the following two assumptions:

- The local receptive field of each Expert is defined a priori and fixed.
- The number of hidden states of the model in each Expert is chosen a priori and fixed as well.

These assumptions (see Fig. 3) have the following implications:

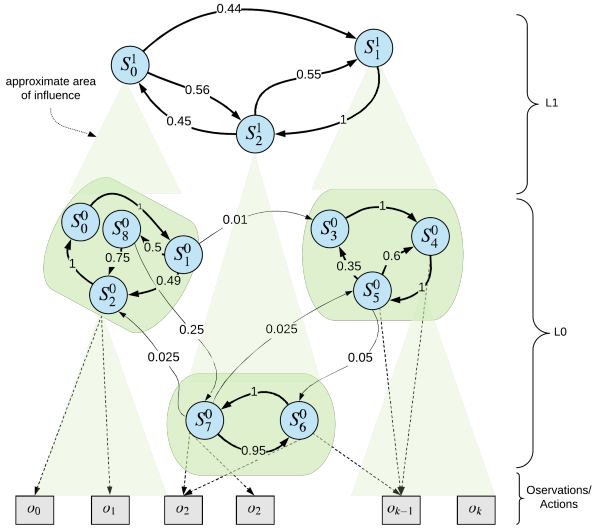


Fig. 2. Example of the hierarchical structure of the world which fulfills the locality in space assumption, and has a fixed number of hidden states. The hierarchy has two levels, in $L0$ there are 3 parallel Markov models, and one in $L1$ on the top. The S_i^j denotes state i in layer j, numbers on the edges are illustrative transition probabilities.

- An Expert might not perceive all the observations that are necessary to determine the underlying sub-process of the Generator responsible for the observations.
- An Expert might not have sufficient resources (e.g. number of hidden states/sequences) to capture the underlying sub-process.

Note that even without the aforementioned assumptions, with the ideal structure and topology of the Experts, their models would not correspond exactly to the Generator until fully learned, which can be impossible to achieve due to limited time and limited information being conveyed via the observations. Therefore, the architecture has to be robust enough so that multiple independent sub-processes of the Generator can be modeled by one Expert, and conversely, multiple Experts might be needed to model one subprocess. Such Experts can then be linked via the context channel (see Appendix A-B). It is a topic of further research whether, and how much, fixing each parameter limits the expressivity and efficiency of the model.

So instead of modelling the input as one HMM as described in Eq. (3), each Expert is trying to model the perceived sequences of observations $\mathbf{o}$ using a predefined number of hidden states $\mathbf{x}$ and some history of length T_h.

Additionally, we define an output projection function computing the output of the Expert $\mathbf{y}$:

$$\begin{aligned} \mathbf{x}(t+1) &= f_1(\mathbf{x}(t), \mathbf{x}(t-1), \ldots, \mathbf{x}(t-T_h)), \\ \mathbf{o}(t) &= f_2(\mathbf{x}(t)), \\ \mathbf{y}(t) &= f_3(\mathbf{x}(t), \mathbf{x}(t-1), \ldots, \mathbf{x}(t-T_h)), \end{aligned} \tag{4}$$

where f_1, f_2 and f_3 are some general functions, $\mathbf{x}(t)$ is the hidden state of the Expert at time t, and $\mathbf{o}(t)$ is the vector of

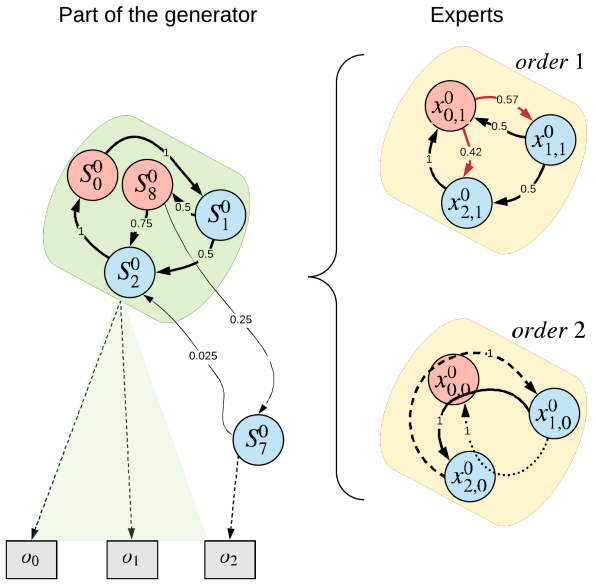


Fig. 3. Approximation of one Markov model from the Generator shown in Fig. 2. Here, one part of the Generator (*green box*) is approximated by two Experts (*yellow boxes*). While both Experts have insufficient number of states, the bottom one mitigates this problem by increasing the order of its Markov chain (parameter T_h in Eq. (4)). The $x^j_{i,k}$ denotes the hidden state i of Expert k in layer j. The top Expert $k = 1$ (which models the process with Markov order 1) shows that the original process cannot be learned well if it has an insufficient number of states (the red Expert states corresponds to the red Generator states S^0_0 and S^0_8 in the original process). Given the state $x^0_{0,1}$, the Expert is unable to predict the probabilities of the next states correctly. Compared to this, the bottom Expert $k = 0$ models the process with Markov order 2 ($T_h = 1$), therefore the probabilities of the next states depend on the current and previous state (indicated by arrows across 3 states in the image). In this case, despite the fact that $x^0_{0,0}$ is ambiguous, the bottom Expert can correctly predict the next state of the original process (for simplicity, transition probabilities are illustrative and not all are depicted).

observations in time t. The output projection function f_3 provides a compressed representation of the Expert's hidden state to its parents, which is then processed as their observations.

We expect that there will be many Experts with highly overlapping (or nearly identical) receptive fields on each layer, which is motivated by the following two points:

- Typically there will be multiple independent processes generating every localized part of the observation vector. So it might be beneficial to model them independently in multiple Experts.
- Since the Experts will learn in an unsupervised way, it is useful to have multiple alternative representations of the same observation $\mathbf{o}$ in multiple Experts. It might even be necessary in practice, since there is no one good representation for all purposes. Other Experts in higher layers can then either pick a lower-level Expert with the right representation for them or use outputs of multiple Experts below as a distributed representation of the problem (which has a higher capacity than a localized one [41]).

B. Resulting Requirements on the Expert

As discussed in the previous section, the local model in each Expert might need to violate the Markov property and will never exactly correspond to a Generator sub-process. Thus, the goal of the Expert is not to model the input observations perfectly by itself, but to process them so that its output data is more informative about the environment than its inputs, and the Experts following in the hierarchy can make their own models more precise.

In order to be able to successfully stack layers of multiple Experts on top of each other, the output of Expert y has to use a suitable representation. This representation has to fulfill two seemingly contradictory requirements:

- It preserves spatial similarity of the input (see e.g. the Similar Input Similar Code (SISC) requirement in [79] or Locality Sensitive Hashing (LSH) [45]). In this case, the architecture should be able to hierarchically process the spatial inputs, even if there is no temporal structure that could be learned[3].
- It should disambiguate two identical inputs based on their temporal (or top-down/lateral) context. The amount of context information added into the output should be weighted by the certainty about this context.

In the current implementation, we address this by converting the continuous observations into a discrete hidden state (based on the spatial similarity), which is then converted again into a (more informative) continuous representation on the output where the continuity captures the information obtained from the context inputs. It does so by working in four steps:

1) **Separation** (disentanglement) of observations produced by different causes (sub-generators). The expert has to partition the input observations in a way that is suitable for further processing. Based on the assumption that values in each part of the observation space are a result of multiple sub-generators/causes (see Section III-E), the Expert should prefer to recognize part of the input generated by only one source. This can be achieved for example via spatial pattern recognition (parts of the observation space which correlate with each other are probably generated by the same source) or by using multiple Experts looking at the same data (see Appendix A-G). Alternative ways to obtain well disentangled representations of observations generated by independent sources are discussed in [39], [90], [95].
2) **Compression** (abstraction, associative learning). Ideally, each expert should be able to parse high-dimensional continuous observations into discrete and localist (i.e. semi-symbolic) representations that are suitable for fur-

[3]Note that in the case where the output of the Spatial Pooler is a one-hot vector, the spatial similarity can be preserved only on the level of multiple experts, which together produce a locality-sensitive binary sparse code representing the input observation(s).

ther processing. This can be done by associating parts of the observation together, which itself is a form of abstraction, and by omitting unimportant details. It is performed based on suitable criteria (e.g. associations of inputs from different sources seen frequently together) and under given resource constraints (e.g. a fixed number of discrete hidden states). This way, the expert efficiently partitions continuous observations into a set of discrete states based on their similarity in the input space.

3) **Expansion** (augmentation). Since the input observations (and consequently the hidden states) can be ambiguous, each Expert should be able to augment information about the observed state so that the output of the Expert is less ambiguous and consists of Markov chains of lower order. This can be resolved e.g. by adding temporal, top-down or lateral context [71].
4) **Encoding**. The observed state augmented with the additional information has to be encoded. This encoding should be in a format which converts spatial and temporal similarity observed in the inputs, and similarity obtained from other sources (context), into a SISC/LSH space of the outputs. thus enabling Experts higher in the hierarchy to do efficient separation and compression.

By iterating these four steps, a hierarchy of Experts is gradually converting a suboptimal or ambiguous model learned in the first layer of Experts into a model better corresponding to the true underlying HMM at a higher level. These mechanisms allow the architecture to partially compensate for the frequent inconsistencies between the hidden Generator and Learned Model topologies. Further improvements could potentially be based on distributed representations and forward/backward credit assignment (similar to [59], [65], [79]).

V. Description of the Prototype Architecture

At a high level, the passive architecture consists of a hierarchy of Experts (where E_i^j denotes i-th expert in j-th layer), whose purpose is to match the hierarchical structure of the world (depicted in Fig. 2) as closely as possible, as described in Section IV-B.

Separation (step 1) is solved on the level of multiple Experts looking at the same data and is described in more detail in Appendix A-G. Unless specifically stated, this version of disentanglement is not used in the experiments described in Section VI.

Compression (step 2) is implemented by clustering input observations $\mathbf{o}(t)$ from a lower level (either another expert or some sensoric input) using the k-means[4]. The Euclidean distance from an input observation to known cluster centers is computed and the winning cluster is then regarded as the hidden state $\mathbf{x}(t)$ (see Eq. (4)). This part is called the *"Spatial Pooler"* (SP) (terminology borrowed from [36]).

The hidden state $\mathbf{x}(t)$ for the current time step is then passed to the next module called the *"Temporal Pooler"* (TP), which performs Expansion (step 3). It partitions the continuous stream of hidden states into sequences of (as in Layered Hidden Markov Models)—Markov chains of some small order $m > 1$, and publishes identifiers of the current sequences and their probabilities. It does so by maintaining a list of sequences and how often they occur. As it receives cluster representations[5] from the SP, the TP learns to predict to which cluster the next input will belong. This prediction is calculated from how well the current history matches with the known sequences, the frequency that each sequence has occurred in the past, and any contextual information from other sources, such as neighboring Experts in the same layer, parent Experts in upper layers, or some external source from the environment.

Encoding (step 4) is implemented via *Output Projection*. The idea is to enrich the winning cluster (what the Expert has observed) with temporal context (past and predicted events). This way, the Expert is able to decrease the order of the Markov chain of recognized states. It is done by calculating the probability distribution over which sequences the TP is currently in, and subsequently, by calculating a distribution over the predicted clusters for the next input. This prediction is combined with the current and past cluster representations to create a *projection* $\mathbf{y}(t)$ over the probable past, present, and future states of the sequence. This projection is passed to the SP of the next Experts in the hierarchy. See Fig. 4 for a diagram illustrating the dataflow.

The TP runs only if the winning cluster in the SP changes which results in an event-driven architecture. The SP serves as a quantization of the input so that if the input does not change enough, the information will not be propagated further.

The context is a one-way communication channel between the TP of an Expert and the TP(s) of the Expert(s) below it in the hierarchy. This context serves two purposes: First, as another source of information for a TP when determining in which sequence it is. And second, as a way for parent Experts to communicate their goals to their children[6], depicted in Fig. 4 and explained in Appendix A-D.

The context consists of three parts: 1) the output of the SP (i.e. the cluster representation), 2) the next cluster probabilities from the TP, and 3) the expected value of any rewards that the architecture will receive if in the next step, the input falls into a particular cluster (interpreted as goals)[7].

[4]In the prototype, we cluster the data via k-means for simplicity and better interpretability, but there are no restrictions on how the compression is performed in general.

[5]In the prototype, this is in the form of a 1-hot vector representing the index of the cluster.

[6]As the parents are not connected directly to the actuators, they have to express their desired high-level (abstract) actions as goals to their children which then incorporate these goals into their own goals and propagate them lower. Experts on the lowest levels of the hierarchy are connected directly to actuators and can influence the environment.

[7]In Fig. 4, the goals are shown as separate from the context for clarity.

In order to influence the environment, the Expert first needs to choose an action to perform, which is the role of the active architecture. The goal is a vector of rewards that the parent expects the architecture will receive if the child can produce a projection $\mathbf{y}(t+1)$ which will cause the parent SP to produce a hidden state $\mathbf{x}(t+1)$ corresponding to the index of the goal value.

An expert receiving a goal context computes the likelihood of the parent getting to each hidden state using its knowledge of where it presently is, which sequences will bring about the desired change in the parent, and how much it can influence its observation in the next step $\mathbf{o}(t+1)$. It rescales the promised rewards using these factors, and adds knowledge about its own rewards it can reach. Then it calculates which of its hidden states lead to these combined rewards. From here, it publishes its own goal Go (next step maximizing the expected reward), and if it interacts directly with the environment picks an action to follow[8].

A much more detailed description of the architecture, its mechanics, and principles can be found in the Appendix.

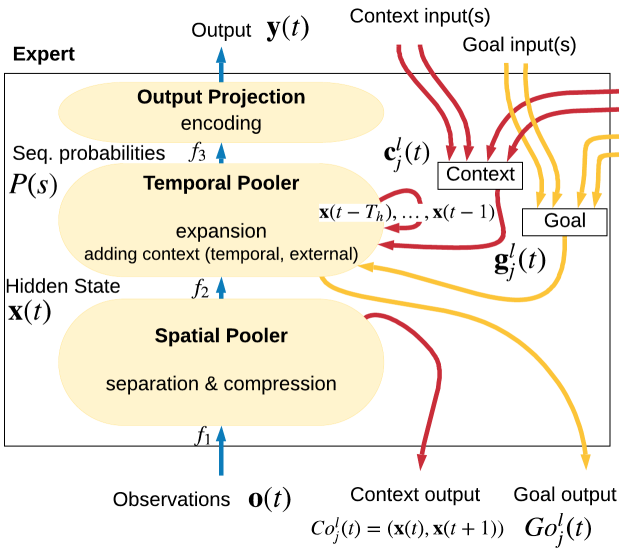


Fig. 4. A High-level description of the Expert and its inputs/outputs. The observations are converted by the Spatial Pooler into a one-hot vector $\mathbf{x}(t)$ representing cluster center probabilities. The Temporal Pooler computes probabilities of known sequences $P(S)(t)$, which are then projected to the output. The external context $\mathbf{c}_j^l(t)$ is received from top-down and lateral connections from other Experts. The corresponding goal vector $\mathbf{g}_j^l(t)$ is used to define a high-level description of the goal state. The Context output of the Expert typically conveys the current cluster center probabilities, while the Goal output represents a (potentially actively chosen) preference for the next state expressed as expected rewards. This can be interpreted as a goal in the lower levels or used directly by the environment (see Appendix A-D).

[8]The action of bottom level Experts at $t-1$ is provided on $\mathbf{o}(t)$ from the environment, so the picking of an action is equivalent to taking the cluster center of the desired state and sampling the actions from the remembered observation(s). See Appendix A-C for more details.

VI. Experiments

This relatively simple architecture combines a number of mechanisms. The general principles of the ToyArchitecture has broad applicability to many domains. This can be seen in the variety of experiments which can be devised for it, from making use of local receptive fields for each Expert, to processing short natural language statements.

Rather than going though all of them, this section will instead show some selected experiments which focus on demonstrating and validating the functionality of the mechanisms described in this paper. The experiments were performed in either BrainSimulator[9] or TorchSim[10]. The source code of the ToyArchitecture implementation in TorchSim is freely available alongside TorchSim itself.

A. Video Compression - Passive Model without Context

We demonstrate the performance of a single Expert by replicating an experiment from [53]. The input to the Expert is the video from the paper with a resolution of $192 \times 192 \times 3$, composed of 434 frames.

The experiment demonstrates a basic setting, where the architecture just passively observes and the data has a linear structure with local dependencies. Therefore, a single Expert is able to learn a model of this data with only the passive model and without needing context, as detailed in Appendix A-A.

The Expert has 60 cluster centers and was allowed to learn 3000 sequences of length 3, where the lookbehind (how far in the past the TP looks to calculate in which sequence it is currently in) is $T_b = 2$ and the lookahead (how many future steps the TP predicts) is $T_f = 1$. Both the SP and TP were learning in an on-line fashion. The video is played for 3000 simulation steps during training (through the video almost 9 times). The cluster centers are initialized to random small values with zero mean.

Fig. 5 shows the course of: reconstruction error, prediction error in the pixel space, and prediction error in the hidden representation[11].

First, the SP learns cluster centers to produce $\mathbf{x}(t)$ given a video frame at time t. In the beginning, only a small number of cluster centers are trained, therefore the winning cluster changes very sporadically (all of the data is chunked into just a small number of clusters). Since the TP runs only if the winning cluster of the SP changes, this results in a situation where the data for the TP changes very infrequently, which

[9]https://www.goodai.com/brain-simulator
[10]https://github.com/GoodAI/torchsim
[11]In all cases the error is computed as the sum of squared differences between the reconstruction (prediction) and the ground truth.

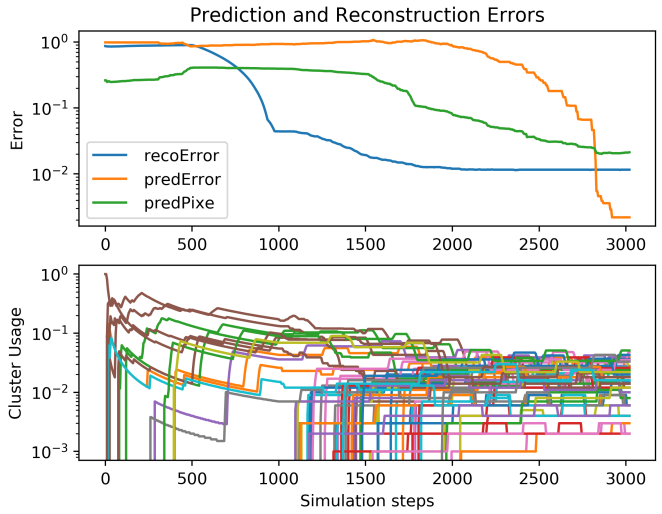
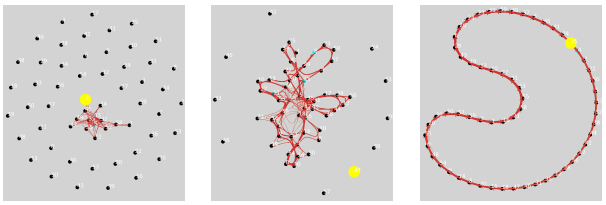


Fig. 5. **Top graph:** reconstruction and prediction errors during the course of on-line training of the Expert on the video. **Bottom graph**: cluster usage (moving window averaged), where each line represents the percentage of time each cluster is active. Both parts of the Expert (the Spatial Pooler and Temporal Pooler) learn on-line (internal training batches are sampled from the recent history). The reconstruction error (in the observation/pixel space: *'recoError'*) decreases first, because the Spatial Pooler learns to cluster the video frames. This causes an overall decrease of prediction error in the observation/pixel space: *'predPixe'*. Note that around step 1700, the prediction error in the observation space decreases, despite the fact that the internal prediction error increases. This is because the changes in the SP representation degrade the sequences learned by the TP. Around step 2000, the learned spatial representation (clustering) is stable (cluster usage shows that all clusters have data) and therefore the inner representation of temporal dependencies starts to improve. Around step 3000, the Temporal Pooler predicts perfectly. The *'predError'* is measured as a prediction error in the hidden space (on the clusters).

means that the TP learns very slowly. This can be seen around step 1000 in Fig. 5, where the reconstruction error converges between 10^{-2} and 10^{-1}. At this point, boosting[12] starts to have an effect, which results in all clusters being used.



(a) Sim. step 1000 (b) Sim. step 1500 (c) Sim. step 3000

Fig. 6. Convergence of the Temporal Pooler's transition probabilities. Each point is a cluster center. The Expert learns sequences of 3 consecutive cluster centers. At the beginning of the simulation, only several cluster centers are used, the Temporal Pooler learns transitions between currently used clusters. Finally, when all the cluster centers are used for some time, the Temporal Pooler converges and learns the linear structure of the video.

The larger the number of clusters in use, the more often the TP sees an event ($\mathbf{x}(t)$ changes), and the more frequently it learns. In the last stage of the experiment, the prediction errors start to converge towards zero.

[12]A mechanism which moves unused cluster centers towards the populated ones with the largest variation in their data points, see Appendix A-A.

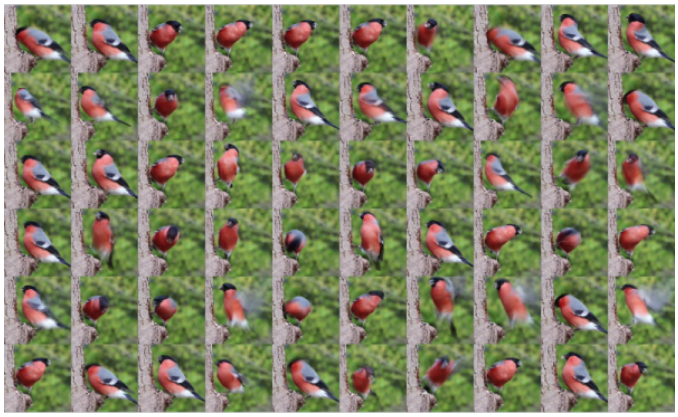


Fig. 7. Resulting cluster centers after learning. Each of the 60 clusters corresponds to approximately 7 frames in the video. The cluster $x(t)$ is active when the nearest 7 frames in the video are encountered. This results in spatial (but also temporal) compression. Note that one Expert is not designed to learn such a large input. Rather, multiple Experts with local receptive fields should typically process the input collaboratively.

This results in a trained Expert, which can recognize the current observation (compute $\mathbf{x}(t)$) and reconstruct it back. The learned spatial representation is shown in Fig. 7 and the convergence of the temporal structure is shown in Fig. 6.

As a result, given two[13] consecutive hidden states $\mathbf{x}(t-1:t)$, the Expert can predict the next hidden state $\mathbf{x}(t+1)$ and reconstruct it in the input space. This process can be seen in a supplementary video[14]. The first part of the video shows how the Expert can recognize and reconstruct current observation and predict the next observation. The second part of the video (48 seconds in) shows the case where the prediction of the next frame is fed back to the input of the Expert. This shows that the Expert can be 'prompted' to play the video from memory. The spatio-temporal compression caused by the clustering and event-driven nature of the TP results in a faster replay of the video as only significant changes in the position of the bird are clustered differently and thus remembered as events by the TP.

Discussion: This experiment demonstrates the capability of on-line learning on linear data of one Expert using the passive model without context. The Expert first learns to chunk/compress the input data into discrete pieces (modelling the hidden space) and then to predict the next state in this hidden space. The prediction can be then fed back to the input of the Expert, which results in replaying meaningful input sequences (where time is distorted by the event-driven nature of the algorithm).

Two important remarks can be made here.

1) The reconstruction error converges to small values fast, but only a small fraction of clusters is used at that

[13]Just the current state would be enough in this experiment, since the learned temporal structure has the Markov property.

[14]Video generated by the Expert: http://bit.ly/2um5zyc

moment. After this first stage, all the clusters start to be used. This change improves the reconstruction error slightly and allows the Temporal Pooler to start learning. This is relevant to [87], where it is argued that the internal structure of the network changes, even if it might not be apparent from the output.

2) The prediction error in the pixel space decreases before the prediction error in the hidden space starts to decrease. The reason for this is that even if the Temporal Pooler predicts a uniform distribution over the hidden states $x(t+1)$ (i.e. the TP is not trained yet), all the cluster centers are moving closer towards the real data and thus the average prediction improves no matter what cluster is predicted.

This experiment shows the performance of an Expert on video, but the same algorithm should process other modalities as well without any changes. It shows a trivial case, where the hidden space just has a linear structure (Markov process). The following experiment extends this to non-Markovian case, where the use of context is beneficial.

B. Audio Compression – Passive Model with Context

This experiment demonstrates a simple layered interaction of three Experts connected in three layers, as depicted in Fig. 8. Its purpose is to demonstrate that top-down context provided by parent Experts helps improve the prediction accuracy at the lower levels.

The setup is the following: Expert E^1 in layer 1 processes[15] the observations $\mathbf{o}(t)$ and computes outputs $\mathbf{y}^1(t)$, the parent Expert E^2 in layer 2 processes the output vector of E^1: $\mathbf{y}^1(t)$ as its own observation and produces the context vector. This context vector is used by the E^1 to improve the learning of its own Temporal Pooler as described in Appendix A-B. The same is done for the third layer.

The input data to the architecture is an audio file with a sequence of spoken numbers 1-9. The speech is converted by Discrete Fourier Transform into the frequency domain with 2048 samples. Each time step, one sample with 2048 values is passed as an observation $\mathbf{o}(t)$ to E^1. The original audio file is available on-line[16].

All the Experts in the hierarchy share the same number of available sequences (2000), and the lookahead $T_f = 1$. The Expert E^1 which processes raw observations has 80 cluster centers and a sequence length $m = 3$. Its parent Expert E^2 has 50 cluster centers with a sequence length $m = 5$. The most abstract Expert E^3 has 30 clusters and can learn sequences of length $m = 5$.

[15]Normally, E_i^j denotes the i-th Expert in the j-th layer. Since this experiment uses just one Expert per layer, the subscript i will be omitted for clarity.

[16]Original audio file with labels: http://bit.ly/2HxdTUA

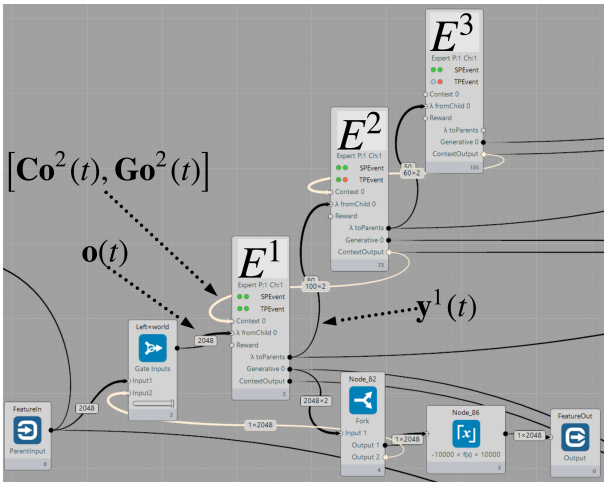


Fig. 8. Setup of the experiment with context. E^1 receives the feature vectors on the $\mathbf{o}(t)$ input and receives the context vector $[\mathbf{Co}^2(t), \mathbf{Go}^2(t)]$ from its parent E^2, which helps it to resolve uncertainty in the Temporal Pooler. The same holds for higher level(s).

The results of a baseline experiment with just the bottom Expert E^1 are shown in Fig. 9. After training both Spatial and Temporal Poolers, it can be seen that the sequences of hidden states $\mathbf{x}^1(t)$ are highly non-Markovian (Fig. 13(a)). The order of the Markov chains is higher than the supported maximum of $m - 1 = 2$. After connecting the prediction[17] to the Expert's input as a new observation $\mathbf{o}(t)$, the Expert is almost able to reconstruct two words, but is stuck in a loop of these two[18]. The reason of is that many sequences are going through several clusters which correspond to relative silence. In these states, the Expert does not have enough temporal context to determine in which direction to continue.

But if we connect several Experts in multiple layers above each other, the parent Experts provide temporal context $Co^l(t)$ to the Experts below. Since the Experts higher in the hierarchy represent the process as a Markov chain of lower order (see Fig. 13), the context vector provided by them serves as extra information according to which the low-level Expert(s) can learn to predict correctly. Due to the event-driven nature of each Expert, the hierarchy naturally starts to learn from the low level towards the higher ones. Once learned, the average prediction error on the bottom of the 3-layer hierarchy is lower compared to the baseline 1-Expert setup.

After connecting the bottom Expert in a closed loop, like in the previous experiment, the entire hierarchy is able to replay the audio correctly. The resulting audio can be found on-line[19] and the representation is shown in Fig. 13.

Figures 10, 11, and 12 show the convergence of the Spatial and Temporal Poolers for each Expert in the hierarchical setting.

[17]In this simulation, the GreedyWTA function was applied on the prediction.

[18]The audio generated by one Expert without context available is located at: http://bit.ly/2W7OXpO, after some time the prediction starts failing.

[19]Audio generated by a hierarchy of 3 Experts: http://bit.ly/2FrnFWg.

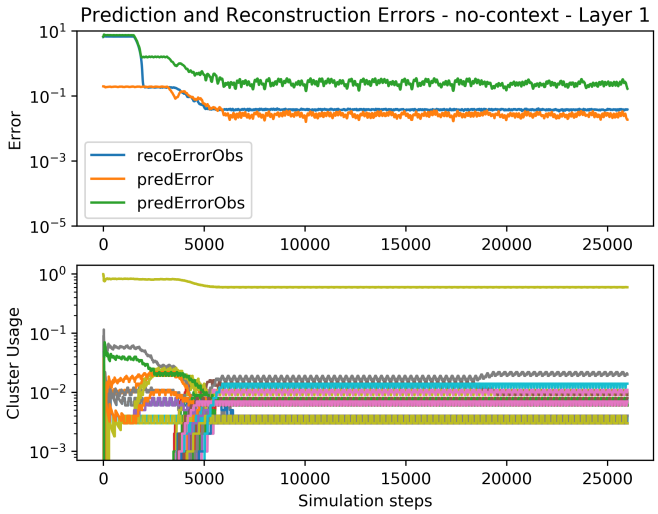


Fig. 9. Convergence of the Spatial Pooler's reconstruction error (*recoErrorObs*) and Temporal Pooler's prediction error both in the observation (*predErrorObs*) and hidden (*predError*) space. The graph below shows cluster usage in time: one of the clusters is used much more often, probably representing a silent part. The prediction error converges to a relatively high value, since the Expert is unable to learn the model correctly by itself.

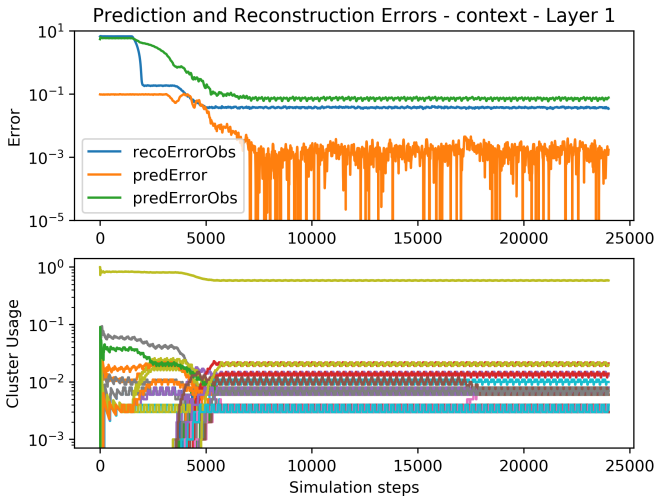


Fig. 10. Error and cluster usage charts tracking the convergence of the bottom Expert E^1 in the presence of the context signal. Compare with the baseline in Fig. 9 which does not use the context. See Fig. 9 for a description of the plotted lines. Since the processing of the SP is not influenced by the context, the SP works identically as in the baseline case (e.g. the cluster usage and SP outputs are the same in both experiments).

The Spatial Pooler in the bottom layer behaves identically as in Fig. 9, but here, the Temporal Pooler can use the top-down context to decrease its prediction errors significantly. The cluster usage graphs show the effect of increasingly abstract representations. In layers 2 and 3, there is no explicit cluster for silence as in the first layer, because those silences cannot be used to predict the next number, and so are disregarded.

Note that the event-driven processing in the Experts, the architecture implements adaptive compression in the spatial and temporal domain on all levels. This is exhibited as either

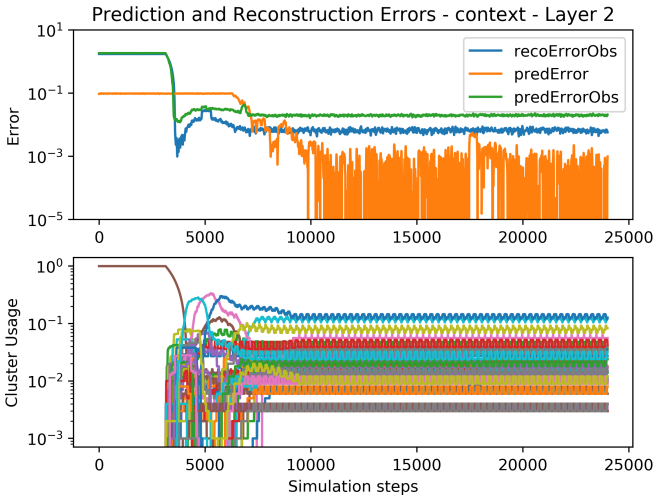


Fig. 11. Error and cluster usage charts tracking the convergence of E^2. See Fig. 9 for a description of the plotted lines.

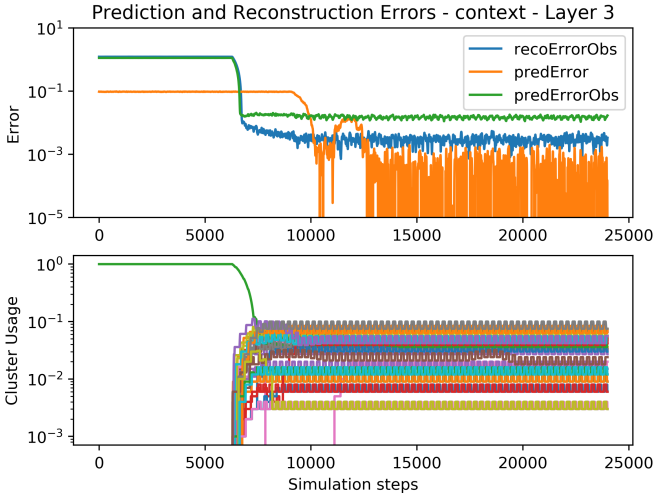


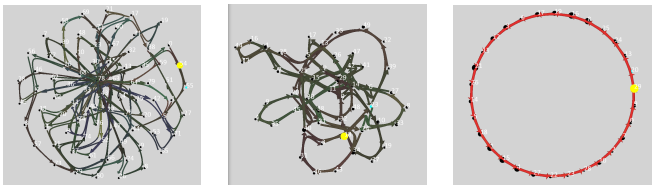
Fig. 12. Error and cluster usage charts tracking the convergence of E^3. See Fig. 9 for a description of the plotted lines.

speeding up the video in the preceding experiment, or speeding up the resulting generated audio in this experiment.

Discussion: The experiment has shown how the context can be used to extendthe ability of a single Expert to learn longer term dependencies. It has also shown that the hierarchy works as expected: higher layers form representations that are more compressed and have lower orders of Markov chains. The activity on higher layers can provide useful top-down context to lower layers, and these lower layers can leverage it to decrease their own prediction error.

C. Learning Disentangled Representations

This experiment illustrates the ability of the architecture to learn disentangled representations of the input space. In other words, this is the ability to recover hidden independent



(a) Transitions in E_1^1 (b) Transitions in E_1^2 (c) Transitions in E_1^3

Fig. 13. Resulting learned sequences in all the Experts. It can be seen how the output projections to $y^i(t)$ help to adaptively compress predictable parts of the input. The higher in the hierarchy, the lower the order of the Markov chain the Experts process. On the top of the hierarchy, the order is 1 and for the Expert E_1^2 the sequence of hidden states has a linear structure.

generative factors of the observations in an unsupervised way. Such an ability may be vital for learning grounded symbolic representations of the environment [8], [39]. In the prototype implementation, the ability to disentangle the generative factors is implemented via a predictive-coding-inspired mechanism (described in Appendix A-G), and is limited only to the input being created by an additive combination of the factors.

The experiment shows how a group of two Experts can automatically decompose the visual input into independently generated parts of the input. And to naturally learn about each of them separately, without any domain-specific modifications.

The input is a sequence of observations of a simple gray-scale version of the game pong (shown in the top left in Fig. 14). The ball moves on realistic trajectories and the paddle is moved by an external mechanism so that it collides with the ball around 90% of the time.

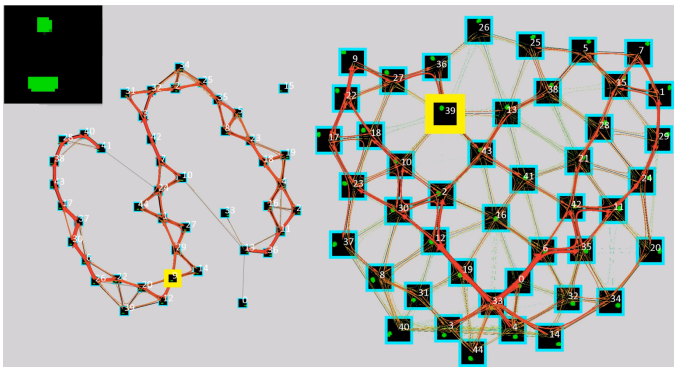


Fig. 14. **Top left:** the current visual input (pong, with ball and paddle). The image shows the representation of two objects learned in an unsupervised way by two Experts competing for the same visual input. The learning automatically decomposes the observations into two independent parts. The independent parts in this case correspond to the paddle (left) and the ball (right). By representing each object in a separate Expert, each is able to learn the simple temporal structures governing the behavior of its object independently of the other, leaving the learning of structures resulting from the interaction of the objects to higher and more abstract layers. From the representation it can be easily seen that the paddle moves just in one axis (linear structure discovered by the TP), while the ball moves through the entire 2D space (grid). The current position of the ball and the paddle are shown in yellow, each cluster center is overlaid with the visual input it represents.

The experiment shows how a simple competition of two Experts for the visual input can lead to the unsupervised decomposition of observations into independent parts. Here, there are two mostly independent parts on the input, therefore the Spatial Pooler of one Expert represents one part (paddle), the other Expert the other part (ball). The resulting representations are shown in Fig. 14. The rest of the architecture works without any modification, therefore each of the Temporal Poolers learn the behavior of just a single object[20]. Representing states of each of the objects independently is much more efficient than representing each state of the scene at once.

Discussion: Although this simple mechanism is not as powerful as DL-based approaches [39], it is interpretable and considerably simpler. It was experimentally tested that such a configuration is able to disentangle up to roughly 6 independent sources of input. In case the number of latent generative factors of the environment is smaller than the number of competing Experts $M < N$, then the group of Experts forms a sparse distributed representation of the input. It is a topic for further research if application of this simple mechanism on each layer of the hierarchy[21] could overcome its limitations and achieve results comparable to deep neural networks.

D. Simple Demo of Actions

The purpose of this experiment is to show the interplay of most of the mechanisms in the architecture. A small hierarchy of two Experts has to learn the correct representation of the environment on two levels of abstraction, then use this representation to explore, discover a source of reward, learn its ability to influence the environment through actions and then collect the rewards. The passive model works identically to the previous experiments, and addition the active parts of the model are enabled. Moreover, all the active parts of the model should be backwards compatible, which means that this configuration of the network should work also on the previous experiments, even though there are no actions available.

This experiment uses a hierarchy of two Experts to find and continuously exploit reward in a simple gridworld. Each time the agent obtains the reward, its position is reset to a random position where there is no reward. The reward location is fixed, but visually indicated. The agent must therefore explore tiles to find it and remember the position. Fig. 15 pictures the initial state of the world.

The agent itself consists of two Experts connected in a narrow hierarchy similar to the one depicted in Fig. 8. Expert E^1 has 44 cluster centers, a sequence length of 5 and lookahead of 3, and E^2 has 5 clusters with 7 and 5 for sequence length and lookahead respectively. As stated in appendices A-C and A-D

[20]See the illustrative video of the inference at http://bit.ly/2CvXnQv

[21]As with each mechanism in the ToyArchitecture, we expect the workload to be distributed among all Experts, closely interacting with other mechanisms, and performed using simple algorithms rather than being localized in one part of the architecture and solving the problem all at once.

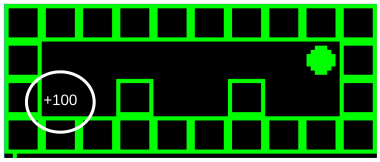


Fig. 15. The initial state of the world, with the agent represented as the green circle and the reward tile highlighted by the authors.

the agent sees the action on the input (the one pixel tall 1-hot vector in the bottom left of Fig. 15), and all levels receive reward (100, in this case) when the agent steps onto the reward tile.

With a lookahead of 3, E^1 can 'see' the reward only 2 actions into the future (the reward is given when the agent is reset, so it is effectively delayed by 1 step). Expert E^2 meanwhile clusters sequences from E^1, so that it has a longer 'horizon' over which to see. Expert E^2 therefore has to guide E^1 to the vicinity of the reward tile by means of the context and goal vectors.

The results of 10 independent runs measured by average reward per step is presented in Fig. 16. As one would expect the average reward increases as time goes on, indicating that the agent has learned where the reward is, and is actively following its learned path to that reward.

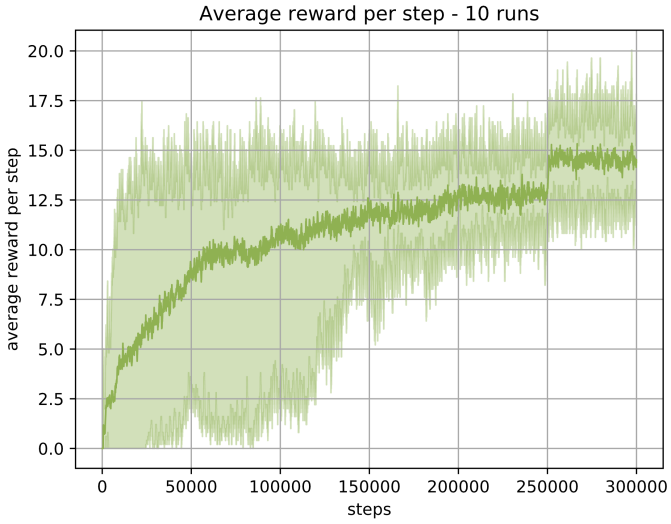


Fig. 16. The average (minimum and maximum) collected reward per step across 10 runs. Learning and exploration was disabled after the step 250,000.

A particularly good example of E^2 clustering is in Fig. 17. This shows that E^2 had created clusters where temporally contiguous projections from E^1 are spatially clustered together. So that if we were to overlap these 5 images there would be a contiguous 'line' of agent positions from anywhere in the environment to right beside the reward tile.

Discussion: This experiment demonstrates that the hierarchical exploration and goal-directed mechanisms are functional and, when trained appropriately, allow an Expert hierarchy to find rewards and follow goals. However, when the clustering is done poorly (as has been the case for at least one run of the experiment), the model encounters a lot of difficulty. Since the model is constantly learning, the cluster centers might find a global (local) optima or continuously drift in time. Therefore, incentivising a 'good' clustering without domain specific knowledge is currently an open question and will be mentioned further in Section VII.

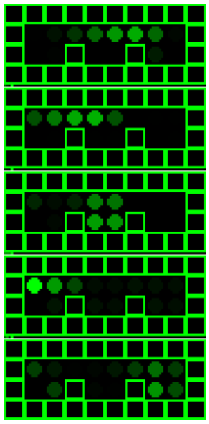


Fig. 17. An interpretation of the clusters of E^2, projected through the clusters of E^1 and into the input space. Expert E^2 clusters spatial and temporal information from E^1, so its clusters represent a superposition of states of E^1.

E. Actions in Continuous Environments

The current design of the architecture supports not only discrete environments, but was also tested in continuous environments with continuous actions. The last experiment serves as a simple illustration of this and is similar to another experiment of the authors of [53][22]

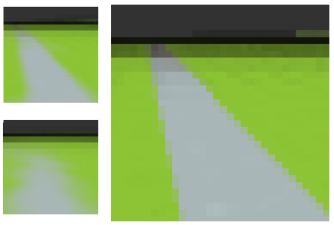


Fig. 18. An example of first-person visual input to the Expert. **Right:** current visual input; **top left:** reconstruction of the current cluster (the part which corresponds to the visual input); **bottom left:** reconstruction of selected next cluster center (the part which corresponds to the visual input, the other part is taken as an action to be executed). The Expert is predicting that it will turn left in the next step, and therefore the track will correspondingly be more in the center of the visual field.

The environment is a simple first-person view of a race track. The goal is to stay on the road and therefore to drive as fast as possible.

[22]Link to the video of the original experiment: https://bit.ly/2XVZmXF.

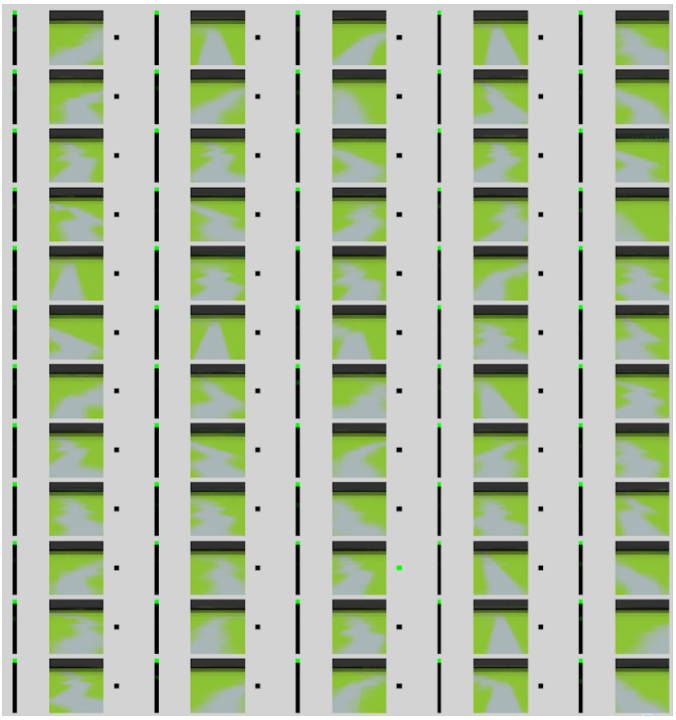


Fig. 19. An example of cluster centers learned in E^1 after puppet-learning (showing ground-truth action). The task can be solved pretty well by a reactive agent (stimulus $\rightarrow$ response policy). As a consequence of this, each cluster center represents some visual input and its corresponding learned action. Training in a RL setting, where the reward is given for staying on the road, leads to very similar cluster centers.

The topology is composed of just one Expert E^1 which receives a visual image and a continuous action (the top bit is forward, and then there are barely visible slight turning actions below) stacked together.

Discussion: The single Expert was able to learn to drive on a road in a so called puppet-learning setting, where the correct (optimal) actions are shown (a human drove through the track manually several times). But it was also able to learn correct behavior in a RL setting, where just the visual input and a reward signal (for staying on the road) was provided. Despite the fact that the learned representation is simple and seems to be on the edge of memorization, the agent was able to generalize well and was able to navigate also on previously unseen tracks (with the same colors). An example of agent autonomously navigating in the racing track is online[23].

These five experiments suggest that hierarchical extraction of spatial and temporal patterns is a relatively domain-independent inductive bias that can create useful models of the world in an unsupervised manner, forming a basis for sample efficient supervised learning. The same basic architecture has been tested on a variety of tasks, exhibiting non-trivial behaviour without requiring domain specific information, nor huge volumes of data on which to train.

[23]Autonomous navigation of the agent on the race track: http://bit.ly/2OgkVO5.

VII. Discussion and Conclusions

This paper has suggested a path for the development of general-purpose learning algorithms through their interpretability. First, several assumptions about the environments were made, then based on these assumptions a decentralized architecture was proposed and a prototype was implemented and tested. This architecture attempts to solve many problems using several simple and interpretable mechanisms working in conjunction. The focus was not on performance on a particular task, it was rather on the generality and the potential to provide a platform for sustainable further development.

We presented one relatively simple and homogeneous system which is able to model and interact with the environment. It does this using the following mechanisms:

- extraction of spatio-temporal patterns in an unsupervised way,
- formation of increasingly more abstract and more informative representations,
- improvement of predictions on the lower levels by means of the context provided by these abstract representations,
- learning of simple disentangled representations,
- production of actions and exploration of the environment in a decentralized fashion,
- and hierarchical, decentralized goal-directed decision making in general.

A. Similar architectures

There are many architectures/algorithms which share some aspects with the work presented here. The similarities can be found in the focus on unsupervised learning, hierarchical representations, recurrence in all layers, and the distributed nature of inference.

The original inspiration for this work was the PhD Thesis "How the Brain Might Work" [18]. The hierarchical processing with feedback loops in ToyArchitecture is similar to *CortexNet* [13], a class of networks inspired by the human cortex. There are also a lot of architectures that are more or less inspired by predictive coding [6], [90], but they are focused on passively learning from the data.

Many of these architectures are implemented in ANNs, using the most common neuron model. They are often similar in their hierarchical nature, such as the Predictive Vision Model [77]; a hierarchy of auto-encoders predicting the next input from the current input and top-down/lateral context. More recently, the Neurally-Inspired Hierarchical Prediction Network [74] uses convolution and LSTMs connected in a predictive coding setting. Several publications try to gate the LSTM cells in a manner inspired by cortical micro-circuits [72].

There are more networks that are loosely inspired by these concepts. The main idea is usually in the ability to have some objective in all layers, enabling the network to produce intermediate gradients which improves convergence and robustness. Examples of these are Ladder Networks [75], or the Depth-gated LSTM [102].

There are also networks that use their own custom model of neurons. These include the Hierarchical Temporal Memory (HTM) [34], the Feynman Machine [52] or Sparsey [78].

A model inspired by similar principles was also able to solve CAPTCHA. It is the Recursive Cortical Network (RCN) [59]. It works on visual inputs that are manually factorised into shape and texture. Compared to other architectures mentioned here, it is based on probabilistic inference and therefore is closer to the hypothesis that the brain implements Bayesian inference [55].

There are fewer architectures that are also focused on learning actions. An example of a system implemented using deep learning techniques is Predictive Coding-based Deep Dynamic Neural Network for Visuomotor Learning [44]. It learns to associate visual and proprioceptive spatio-temporal patterns, and is then able to repeat the motoric pattern given the visual input. The Feynman Machine was also shown to learn and later execute policies taught via demonstration [53]. Despite the fact that both of the architectures are able to learn and execute sequences of actions, none of them currently support autonomous active learning. In contrast to the ToyArchitecture, the mechanisms for exploration and learning from rewards are missing. An architecture emphasizing the role of actions and active learning in shaping the representations is [37]. Similarly to the ToyArchitecture, actions are part of the concept representation and not just the output of the architecture.

A more loosely bio-inspired architecture is World Models [30]. These combine VAE for spatial compression of the visual scene, RNNs for modeling the transitions between the world states, and a part which learns policies. Compared to the ToyArchitecture, this structure is only has a single layer (just one latent representation) and learns its policies using an evolutionary-based approach. Here, the interesting aspect is that after learning the model of the environment, the architecture does not need the original environment to improve itself. It instead 'dreams' new environments on which to refine its policies.

Another deep learning approach focused on a universal agent in a partially observable environment is the MERLIN architecture [97]. Based on predictive modelling, it tries to learn how to store and retrieve representations in an unsupervised manner, which are then used in RL tasks. Unlike the ToyArchitecure, it is a flat system where the memory is stored in one place instead of in a distributed manner.

B. Limitations and Future Work

Despite promising initial results, the theory is far from complete and there are many challenges ahead. The performance of the model is partially sacrificed for interpretability, and in the current (purely unsupervised or semi-supervised setup) it is far behind its DL-based counterparts. It seems that the current biggest practical limitation of the model is that the Experts do not have efficient mechanisms to make the representation in other Experts more suitable for their own purposes (i.e. a mechanism which implements credit assignment through multiple layers of the hierarchy). There are some potentially promising ways how to improve this (either based on an alternative basis [79], a DL-framework [74] or a probabilistic one [59]).

Another way to scale up the architecture would be to use multiple Experts with small, overlapping receptive fields (as discussed in Section IV-A), ideally in combination with a mechanism efficiently distributing the representations among them (see Appendix A-G). Our preliminary results (not presented in this paper) show that such redundant representations can not only increase the capacity of the architecture [41], but also provide a population for evolutionary based algorithms of credit assignment.

During development, empirical evidence suggested that a better form of lateral coordination (lateral context between Experts) is missing in the model, especially in the case of wide hierarchies with multiple experts on each layer processing information from local receptive fields. Examples of this can be seen in [71] and [59].

Some mechanisms to obtain a grounded symbolic representation of the environment were tested in the form of disentanglement. It is not clear now whether these mechanisms would be scalable all the way towards very abstract conceptual representations of the world, or if there is something missing in the current design which would support abstract reasoning.

One of the big challenges in designing complex adaptive systems is in life-long or gradual learning; i.e. the ability to accumulate new non-i.i.d. knowledge in an increasingly efficient way [80]. The system has to be able to integrate new knowledge into the current knowledge-base, while not disrupting it too much. It should also be able to use the current knowledge-base to improve the efficiency of gathering new experiences. So despite that some of these topics are partially covered by the architecture (decentralized system, natural reuse of sub-systems in the hierarchy, event-driven nature of the computation mitigating forgetting), there are still many open questions that need to be addressed.

REFERENCES

[1] Rick A Adams, Stewart Shipp, and Karl J Friston. Predictions not commands: active inference in the motor system. *Brain Structure and*

Function, 218(3):611—-643, 2013.

[2] Bas R. Steunebrink And, Kristinn R. Thorisson And, and Jurgen Schmidhuber. Growing Recursive Self-Improvers. In *Artificial General Intelligence - 9th International Conference, AGI 2016, New York, NY, USA, July 16-19, 2016, Proceedings*, volume 7716, pages 1–11, 2016.

[3] Joscha Bach. The MicroPsi Agent Architecture. *Proceedings of ICCM5 International Conference on Cognitive Modeling Bamberg Germany*, 1(1):15–20, 2003.

[4] Joscha Bach. Representations for a Complex World: Combining Distributed and Localist Representations for Learning and Planning. Technical report, University of Osnabrück, 2005.

[5] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-Critic Architecture. Technical report, School of Computer Science McGill University, 2016.

[6] Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. Canonical Microcircuits for Predictive Coding. 2012.

[7] Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, dec 1966.

[8] Yoshua Bengio. The Consciousness Prior. *arXiv preprint arXiv:1709.08568*, abs/1709.0, sep 2017.

[9] Tarek R Besold, Artur D ', Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado, Vieira Lima, Illuminoo B V Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-Symbolic Learning and Reasoning Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. *arXiv preprint arXiv:1711.03902*, abs/1711.0, 2017.

[10] Peter Blouw, Eugene Solodkin, Paul Thagard, and Chris Eliasmith. Concepts as Semantic Pointers: A Framework and Computational Model. *Cognitive Science*, 40(5):1128–1162, jul 2016.

[11] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-Free Episodic Control. *arXiv preprint arXiv:1606.04460*, pages 1–12, 2016.

[12] Nicolas Brodu. Reconstruction of Epsilon-Machines in Predictive Frameworks and Decisional States. *Advances in Complex Systems*, 14(05):761—-794, 2011.

[13] Alfredo Canziani and Eugenio Culurciello. CortexNet: a Generic Network Family for Robust Visual Temporal Representations. *arXiv preprint arXiv:1706.02735*, abs/1706.0(1), 2017.

[14] Andrew Carlson, Justin Betteridge, and Bryan Kisiel. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pages 1306–1313, 2010.

[15] Xuan Choo and Chris Eliasmith. General Instruction Following in a Large-Scale Biologically Plausible Brain Model. In *35th Annual Conference of the Cognitive Science Society*, pages 322–327. Cognitive Science Society, 2013.

[16] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A Survey on Policy Search for Robotics. *Foundations and Trends R in Robotics*, 2:1–2, 2011.

[17] Thomas G Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value FUnction Decomposition. *arXiv preprint arXiv:cs/9905014*, cs.LG/9905, 1999.

[18] George Dileep. How brain might work. *PhD Thesis*, 30(6):541–550, 1987.

[19] Benjamin Eisenreich, Rei Akaishi, and Benjamin Hayden. Control without controllers: Towards a distributed neuroscience of executive control. *doi.org*, page 077685, sep 2016.

[20] Jerome Feldman. The neural binding problem(s). *Cognitive neurodynamics*, 7(1):1–11, feb 2013.

[21] Richard E Fikes and Nils J Nhsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189—-208, 1971.

[22] Shai Fine. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1):41–62, 1998.

[23] Stan Franklin and F.G. Patterson. The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent. *Integrated Design and Process Technology*, pages 1–8, 2006.

[24] Arthur Franz. Artificial general intelligence through recursive data compression and grounded reasoning: a position paper. Technical report, Goethe University Frankfurt, 2015.

[25] Arthur Franz. On Hierarchical Compression and Power Laws in Nature. In *International Conference on Artificial General Intelligence*, pages 77—-86, 2017.

[26] Karl Friston. Hierarchical Models in the Brain. *Citation: Friston K PLoS Comput Biol*, 4(1110), 2008.

[27] Karl J Friston, Jean Daunizeau, and Stefan J Kiebel. Reinforcement Learning or Active Inference? *PLoS ONE*, 4(7), 2009.

[28] Ilche Georgievski and Marco Aiello. An Overview of Hierarchical Task Network Planning. *arXiv preprint arXiv:1403.7426*, 2014.

[29] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[30] David Ha and Jürgen Schmidhuber. World Models. *CoRR*, abs/1803.1, mar 2018.

[31] George F Harpur and Richard W Prager. Development of low entropy coding in a recurrent network. *Network: Computation in Neural Systems*, 7:277–284, 1996.

[32] David A Hart and Ben Goertzel. OpenCog: A Software Framework for Integrative Artificial General Intelligence. In *AGI*, 2008.

[33] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-Inspired Artificial Intelligence. *Neuron*, 95:245–258, 2017.

[34] Jeff Hawkins and Subutai Ahmad. Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex. *Frontiers in Neural Circuits*, 10, oct 2016.

[35] Jeff Hawkins, Subutai Ahmad, and Yuwei Cui. Why Does the Neocortex Have Layers and Columns , A Theory of Learning the 3D Structure of the World. *bioRxiv*, pages 0–15, 2017.

[36] Jeff Hawkins and Dileep George. Hierarchical Temporal Memory Concepts, Theory, and Terminology. Technical report, Numenta, 2006.

[37] Nicholas Hay, Michael Stark, Alexander Schlegel, Carter Wendelken, Dennis Park, Eric Purdy, Tom Silver, D Scott Phoenix, and Dileep George. Behavior is Everything-Towards Representing Concepts with Sensorimotor Contingencies. Technical report, Vicarious, 2018.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, abs/1512.0, dec 2015.

[39] Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, Alexander Lerchner, and Google Deepmind. Early Visual Concept Learning with Unsupervised Deep Learning. *arXiv preprint arXiv:1606.05579*, 2016.

[40] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. *arXiv preprint arXiv:1707.08475*, 2018.

[41] G E Hinton, J L McClelland, and D E Rumelhart. Chapter 3-Distributed representations. In David E Rumelhart, James L McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed Processing*, chapter Distribute, pages 77–109. MIT Press, Cambridge, MA, USA, 1986.

[42] Sepp Hochreiter and Jj Urgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[43] Marcus. Hutter. *Universal Artificial Intelligence Sequential Decisions Based on Algorithmic Probability*. Springer, 2010.

[44] Jungsik Hwang, Jinhyung Kim, Ahmadreza Ahmadi, Minkyu Choi, and Jun Tani. Predictive coding-based deep dynamic neural network for visuomotor learning. *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, sep 2017.

[45] Piotr Indyk and Rajeev Motwd. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604—-613, 1998.

[46] Jason Jo and Yoshua Bengio. Measuring the tendency of CNNs to Learn Surface Statistical Regularities. *arXiv preprint arXiv:1711.11561*, nov 2017.

[47] Henry Kautz, David Mcallester, and Bart Selman. Encoding Plans in Propositional Logic. In *Proceedings ofthe Fifth International Conference on Principles ofKnowledge Representation and Reasoning*, pages 374—-384, 1996.

[48] Iuliia Kotseruba and John K Tsotsos. 40 Years of Cognitive Architectures Core Cognitive Abilities and Practical Applications. *arXiv preprint arXiv:1610.08602*, 2017.

[49] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *arXiv preprint arXiv:1604.06057*, apr 2016.

[50] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 2007.

[51] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building Machines That Learn and Think Like People. *arXiv preprint arXiv:1604.00289*, 2016.

[52] Eric Laukien, Richard Crowder, and Fergal Byrne. Feynman Machine: The Universal Dynamical Systems Computer. *arXiv preprint arXiv:1609.03971*, 2016.

[53] Eric Laukien, Richard Crowder, and Fergal Byrne. Feynman Machine: A Novel Neural Architecture for Cortical And Machine Intelligence. In *The AAAI 2017 Spring Symposium on Science of Intelligence: Computational Principles of Natural and Artificial Intelligence*, 2017.

[54] Miguel Lázaro-Gredilla, Yi Liu, D. Scott Phoenix, and Dileep George. Hierarchical compositional feature learning. *arXiv preprint arXiv:1611.02252*, pages 1–18, nov 2016.

[55] Tai Sing Lee and David Mumford. Hierarchical Bayesian inference in the visual cortex. *J. Opt. Soc. Am. A*, 20:1434–1448, 2003.

[56] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, nov 2014.

[57] Henry W Lin and Max Tegmark. Criticality in Formal Languages and Statistical Physics. *arXiv preprint arXiv:1606.06737*, 2016.

[58] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *arXiv preprint arXiv:1608.08225*, 2017.

[59] Yi Liu, Xinghua Lou, Christopher Laan, Dileep George, Wolfgang Lehrach, Ken Kansky, D. Scott Phoenix, Bhaskara Marthi, Huayan Wang, Miguel Lázaro-Gredilla, Zhaoshi Meng, and Alex Lavin. A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. *Science*, 358(6368):eaag2612, oct 2017.

[60] E Machery, M Werning, Terrence Stewart, and Chris Eliasmith. Compositionality and Biologically Plausible Models. In W. Hinzen and E. Machery and M. Werning, editor, *Oxford Handbook of Compositionality*. Oxford University Press, 2009.

[61] Adam Marblestone, Greg Wayne, and Konrad Kording. Towards an integration of deep learning and neuroscience. *arXiv preprint arXiv:1606.03813*, 2016.

[62] Tomas Mikolov, Armand Joulin, and Marco Baroni. A Roadmap towards Machine Intelligence. *arXiv preprint arXiv:1511.08130*, pages 1–36, 2015.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015.

[64] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[65] Kumpati S. Narendra, Osvaldo A. Driollet, Matthias Feiler, and Koshy George. Adaptive control using multiple models, switching and tuning. *International Journal of Adaptive Control and Signal Processing*, 17(2):87–102, mar 2003.

[66] E. Nivel, K. R. Thórisson, B. R. Steunebrink, H. Dindo, G. Pezzulo, M. Rodriguez, C. Hernandez, D. Ognibene, J. Schmidhuber, R. Sanz, H. P. Helgason, A. Chella, and G. K. Jonsson. Bounded Recursive Self-Improvement. *arXiv preprint arXiv:1312.6764*, (December 2013), 2013.

[67] Eric Nivel. Ikon Flux 2.0. *Technical Report*, 2007.

[68] Randall C O 'reilly, Dean R Wyatte, and John Rohrlich. Deep Predictive Learning: A Comprehensive Model of Three Visual Streams. *arXiv preprint arXiv:1709.04654*, 2017.

[69] Nuria Oliver, Ashutosh Garg, and Eric Horvitz. Layered representations for learning and inferring office activity from multiple sensory channels. *Computer Vision and Image Understanding*, 96:163–180, 2004.

[70] Giovanni Pezzulo, Francesco Rigoli, and Karl J. Friston. Hierarchical Active Inference: A Theory of Motivated Control. *Trends in Cognitive Sciences*, feb 2018.

[71] Filip Piekniewski, Patryk Laurent, Csaba Petre, Micah Richert, Dimitry Fisher, and Todd L Hylton. Unsupervised Learning from Continuous Video in a Scalable Predictive Recurrent Network. *arXiv preprint arXiv:1607.06854*, jul 2016.

[72] Rui Ponte Costa, Yannis M Assael, Brendan Shillingford, and Tim P Vogels. Cortical microcircuits as gated-recurrent neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 271–282. Curran Associates Inc., 2017.

[73] Qiang Yang and Sinno Jialin Pan. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345—-1359, 2010.

[74] Jielin Qiu, Ge Huang, and Tai Sing Lee. A Neurally-Inspired Hierarchical Prediction Network for Spatiotemporal Sequence Learning and Prediction. *arXiv preprint arXiv:1901.09002*, 2019.

[75] Antti Rasmus, Harri Valpola, Mikko Honkala, Mathias Berglund, and Tapani Raiko. Semi-Supervised Learning with Ladder Networks. *arXiv preprint arXiv:1507.02672*, 2015.

[76] Micah Richert, Dimitry Fisher, Filip Piekniewski, Eugene M. Izhikevich, and Todd L. Hylton. Fundamental principles of cortical computation: unsupervised learning with prediction, compression and feedback. *arXiv preprint arXiv:1608.06277*, aug 2016.

[77] Micah Richert, Dimitry Fisher, Filip Piekniewski, Eugene M. Izhikevich, and Todd L. Hylton. Fundamental principles of cortical computation: unsupervised learning with prediction, compression and feedback. aug 2016.

[78] Gerard J. Rinkus. Sparsey: event recognition via deep hierarchical sparse distributed codes. *Frontiers in Computational Neuroscience*, 8:160, dec 2014.

[79] Gerard J Rinkus. Sparsey™: event recognition via deep hierarchical sparse distributed codes. *Frontiers in computational neuroscience*, 8:160, 2014.

[80] Marek Rosa, Jan Feyereisl, and The GoodAI Collective. A Framework for Searching for General Artificial Intelligence. Technical report, GoodAI, nov 2016.

[81] Sam Roweis and Zoubin Ghahramani. A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2), 1999.

[82] Asim Roy. A theory of the brain - the brain uses both distributed and localist (symbolic) representation. In *The 2011 International Joint Conference on Neural Networks*, pages 215–221. IEEE, jul 2011.

[83] Sara Sabour, Nicholas Frosst, Geoffrey E Hinton, and Google Brain Toronto. Dynamic Routing Between Capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[84] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot Learning with Memory-Augmented Neural Networks. *arXiv preprint arXiv:1605.06065*, 2016.

[85] Adam Santoro, David Raposo, David G T Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, Timothy Lillicrap, and Deepmind London. A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*, 2017.

[86] Jürgen Schmidhuber. PowerPlay: Training an Increasingly General Problem Solver by Continually Searching for the Simplest Still Unsolvable Problem. *Frontiers in Psychology*, 4:313, jun 2013.

[87] Ravid Schwartz-Ziv and Naftali Tishby. Opening the Black Box of Deep Neural Networks via Information. *arXiv preprint arXiv:1703.00810*, 2017.

[88] Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. Technical report, University of Pennsylvania, 1990.

[89] Joshua E. S. Socolar. Nonlinear Dynamical Systems. In *Complex Systems Science in Biomedicine*, pages 115–140. Springer US, Boston, MA, 2006.

[90] M. W. Spratling. A review of predictive coding algorithms. *Brain and Cognition*, 112:92–97, 2017.

[91] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*, 2017.

[92] Richard S Sutton and Andrew G Barto. Sutton and Barto Book: Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 1988.

[93] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014.

[94] Ryan Tan, Daniel R Terno, Jayne Thompson, Vlatko Vedral, and Mile Gu. Towards Quantifying Complexity with Quantum Mechanics. *arXiv preprint arXiv:1404.6255*, 2014.

[95] Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio. Disentangling the independently controllable factors of variation by interacting with the world. In *NIPS 2017 Workshop*, 2017.

[96] Pei Wang. From NARS to a Thinking Machine. In *Proceedings of the 2007 Conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, pages 75—-93, 2007.

[97] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised Predictive Memory in a Goal-Directed Agent. mar 2018.

[98] Steven D. Whitehead and Long-Ji Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, feb 1995.

[99] Laurenz Wiskott and Terrence J Sejnowski. Slow Feature Analysis : Unsupervised Learning of Invariances. *Neural Computation*, 770(4):715–770, 2002.

[100] A D Wissner-Gross and C E Freer. Causal Entropic Forces. *Physical Review Letters*, 110(16), 2013.

[101] D H Wolpert and W G Macready. No Free Lunch Theorems for Optimization. *Trans. Evol. Comp*, 1(1):67–82, apr 1997.

[102] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-Gated LSTM. *CoRR*, pages 1–5, 2015.

APPENDIX A
DETAILED DESCRIPTION OF THE ARCHITECTURE

This appendix describes the various mechanisms of the Toy-Architecture Experts and how hierarchies of them interact. We will first focus first on the description of a basic version of a passive Expert which does not actively influence its environment and is without context. Then, we will show how it can be extended with context (A-B) and actions (A-C). Afterwards, we will extend the definition of the context to allow experts in higher levels to send goals to the experts in lower levels (A-D). In order to enable the architecture to learn how to interact with the environment, we will define the exploration mechanisms (A-F) and describe how a Reinforcement Learning (RL) signal can interface with the architecture. Together, these mechanisms implement distributed hierarchical goal-directed behavior.

Variable	Description
T_h	Length of the past
$T_b = T_h + 1$	Length of the lookbehind part (past + current step)
T_f	Length of the future
$m = T_b + T_f = T_h + 1 + T_f$	Length of the whole sequence
$l \in 1, \ldots, L$	Index of a layer
$j \in J_l$	Index of an Expert in the layer l
$V = \{v_1, v_2, \ldots, v_K\}$	Set of cluster centers (states) of an Expert
$D(\mathbf{x}) = \lvert V \rvert = K$	Dimension of $\mathbf{x}$, number of cluster centers
$\mathbf{O} = \mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(T)$	Sequence of complete observations
$\mathbf{O}_j^l = \mathbf{o}_j^l(1), \ldots, \mathbf{o}_j^l(T)$	Sequence of observations of Expert j in layer l
$\mathbf{x}_j^l(t)$	Hidden state of the Expert j in layer l
$\mathbf{y}_j^l(t)$	Output vector of the Expert j in layer l
$M = \lvert S \rvert$	Number of sequences considered in a TP
$\mathcal{P}$	Set of all providers of context to an Expert.
S_c	Likelihoods of seeing each context element from provider in each position of each sequence

TABLE I
SELECTED NOTATION.

During inference, the task of an Expert j in layer l (E_j^l) is to convert a sequence of observations perceived in its own receptive field $\mathbf{o}_j^l(t)$ into a sequence of output values $\mathbf{y}_j^l(t)$. In the following, when talking about one Expert only, we will omit the subscripts j and superscripts l in the notations of Expert hidden states $\mathbf{x}_j^l$, observations $\mathbf{o}_j^l$, outputs $\mathbf{y}_j^l$, etc. for simplicity.

A. Passive Model without Context

As discussed in Section IV-B, the process is split into the **Spatial Pooler**, the **Temporal Pooler**[24], and **Output Projection**, which can be expressed by the following three equations:

$$\mathbf{x}(t) = f_1(\mathbf{o}(t)|\theta_{\text{SP}}), \tag{5}$$

$$P(S)(t) = f_2(\mathbf{x}(t), \mathbf{x}(t-1), \ldots, \mathbf{x}(t-T_h)|\theta_{\text{TP}}), \tag{6}$$

$$\mathbf{y}(t) = f_3(P(S)(t)), \tag{7}$$

where the θ_{SP} and θ_{TP} are learned parameters of the model.

1) Spatial Pooler: The non-linear observation function from Eq. (5) is implemented by k-means[25] clustering and produces one-hot vector over the hidden states:

$$\mathbf{x}(t) = f_1(\mathbf{o}(t)|\theta_{\text{SP}}) = \delta\left(\underset{v_i \in V}{\operatorname{argmin}}\left(\operatorname{dist}(v_i, \mathbf{o}(t))\right)\right), \tag{8}$$

where $\operatorname{dist}(\mathbf{v}_1, \mathbf{v}_2)$ is the standard L^2 Euclidean distance between two vectors $\mathbf{v}_1, \mathbf{v}_2$, and V is a set of learned cluster centers of the Expert (corresponds to the parameter θ_{SP}), and $\delta(v_i \in V)$ is a winner-takes-all (WTA) function which returns a one-hot representation of v_i. The observation function f_1 considers only the current observation and covers step number two (compression) as described in Appendix IV-B (separation is performed on a level of multiple Experts and is described in Appendix A-G).

Because we are learning from a stream of data, it might happen that some cluster centers in the Saptial Pooler do not have any data points and thus would be never adapted. There can be two underlying reasons: 1) the cluster centers were initialized far from any meaningful input, or 2) the agent has not seen some types of inputs for a longer period (e.g. stays inside a building for some time and does not see any trees). In the situation 1, we would like to move the cluster to an area where it would be more useful, but in situation 2, we would typically want to keep the cluster on its current position in order not to forget what was learned and can be useful again in the future. We solve this dilemma by implementing a *boosting algorithm* (similar to [36]). We define a hyper-parameter b (*boosting threshold*) and every cluster center, which did not have any data for the last b steps, starts to be boosted - it is moved towards the cluster center with highest variance among its data points. By this parameter we can modify the trade-off between adaptation to new knowledge and not-forgettig the old one.

2) Temporal Pooler: The goal of the Temporal Pooler is to take into account a past sequence of hidden states $\mathbf{x}(0), \ldots, \mathbf{x}(t)$ and predict a sequence of future states $\mathbf{x}(t+1), \ldots, \mathbf{x}(t+T_f)$. Since the sequence of observations $\mathbf{O}_j^l$ might not have the Markovian property, and it might have been further compromised by the Spatial Pooler, the problem is not

[24]The terminology adopted from [34]

[25]Due to simplicity and interpretability reasons, as described in Section I.

solvable in general. So we limit the learning and inference in one Expert to Markov chains of small order and learn the probabilities:

$$P(\mathbf{x}(t+T_f),\ldots,\mathbf{x}(t+1)|\mathbf{x}(t),\mathbf{x}(t-1),\ldots,\mathbf{x}(t-T_h)), \quad (9)$$

which we express in a form of sequences $s_i = v_{i_1},\ldots,v_{i_{T_h}},v_{i_{T_h+1}},v_{i_{T_h+2}}\ldots,v_{i_{m=T_h+1+T_f}}$. Each sequence can thus be divided into three parts of fixed size: a history part $v_{i_1},\ldots,v_{i_{T_h}},v_{i_{T_h}}$, the current state $v_{i_{T_h}+1}$ and a future (*lookahead*) part $v_{i_{T_h+2}},\ldots,v_{i_m}$, the entire sequence having the length of $m = T_h+1+T_f$. We call *lookbehind* the history together with the current step (sequence of length $T_b = T_h+1$). See the bottom of Fig. 21 for an illustration.

The theoretical number of possible sequences of hidden states grows very quickly with the number of states and the required order of Markov chains. But the observed sub-generator usually generates only a very small subset of these sequences in practice. Using a reasonable number of states and length of sequences (e.g. $N = 30$ and $m = 4$), it is possible to learn the transition model simply by storing all encountered sequences in each Expert $S_j^l = \{s_1, s_2, \ldots, s_M\}$ and computing their prior probabilities based on how often they were encountered. Then, the probability of the i-th sequence $P(s_i)$ is computed as:

$$P(s_i) = \left\langle \bar{P}(s_i) \right\rangle_1 = \frac{\bar{P}(s_i)}{\sum_{s_j \in S} \bar{P}(s_j)}, \quad (10)$$

where $\langle D \rangle_1$ denotes normalization of values D to be probabilities, and:

$$\bar{P}(s_j) = P_{pr}(s_j) \cdot P(s_j|\mathbf{x}(t-T_h:t)), \quad (11)$$

where $P_{\text{pr}}(s_j)$ is the prior probability of the sequence $s_j \in S$ (how often it was observed relative to other sequences), $P(s_j|\mathbf{x}(t-T_h:t) = \prod_{d=0}^{T_h} I(s_j, d, \mathbf{x}(t-T_h+d))$ is the match of the beginning of the sequence s_j with the recent history of states $\mathbf{x}(t-T_h),\ldots,\mathbf{x}(t)$, and $I(s_j, d, \mathbf{x}) \mapsto \{1-\epsilon, \epsilon\}$ is an indicator function producing value close to 1 if the hidden state $\mathbf{x}$ corresponds to the cluster v_{j_d} at the d-th position in the sequence s_j, ϵ otherwise[26]. The parameter $T_h < m$ defines the fixed length of the required match of the sequence, so given $T_h = 2$ and $m = 5$, the sequence probabilities will be computed based on first $T_h + 1 = 3$ clusters in the sequence, therefore these sequences can be used for predicting 2 steps into the future. The value $P(S)(t)$ from Eq. (6) is then a probability distribution over all sequences in time step t:

$$P(S)(t) = \{P(s_i)(t) : s_i \in S\}. \quad (12)$$

These are the main principles behind learning and inference of the Temporal Pooler.

[26] $\epsilon > 0$ is a small constant ensuring each sequence has a nonzero probability and that sequences corresponding at least partially to the data have higher probabilities than those which do not correspond at all.

3) Output Projection: Finally, the Expert has to apply the output function described in Eq. (7). In each time step, the output function takes the the current sequence probabilities and produces output of the Expert $y(t)$:

$$\mathbf{y}(t) = f_3(\{P(s_i)(t) : s_i \in S\}). \quad (13)$$

When defining the output function, the following facts need to be taken into account: the outputs $\mathbf{y}_j^l$ of Experts in layer l are processed by the Spatial Poolers of Experts in layer $l+1$ where the clusters of the observations are made based on some distance metric (k-means clustering with Euclidean distance in the current version). There are two extreme situations:

- In the case that the sequence of states $\mathbf{x}_j^l(t)$ for the child Experts $j \in J_l$ on layer l are not predictable, the parent Experts in layer $l+1$ should form their clusters mostly based on the spatial similarities of the hidden states of the Experts[27] in layer l. This way, the level of detail of unpredictable processes is preserved as much as possible and passed into higher layers of abstraction where these uncertainties can be resolved.
- On the other hand, in the case that the state sequences $\mathbf{x}_j^l(t)_t$ are perfectly predictable, the spatial properties of the observations are relatively less important than their behavior in time, and the clustering in the layer $l+1$ should be done mostly based on similarity of sequences (temporal similarity).

Based on these properties, the output function should be defined so that the resulting hierarchy implements **implicit efficient data-driven allocation of resources**. The parts of the process that are easily predictable by separate Experts low in the hierarchy will be compressed early. The unpredictable parts of the process will propagate higher into the hierarchy where the following Experts try to learn/resolve them with use of wider (spatial and temporal) context. This is a compromise between sending what the architecture knows well vs sending just residual errors [90].

In the current version, we use the following output projection function: The output dimension $D(\mathbf{y})$ is fixed to be the same as the number of hidden states in the Expert $D(\mathbf{x}) = |V| = K = D(\mathbf{y})$. The output function is defined as follows:

$$\mathbf{y}_j^l(t) = f_3(P(s_i \in S_j^l)(t)) = \left\langle \mathbf{x}(t) + \sum_{k \in 1..K} \delta(x_k) \sum_{i \in 1..M} \bar{P}(s_i) \sum_{d \in 1..m} I(s_i, d, \delta(x_k)) \right\rangle_1, \quad (14)$$

[27] The one-hot output of the Spatial Pooler does not fulfill this requirement. But the spatial similarity is preserved over the outputs of multiple Experts which receive similar inputs (distributed representation). The parent Experts E_i^{l+1} then receive outputs of multiple experts from layer l, therefore they perceive the code which preserves the spatial similarity.

where I is the indicator function from Eq. (11), δ is the WTA function from Eq. (8), $\langle . \rangle_1$ is the normalization function from Eq. (10), and $\bar{P}(s_i)$ is the probability that we are currently in the sequence s_i from Eq. (11). This definition of the output function has the following properties in the two extremes:

- In the that case the observation sequence is not predictable, the predictions from the sequences with high probability will have high dispersion over the future clusters. Therefore the position corresponding to the current hidden state and recent history of length $T_h + 1$ will be dominant in the output vector $y(t)$. So the parent Expert(s) in layer $l + 1$ will tend to cluster these outputs mostly based on the recent history of length $T_h + 1$ (as opposed to the predictions).
- In the case that the observation sequence is perfectly predictable, only one sequence will have high probability each time step, so both the past and predicted states will have high probability. Therefore the parent Expert(s) in $l + 1$ will tend to cluster based on the predicted future more than in the previous case. The sequence of observations for E_i^{l+1} will therefore be more linear (similar to sliding window over the recent history and future), therefore it will be possible to chunk the observations more efficiently. More importantly: the output of these parent Experts $y_i^{l+1}(t)$ will correspond more to the future (since the lower-level Experts are predicting better). As a result, the higher levels in the hierarchy should compute with data which correspond to the increasingly more distant future. This way the hierarchy does not think about what happened but rather what is going to happen.

This means that the temporal resolution in higher layers is determined automatically based on the predictability of the observations in the current layer, and this resolution can dynamically change in time. Since the clustering applies strict winner-takes-all (WTA) functionality and the Temporal Pooler does not accept repeating inputs, the entire mechanism naturally results in a **completely event-driven architecture**.

B. Passive Model with External Context

Until now, the goal of each Expert was to learn a model of the part of the environment solely based on its own observations $\mathbf{o}_j^l$. This can lead to highly suboptimal results in practice. Often it is necessary to use some longer-temporal and/or longer-spatial dependencies (as described in Section IV-B). A context input (see Fig. 4) vector is used for providing these information if necessary.

The meaning and use of the bottom-up and context (top-down and lateral) connections should be asymmetrical: the bottom-up (excitatory) connections decode "visual appearance" of the concept, while top-down (modulatory) connections [1] just help to resolve the interpretation of the perceived input using the context. This asymmetry should prevent encountering positive feedback loops (where the bottom-up input might be completely ignored) both during learning and inference. As a result, the hidden state of the architecture should still track actual sensory inputs.

The context input can be then seen as a **high-level description of the current situation** from the Expert's surroundings (both from the higher level and possibly from neighboring experts in the same layer).

It is possible to use various sources of information as a context vector, such as:

- **Past activity** of other Experts: this extends the ability of E_j^l to take into account dependencies further in the past.
- **Recent activity** of other Experts: this increases spatial and temporal range.
- **Predicted activity** of other Experts: this extends the ability of E_j^l to distinguish the recent observation history more according to the future. This process could be likened to Epsilon Machines, where the idea is to differentiate histories according to their future impact [12], [94].

The context output of an Expert: $Co_i^l(t) = \langle \mathbf{x}_j^l(t), \mathbf{x}_j^l(t+1) \rangle$ is a concatenation of the Spatial Pooler output (i.e. the winning cluster for this input) and the Temporal Pooler prediction of the next cluster. The goal $Go_i^l(t)$ is also attached to the context (see Fig. 20), but we will talk about them separately for clarity. The ensemble can be thought of colloquially as communicating: "Where, I am", "Where I expect to be in the future", and "What reward I expect for each possible future (clusters)".

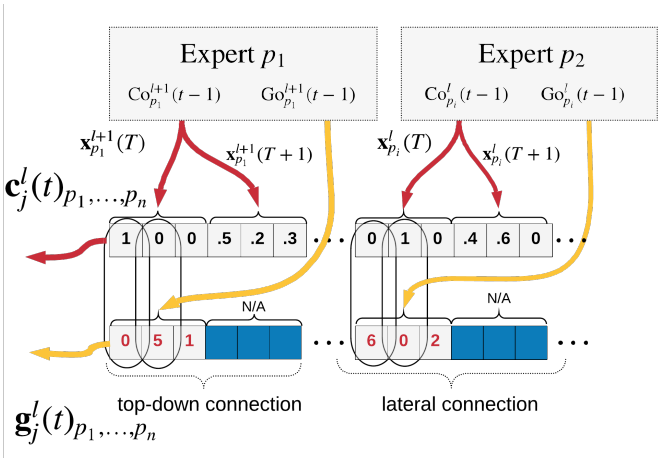


Fig. 20. Context and Goal input vectors to Expert E_j^l. Both are collections of top-down and lateral inputs from other Experts from the previous time step. The Goal input has some parts masked-out (blue parts). The resulting two input vectors can be interpreted as a high-level description of the current state, and a passive prediction of what will happen next ($\mathbf{x}_j^l(T)$, $\mathbf{x}_j^l(T+1)$), and the goal as a preference (measured in the expected value of reward) for the next state. Note that in this figure, the variable t denotes the time for the Expert receiving the context (E_j^l), while T denotes the time for an Expert sending the context (because all Experts are event driven, the time between two changes in an Expert states is different for different Experts).

The context input is a collection of context outputs (refer to red lines in the Fig. 4) from multiple other Experts. Each Expert supplying context is known as a provider defined as, and there is no distinction between parent providers and lateral providers[28]. The context input to Expert E_j^l is therefore defined as:

$$\mathbf{c}_j^l(t) = (\langle \mathbf{x}_{i(T)}(t-1), \mathbf{x}_{i(T+1)}(t-1) \rangle : i \in \mathcal{P}_j^l) \quad (15)$$

where $\langle . \rangle$ denotes concatenation, $\mathcal{P}_j^l$ is set of providers for E_j^l, and $\mathbf{x}_{i(T)}(t-1)$ and $\mathbf{x}_{i(T+1)}(t-1)$ are the current (T) and predicted ($T+1$) clusters of the provider i from the previous step[29] respectively.

Context is incorporated in the Temporal Pooler prediction process by having the TP learn what the likelihood of each context element from each provider being 1 is for each position in each sequence to produce ${S_C}_j^l$.

In using the context during inference, we augment the calculation of the unnormalised sequence probabilities (Eq. (11)) by also matching the current history of contexts $\mathbf{c}_j^l(t - T_h), \ldots, \mathbf{c}_j^l(t)$ with the remembered sequence contexts ${S_C}_j^l \in \theta_{TP}$.

We start by extending the definition of $P(S)(t)$ in Eq. (6):

$$P(S)(t) = f_2\big(\mathbf{x}(t), \ldots, \mathbf{x}(t-T_h), \mathbf{c}(t), \ldots, \mathbf{c}(t-T_h) | \theta_{TP}\big) \quad (16)$$

We consider each context provider separately. For each sequence, we calculate the likelihood of that sequence based on the history from each individual provider $p \in \mathcal{P}$:

$$P_c(S_j^l)(p) = \mathbf{c_j^l}(t - T_h : t)(p) \cdot {S_C}_j^l(p) \quad (17)$$

Considering the role of the context, we wish that in a world where multiple sequences are equally probable, the context will disambiguate the situation. Given that ${S_C}_j^l$ is learned alongside S_j^l, in a situation where each Expert has the same data, the contexts sould correlate highly with $\bar{P}(S)$ and the predictions based solely on the context history would approximate to the predictions using the cluster history and priors:

$$\bar{P}(S) \approx \mathbb{E}_{p_i}(P_c(S)(p_i)) \quad (18)$$

But in reality, each Expert might be looking at a different receptive field and have generally different information. On the other hand, context from most of the Experts can be of no use for the recipient and it is probable that it will be highly correlated among the providers. Thus averaging the predictions based on the individual contexts might obscure the valuable information. So rather than using every context equally for disambiguation, we would like to use only the most informative one. We choose the most unexpected context to use, as the context which is the most disruptive to the otherwise anticipated predictions is likely to contain the most information about the current state of the agent and environment. As a metric of unexpectedness, we are using Kullback-Leibler [50] divergence between the predictions based on the history of cluster centers and the one individual context vs predictions based just on the history of cluster centers.

We therefore update Eq. (10) to include this selection and use of the most informative context:

$$P(s_i) = \langle \bar{P}(s_i) \cdot \mathcal{F}(\bar{P}(s_i), P_c(s_i)) \rangle_1, \text{ where} \quad (19)$$

$$\mathcal{F}(\bar{P}(s_i), P_c(s_i)) = S_C(\operatorname*{argmax}_{p \in \mathcal{P}} D_{KL}(\langle \bar{P}(s_i) \rangle_1 || \langle \bar{P}(s_i) \cdot P_c(s_i) \rangle_1)) \quad (20)$$

As a result, by taking into account the context (a high level description of the current situation), each Expert can consider also longer spatial and temporal dependencies which defy the strict hierarchical structure (see Section III-D) in order to learn the model of its own observations more accurately.

C. Actions as Predictions

Until now, the architecture was able only to passively observe the environment and learn its model. Now, the mechanisms necessary to actively interact with the environment (i.e. to produce actions) will be introduced with as small a change to the architecture as possible. First, we define how the actions are represented. In the following sections we will focus on mechanisms which determine which action to take.

From the theoretical perspective, the HMM can be extended to a Partially Observable Markov Decision Process (POMPD) [92]. While taking into account that each Expert processes Markov chains of order $m-1$, the decision process corresponds to the setting:

$$P\big(\mathbf{x}(t) | \mathbf{x}(t-1), a(t-1), \ldots, \mathbf{x}(0), a(0)\big) = P\big(\mathbf{x}(t) | \mathbf{x}(t-1), a(t-1), \ldots, \mathbf{x}(t-m), s(t-m)\big), \quad (21)$$

where $a(t)$ denotes an action taken by the Expert (at a particular layer) at time t. Note that this setting could be treated as a task for active inference, where the agent proactively tries to discover the true state of the environment if necessary [27], [98]. But for now, we will consider a similar approximation

[28]Note that in general context connections that skip multiple layers are allowed as well.

[29]The variable t denotes the time for the Expert receiving the context (E_j^l), while T denotes the time for an Expert sending the context (because all Experts are event driven, the time between two changes in an Expert states is different for different Experts).

of the problem as in the previous sections and leave explicit active inference to future work.

Since we want the hierarchy to be as general as possible, it is desirable to define the actions in such a way that they can be used in case the Expert has the actual ability to control the actuators (either directly or indirectly through other Experts), but they will not harm the performance much in case that the Expert is not able to perform actions, and can only passively observe.

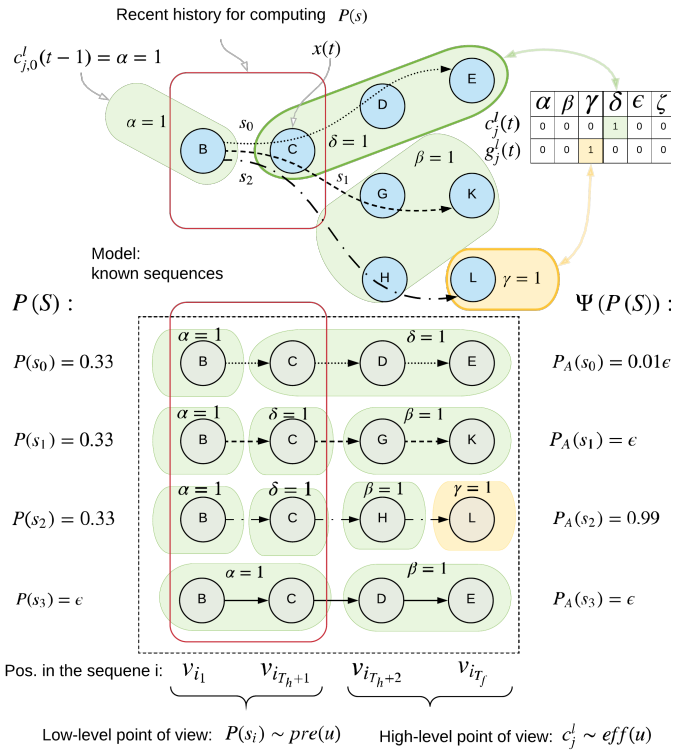


Fig. 21. An example of recent sequence of states (where $T_h=1$, $T_f=2$) $\mathbf{x}(t-T_h:t)$ in E^l_j which shows: 1) The sequence of context inputs $\mathbf{c}^l_j(t-T_h:t)$ helping to resolve uncertainty during computation of $P(s)$ of known sequences; 2) A goal vector $\mathbf{g}^l_j(t)$ defining the required target state. The $P(s)$ is modified to $P_A(s)$ so that to minimize the distance between the $\mathbf{c}^l_j(t)$ and $\mathbf{g}^l_j(t)$ in the given time horizon T_f. **Bottom**: library of learned sequences S^l_j, each sequence is defined by an ordered list of states x, each potentially in different context c^l_j. **Top**: visualization of the current state $\mathbf{x}(t)$ and several possible futures. These futures are estimated based on the content of the Model: (which stores known sequences). First, the probability distribution $P(s_i)$ is computed based on sequence of recent states and contexts. Then, the probabilities of sequences satisfying the goal description in the horizon T_f are increased proportionally to the probability of satisfying this goal (updated sequence probabilities $P_A(s_i)=\Psi(P(S))$ based on the goal are depicted on the right). As a result, this increases the probability of choosing the state H as an action—setting it as a goal output $\mathrm{Go}^l_j(t)=H$. In this example, the first 3 sequences are equally matched by the $\mathbf{x}(t-T_h:t)$ and $\mathbf{c}(t-T_h:t)$ therefore they have equal probabilities. But after applying the Ψ, the s_0 has the smallest probability, the s_1 has higher probability since it sets the δ to zero in the future, but the s_2 has the highest probability, because it both: sets the δ to zero and γ to one, as required by the goal input $\mathbf{g}^l_j(t)$.

For this reason, actions are not explicitly modeled in this architecture. Instead, **an action is defined as an actively selected prediction of the next desired state**[30] $\mathbf{x}(t+1)$. The selected action (desired state in the next step) is indicated on the Goal output of the Expert (see Fig. 4): $Go^l_j(t)$.

Given the library of sequences S and recent history of hidden states $\mathbf{x}(t-T_h:t)$ and context inputs $\mathbf{c}(t-T_h:t)$, the Expert computes the sequence probabilities $P(S)$ using Eq. (19). Then, those sequence probabilities are altered (based on preferences over the states to which they lead, see Appendix A-D and A-E). This results in a new probability distribution

$$P_A(S)=\Psi\big(P(S)\big), \tag{22}$$

where Ψ can be seen as a sequence selection function, see Fig. 21 for illustration. Finally, the Goal output of the Expert for the next simulation step $\mathrm{Go}^l_j(t)$ is computed (see Fig. 4). This can be seen as actively predicting the next position in a sequence:

$$\mathrm{Go}^l_j(t)=\Theta\left(\Phi(P_A(S))\right), \tag{23}$$

where the Φ converts the probabilities of sequences $P(S)(t)$ into probability distribution over clusters $P(V)(t+1)$ predicted in the next step:

$$P(v\in V)=\Phi\left(P(s\in S)\right)=\sum_{s\in S}P(s)\cdot I(s,T_h+2,\delta(v)), \tag{24}$$

where I is the indicator function from Eq. (11), position T_h+2 corresponds to the next immediate step and δ is the WTA function from Eq. (8). Θ in Eq. (23) is an action selection function for which it is possible to use multiple functions, namely identity, ϵ-greedy selection, sampling or ϵ-greedy sampling.

In the example in Fig. 21, without considering any preferences over the sequences (the Ψ in Eq. (22) collapses to identity), the probabilities of the first three sequences are equal, therefore the function Θ would choose the states D, G and H with equal probability.

The whole process can be seen as follows: Each Expert throughout the hierarchy, calculates a plan based on a short time horizon T_f, chooses the desired imminent actions (states one step in the future which are desired and probably reachable) and encodes this information as the Goal output $\mathbf{g}^l_j$. This signal is then either received by other Experts and interpreted as the goal they should reach), or used directly by the motor system in the case that the Expert is able to control something.

In the presented prototype implementation, the desirability of the goal states is encoded as a vector of rewards that the parent expects that the architecture will receive if the child can produce a projection $\mathbf{y}(t+1)$ which will cause the parent

[30]This actively chosen predicted state action should be reachable from the current state $\mathbf{x}(t)$ with a high probability, i.e., be in coherence with what is possible.

SP to produce the hidden state $\mathbf{x}(t+1)$ corresponding to the index of the goal value.

An expert receiving a goal context computes the likelihood of the parent getting to each hidden state using its knowledge of where it presently is (Eq. (19)), which sequences will bring about the desired change in the parent (Eq. (30)), and how much it can influence its observation in the next step $\mathbf{o}(t+1)$ by its own actions (see Appendix A-F). It rescales the promised rewards using these factors, combines them with knowledge about its own rewards (see Appendix A-E) and then calculates which hidden states in the next step correspond to sequences leading towards these combined rewards. From here, it either publishes its own goal Go (expected reward for getting into each cluster), or if it interacts directly with the environment picks an action to follow[31]. This mechanism is described in more details in the following section.

D. Goal-directed Inference

This section will describe the mechanisms which enable the Expert:

- To decode the goal state received from an external source (usually other Experts).
- To determine to what extent the goal state can be reached, or at least if the distance between the current state and the goal can be decreased.
- To make a first step ("action") leading towards this goal if it is possible (by setting Go_j^l to the appropriate value).

As a result, these mechanisms should allow the hierarchy of Experts to act deliberately. The architecture will hierarchically decompose a decision—potentially a complex plan, represented as one or several steps on an abstract level, into a hierarchy of short trajectories. This corresponds to the ability to do decentralized goal-directed inference, which is similar to hierarchical planning (e.g. state-based Hierarchical Task Network (HTN) planning [28]). Note that such a hierarchical decomposition of a plan has many benefits, such as the ability to follow a complex abstract plan for longer periods of time, but still be able to reactively adapt to unexpected situations at the lower levels. There are also theories that such a mechanisms are implemented in the cortex [70].

In this section, we will show a simple mechanism which approximates the behavior of a symbolic planner. This demonstrates one important aspect: the hierarchy of Experts converts the input data into more structured (symbolic-like) representations. On each level of the hierarchy the representation can be seen either as sub-symbolic or as a symbolic one. This gives us the ability to define **symbolic inference mechanisms on all levels** of the hierarchy (e.g. planning), which then **use grounded representations**.

[31]The action of bottom level Experts at $t-1$ is provided on $\mathbf{o}(t)$ from the environment, so the picking of an action is equivalent to taking the cluster center of the desired state and sampling the actions from the remembered observation(s).

Furthermore, in Appendix A-E, we will show how a reinforcement signal can be used for setting preferences over the states in each Expert. This will in fact equip the architecture with model-based RL [64]. It also means that **locally reachable goal states can emerge across the entire hierarchy** with them appearing on different time scales and levels of abstraction, which leads to completely decentralized decision making.

The main idea of goal-directed inference is loosely inspired by the principles of predictive coding in the cortex [90], where it is assumed that each region tries to minimize the difference between predicted and actual activity of neurons. In ToyArchitecture, a more explicit approach for determining the desired state is used. The approach can be likened to a simplified, propositional logic-based version [47] of the symbolic planner called Stanford Research Institute Problem Solver (STRIPS) [21]. In this architecture, each Expert will be able to implement forward state-space planning with limited horizon [29].

STRIPS definition: Let $\mathcal{L}$ be a propositional language with finitely many predicate symbols, finitely many constant symbols, and no function symbols. A restricted state-transition system is a triple $\Sigma^p = (S^p, A^p, \gamma^p)$, which is described in the Table II.

variable	meaning
$s_i^p = x_1, x_1, ..$	State—set of ground atoms of L
$S^p = \{s_1^p, s_2^p, ...\}$	Set of states
$U = \{u_1, u_2, ...\}$	Set of operators (actions)
$\gamma : S^p \times U \to S^p$	State transition function
$u = (\mathrm{pre}(u), \mathrm{eff}(u))$	Operator —transforms one state to another, if applicable
$\mathrm{pre}(u) = \{\mathrm{pre}^+(u), \mathrm{pre}^-(u)\}$	Precondition—set of literals which determines if the operator is applicable
$\mathrm{eff}(u) = \{\mathrm{eff}^+(u), \mathrm{eff}^-(u)\}$	Effect—set of literals which determines how the operator changes the state if applied

TABLE II
STRIPS LANGUAGE DEFINITION.

State s^p satisfies a set of ground literals g^p (denoted $s^p \vDash g^p$) iff: every positive literal in g^p is in s^p and every negative literal g^p is not in s^p. It is possible to represent states s^p as binary vectors (where each ground literal corresponds to one position in the vector) and operators/actions u as operations over these vectors (therefore a finite state space is expected).

The operator u is applicable in the state s^p under the following conditions.

$$\mathrm{pre}^+(u) \subseteq s^p, \tag{25}$$

$$\mathrm{pre}^-(u) \cap s^p = \{\}. \tag{26}$$

Then, the state transition function γ for an applicable operator u in state s^p is defined as:

$$\gamma(s^p, u) = (s^p - \mathrm{eff}^-(u^p)) \cup \mathrm{eff}^+(u^p). \tag{27}$$

The STRIPS planning problem instance is a triple $P^p = (\Sigma^p, I^p, G^p)$, where: Σ^p is the restricted state-transition system described above, $I^p \in S^p$ is a current state and G^p is a set of ground literals describing the goal state (which means that the G^p describes only required properties, which are subset of the propositional language $\mathcal{L}$).

Given the planning instance P^p, the task is to find such a sequence of operators (actions), which consecutively transforms the initial state I^p into a form which fulfills the goal state G^p conditions.

One possibility how to find such a sequence is to search through the state-space representation. Since the decision tree has potentially high branching factor, it is useful to apply some heuristic while choosing the operators to be applied. To quote from the original paper [21]: "We have adopted the General Problem Solver strategy of extracting differences between the present world model [state] and the goal and of identifying operators that are relevant to reducing these differences".

Now we will describe how an approximation of this mechanism is implemented in ToyArchitecture.

Similar mechanisms in the ToyArchitecture: The architecture learns sequences s_i of length m and each step in the sequence corresponds to an action. Each sequence is a trajectory in a state-space of Expert E_j^l (see states with big letters and transitions between them in the Fig. 21). But, more crucially, from the point of view of the parent[32] Expert E_j^{l+1}, **each sequence can be seen as an operator** u.

For the purposes of planning, aside the context vector input $\mathbf{c}_j^l$, the Expert is equipped with a goal vector input $\mathbf{g}_j^l$, which specifies the goal description G.

From the point of view of E_j^l, $\mathbf{c}_j^l(t)$ describes the current state (corresponds to $s_i^p(t)$ in the STRIPS), while $\mathbf{g}_j^l(t)$ describes a superposition of desirable goal states. With each position marked by a real number indicating how preferable the state is for the parent[33].

Note that (as explained in Appendix A-B) each Expert learns the probabilities of sequences dependent on context and position in the sequence (Eq. (19)) and stores them the form of a table of frequencies of observing each combination. This allows us to define the operator $u_i = \{\mathrm{pre}(u_i), \mathrm{eff}(u_i)\}$ (corresponding to the learned sequence s_i) in a stochastic form, where we define the probability of $\mathrm{eff}(u_i)$ being $\hat{\mathbf{c}}$ as the probability that the ending clusters of the sequence s_i (its lookahead part, see Appendix A-A2) will be observed in the context $\hat{\mathbf{c}}$:

$$P\left(\mathrm{eff}(u_i) = \hat{\mathbf{c}}\right) = \max_{f \in 1..T_f} P\left(\mathbf{c}_j^l(t+f) = \hat{\mathbf{c}} | s_i(t)\right). \quad (28)$$

[32]For simplification, we can consider one parent Expert, but the approach generalizes to top-down connections from multiple parents as well as multiple lateral connections from other Experts in the same layer simultaneously.

[33]We can think of this as the expected value of the state for the parent.

The precondition $\mathrm{pre}(u_i)$ determining the applicability of the operator can be also defined in a stochastic manner as the probability that the Expert is currently in the sequence s_i:

$$P\left(\mathrm{pre}(u_i) = \mathbf{c}_j^l\right) = P^G\left(s_i(t)\right), \quad (29)$$

where $P^G\left(s_i(t)\right)$ is a probability of the sequence s_i similar to Eq. (19), but computed for the situation when the Expert actively tries to influence it (see Appendix A-F for more details). Note that Eqs. (28) and (29) imply that the meaning of the operators $u_i = \{\mathrm{pre}(u_i), \mathrm{eff}(u_i)\}$ is different in each Expert and each time step.

Finally, the sequence selection function Ψ from Eq. (22) can be defined as follows:

$$\Psi\left(P(s_i(t))\right) = \left\langle P\left(\mathrm{pre}(u_i) = \mathbf{c}_j^l(t)\right) \cdot P\left(\mathrm{eff}(u_i) = \mathbf{g}_j^l(t)\right)\right\rangle_1 = \left\langle P^G(s_i(t)) \cdot \max_{f \in 1..T_f} P\left(\mathbf{c}_j^l(t+f) = \mathbf{g}_j^l(t) | s_i(t)\right)\right\rangle_1, \quad (30)$$

where $\langle . \rangle_1$ denotes normalization to probabilities described in Eq. (10).

This means that each Expert can implement deliberate decision making looking ahead T_f steps into the future. Each step, it looks for currently probable sequences which maximise the expected value when moving the parent from the current context vector $\mathbf{c}_j^l(t)$ to the state dictated by $\mathbf{g}_j^l(t)$ as much as possible[34].

The entire process is summarized in the Algorithm 1 and illustrated on an example in Fig. 21. Compared to the STRIPS, each Expert can plan with only limited lookahead, but this decision is decomposed into sub-goal of Experts in lower layer. This leads to efficient hierarchical decomposition of tasks. Moreover, compared to classical symbolic planners, **representation in hierarchy is completely learned from data**. Moreover, since the Experts still compute with probabilities, the inference is stochastic and can be interpreted as continuous and sub-symbolic.

E. Reinforcement Learning

In the previous section we described how the Expert can actively follow an externally given goal. The same mechanism can be used for reinforcement learning with a reward $r \in \mathbb{R}$.

When reaching a reward, every Expert in the architecture gets the full reward or punishment value. During learning, each Expert assumes that it was at least partially responsible for gaining the reward and therefore associates the reward gained at t with the state/action pair at $t-1$, so that for all $\{(\mathbf{x}(t), a)|, a \in A_j^l\}$ there is a corresponding $r \in S_{R_j}^l$ which

[34]This allows very high expected value but improbable actions to also be considered.

Algorithm 1: Goal directed inference—approximation of a stochastic version of STRIPS state-space planning with limited horizon. Describes how the Expert decides on which action to apply in order to maximise the expected value of rewards communicated in $\mathbf{g}_j^l(t)$ from the current context $\mathbf{c}_j^l(t)$. If an Expert is directly connected to the actuators, then an action is selected directly, otherwise the expert propagates the expected values of the states to its children.

Data: Observation history $\mathbf{o}_j^l(t-T_h:t)$,
Context history $\mathbf{c}_j^l(t-T_h:t)$,
Goal description $\mathbf{g}_j^l(t)$
Result: Goal output $Go_j^l(t)$

```
Compute applicability of the operators: compute
  sequence probabilities P(s_i)(t) (Eq. (29));
Select operators that are applicable and have high chance
  of achieving one of the the goal states: weight sequence
  probabilities by these Ψ(P(s_i)(t)) (Eq. (30));
Compute the probabilities of preferred states in the next
  step x(t+1) by Φ(Ψ(P(s_i)(t))) (Eq. (24));
if Expert is to produce an action then
    Apply an action selection function Θ (Eq. (23));
    Set the selected action to the Go_j^l(t);
else
    Set Go_j^l(t) to the values of the received expected
      values weighted by the computed next step
      probabilities;
end
```

is an estimate of the reward gained when in state $\mathbf{x}$ and taking action a. Because the Experts are event driven, they sum up all the rewards received during the steps they did not run (their cluster did not change).

The initial expert reward calculation is:

$$\mathbb{E}[R_j^l(t\dots T_f)] = Go_j^l(t)\cdot {S_C}_j^l(t\dots T_f)+{S_R}_j^l(t\dots T_f) \quad (31)$$

This is the expected value of the promised reward from each provider, for each future state in each sequence. Any rewards that the Expert can 'see' from this point are also included as the term ${S_R}_j^l(t\dots T_f)$).

The action which the Expert should perform is related to the sequence that it wants to move to. As it is trying to maximise its rewards, the Expert should pick an action which would position it in a sequence where there are the most rewards. As this is the expected value (and also assume that rewards are sparse, and that an Expert can only expect reward once in the current lookahead (i.e. T_f) of the sequence.), the maximum of rewards from the sequence is used[35]:

[35]Taking the maximum as a lower bound on the expected reward works only in case rewards are all non-negative or all non-positive. In case we want the agent to accept both rewards and punishments at once, they need to be processed separately and combined just in the lower Experts sending the actual actions to the environment.

works only in case rewards are all non-negative or all non-positive. In case we want the agent to accept both rewards and punishments at once, they need to be processed separately and combine just in the lower Experts doing the actual actions.

$$\mathbb{E}[R(t+1)] = \max_{f\in 1\dots T_f} R(t+f)\cdot I(S)(t+f)\cdot P(S)(t)\cdot d^f \quad (32)$$

where $I(S)$ is an influence model, which is a model of how able the Expert is to move from one state to another by taking an action (see Appendix A-F for how this is calculated) d^f is a discount factor through time and $P(S)(t)$ is the probability distribution over sequences that the Expert is in at time t.

From the perspective of the Expert, $r_i = \operatorname{argmax}_i \mathbb{E}[R(t+1)]$ is the best possible reward, $s_i(t+1)$ is the best possible sequence, and $a(t)$ is the best possible action for the Expert to take given the probability of the current state, promised rewards, and the probability of affecting future states. The action $a(t)$ is therefore the one taken by the Expert if it is expected to interact with the outside world. Otherwise the output goal of this layer $Go_j^{l-1}(t+1)$ is set to the values of $\mathbb{E}[R(t+1)]$ thus propagating a promise of the expected reward to to its children.

Because goal-directed inference is hierarchically distributed through the whole architecture, the external reward r has to be provided to each Expert[36]. In this way a top level Expert tries to get to an abstract state where the agent received a reward (for example a quadrant of a map), where the rewarding state is already reachable by some middle layer expert (within its finite horizon T_f) and the agent is driven closer to the reward with higher precision (e.g., to a particular room) until finally a low-layer expert is able to find a sequence of atomic actions leading precisely to the rewarded state.

Thus the standard way of dealing with long term rewards by artificially distributing them along traces [92] can be completely avoided in the case of hierarchically distributed goal-directed inference with enough granularity, because there is always a level of abstraction on which the reward is visible in a few steps [49].

F. Exploration and the Influence Model

The previous sections described how the Expert is able to see into the future and decide what to predict in order to reach preferred states. However, because of the stochasticity of the environment (Section III-C) and because the actions are not expressed explicitly (Appendix A-C), the Expert does not know how much power it has to influence the future states $\mathbf{x}(t+1),\dots,\mathbf{x}(t+T_f)$ by sending the desired goal signal $Go(t)$. For this reason, it is not possible to use the passive model described in Appendix A-B which does not contain this information.

[36]Any type of reward decomposition [17] would be subject of a future work.

Instead, each Expert learns a separate *influence model* which captures the conditional probability that a step $T_h + f, f \in 1 \,..\, T_f$ in a sequence $s_i = v_{i_1}, \ldots, v_{i_{T_h}}, v_{i_{T_h+1}} \ldots, v_{i_{m=T_h+T_f+1}}$ will happen $\Big($i.e., that $\mathbf{x}(t+f) = \delta(v_{i_{T_h+f+1}})\Big)$ given that all the previous steps $0, 1, \ldots, T_h + f - 1$ will have happened, the sequence of contexts will have been $\hat{\mathbf{c}} = \hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2, \ldots, \hat{\mathbf{c}}_{T_h+f+1}$ and the Expert will have actively tried to perform this step —it will have predicted it on its Goal output $Go(t+f-1) = \delta(v_{i_{T_h+f+1}})$:

$$P^I_{\text{pr}}(f) = P^I_{\text{pr}}\Big(\mathbf{x}(t+f) = \delta(v_{i_{T_h+f+1}})| \\ \mathbf{x}(t-T_h+d)) = \delta(v_{i_{d+1}}) \,\forall d = 0 \,..\, T_h + f - 1, \\ \mathbf{c}(t-T_h+e)) = \hat{\mathbf{c}}_e \,\forall e = 0 \,..\, T_h + f, \\ Go(t+f-1) = \delta(v_{i_{Th+f+1}})\Big). \quad (33)$$

It practice, we store the number of successful transitions observed during the agent's lifetime for each dimension of the context independently and then compute the resulting value $P^I_{\text{pr}}(f)$ for each $f \in 1 \,..\, T_f$ in the same way as in Appendix A-B.

The probability $P^G(s_i)$ from the Eq. (29) is then computed as:

$$P^G(s_j) = P(s_j|\mathbf{x}(t-T_h:t)) \cdot \prod_{i=1..f} P^I_{\text{pr}}(f), \quad (34)$$

where $P(s_j|\mathbf{x}(t-T_h:t))$ is computed as in Eq. (11) and f is taken from Eq. (28).

This influence model is then used in Eq. (30) in all Experts in the situation when the agent is supposed to act.

However, because each Expert tries to maximize the reward in a greedy way (see Appendix A-E), its behavior would be biased towards the first rewards it found and the sequences leading potentially to higher rewards might be never explored. For this reason, it is necessary to add an explicit exploration mechanism which ensures that the influence model from Eq. (33) is updated evenly for all sequences.

The exploration can also utilize the fact that the learned model is represented in a hierarchical manner. By performing a random walk strategy (do a random action with exploration probability ϵ) typically used in RL systems in a distributed manner, we obtain a powerful exploration mechanism. Because doing a random step in an Expert high in the hierarchy means performing a whole complex policy [5], because such an Expert has a highly compressed and abstract representation of the world, where each cluster comprises of potentially multiple sequences of clusters on the level below, which themselves represent sequences of spatio-temporal representations from the lower layer, etc.

G. Unsupervised Learning of Disentangled Representations

This chapter does not deal with active model anymore, rather, it describes a passive learning mechanism. It presents an optional mechanism which adds the ability to learn disentangled representation of observed data in an unsupervised way. It enables multiple Experts to efficiently decompose the input space into parts which are generated by independent hidden processes. Learning disentangled representations is vital for modelling the world in efficient and compositional manner (as shown e.g. in [40], [95]).

Compared to recent models based on Deep Learning (DL) [39], the presented approach is based on simple Experts and therefore is not so powerful. In this optional setting, multiple Experts can compete for the same observations by the mechanism inspired in predictive coding [31], [90], but it can be also related to Dynamic Routing between Capsules [83].

The Spatial Pooler of an Expert $i \in \mathcal{G}$ (a set of Experts called *predictive group*) receives sequence of raw observations $\mathbf{o}^i(t)$ and learns the cluster centers $\mathbf{V}^i$. The output of the Spatial Pooler (a hidden state $\mathbf{x}^i(t)$ of the Expert) is then one-hot vector determining index of the closest learned pattern:

$$\mathbf{x}^i(t) = \mathbf{f}^i(\mathbf{o}^i(t)) = \underset{\mathbf{v}^i_j \in V^i}{\text{argmin}} \left(\text{dist}\left(\mathbf{v}^i_j, \mathbf{o}^i(t)\right)\right). \quad (35)$$

Here, the SP is used as an approximation of a generative model, therefore we define also a generative function, which takes the winning cluster $\mathbf{x}^i(t)$ and projects it into the input space:

$$\hat{\mathbf{o}}^i(t) = \mathbf{f}'^i(\mathbf{x}^i(t)) = \mathbf{x}^i(t)\mathbf{V}^i, \quad (36)$$

in case the $\mathbf{x}^i(t)$ is one-hot k-dimensional row vector defining the currently winning cluster center (see the Eq. (35)) and $\mathbf{V}^i$ is a matrix containing one cluster center on each row, the multiplication results in the corresponding cluster center in the input space $\hat{\mathbf{o}}^i(t)$.

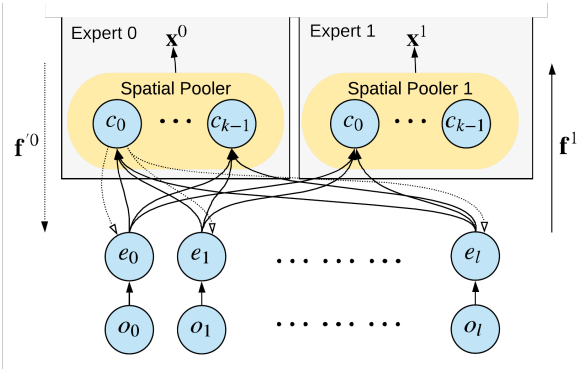


Fig. 22. Illustration of the predictive group comparable to the predictive coding algorithms. The Experts operate as usual (learning, inference), the forward/backward pass is just called iteratively for each observation.

With this formalism, we can define the following two approximations of the predictive coding algorithms. One can think of the proposed algorithm as a group-restricted sparse coding version of them:

1) Rao and Ballard approximation: This approximation of Rao and Ballard's algorithm makes the interaction of the hidden units compatible with the Spatial Poolers. The interaction between Spatial Poolers of Experts is iterative, after the input is presented, the Spatial Poolers compete for the data as follows:

$$\begin{aligned} \mathbf{e}(t) &\leftarrow \mathbf{o}(t) - \sum_{i \in [1..n]} \hat{\mathbf{o}}^i(t), \\ \mathbf{x}^i(t) &\leftarrow \mathbf{f}\left(\hat{\mathbf{o}}^i(t) + \mu \mathbf{e}(t)\right) \end{aligned} \tag{37}$$

Similarly to the original equations shown in [90], the error vector $\mathbf{e}(t)$ is computed as a difference between the observation $\mathbf{o}(t)$ and sum of reconstructions $\hat{\mathbf{o}}^i(t)$ of all the Experts in the predictive group. This means that during the iteration, each Expert receives part of the observation it is able to reconstruct $\hat{\mathbf{o}}^i(t)$ plus the overall residual error $\mathbf{e}(t)$. Note that in this notation, the t denotes time step of the input observation $\mathbf{o}(t)$, the architecture can make multiple of these iterations during one time step t.

2) PC/BC-DIM approximation: Second method of disentangling the k-means representation is based on approximation of the Predictive Coding/Biased Competition-Divisive Input Modulation (PC/BC-DIM) algorithm:

$$\begin{aligned} \mathbf{e}(t) &\leftarrow \mathbf{o}(t) \oslash \left(\boldsymbol{\epsilon}_2 + \sum_{i \in [1..n]} \hat{\mathbf{o}}^i(t)\right), \\ \mathbf{x}^i(t) &\leftarrow \mathbf{f}\left(\left(\boldsymbol{\epsilon}_1 + \hat{\mathbf{o}}^i(t)\right) \otimes \mathbf{e}(t)\right). \end{aligned} \tag{38}$$

Note that the original version (shown in [90]) encourages to increase activity of those hidden neurons $\mathbf{y}$ which are both active and are able to mitigate the residual error $\mathbf{e}$ well. Compared to this, our version is computing element-wise product in the original input space $\mathbf{o}(t)$. This means that each Expert receives higher values at positions at $\mathbf{o}(t)$ which it already reconstructs and which contain some residual error. The first part of the equation remains similar with the original version: it amplifies parts of the input which are not reconstructed well yet.

The resulting mechanism enables to represent the observations in a compositioal way. It was experimentally shown that in case the input is generated by N independent latent factors and N Experts are used, the architecture is able to represent each factor in one Expert. Compared to this, in case the input is generated by just M latent factors, where $M < N$ the group of N Experts will form a **sparse distributed representation** of the input.