LEARNING TRANSFERABLE META-POLICIES FOR HIERARCHICAL TASK DECOMPOSITION AND PLANNING COMPOSITION

by

PREDRAG DJURDJEVIC

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2019

Acknowledgements

I would like to extend profound appreciation towards my advisor, Dr Huber, whose unique blend of exceptional subject matter expertise, enlightening philosophical wisdom, humor and patience inspired the creative energy needed to make this research finally materialize. Also to my committee members, Dr Zaruba, Dr Kamangar and Dr Levine for contributing the insightful and open-minded questions and comments that provided additional critical opinions and perspective to this research.

November 26, 2019

Abstract

LEARNING TRANSFERABLE META-POLICIES FOR HIERARCHICAL TASK DECOMPOSITION AND PLANNING COMPOSITION

Predrag Djurdjevic, PhD

The University of Texas at Arlington, 2019

Supervising Professor: Manfred Huber

In real world scenarios where situated agents are faced with dynamic, high-dimensional, partially observable environments with action and reward uncertainty, the traditional states space Reinforcement Learning (RL) becomes easily prohibitively large for policy learning. In such scenarios, addressing the curse of dimensionality and eventual transfer to closely related tasks is one of the principal challenges and motivations for Hierarchical Reinforcement Learning (HRL). The prime appeal of hierarchal and particularly recursive approaches is in effective factored state, transition and reward representations which abstract out aspects that are not relevant to subtasks and allow potential transfer of skills which represent solutions to potential task subspaces. With the advent of deep learning techniques, a range of techniques for representation learning have become available for a range of problems, mostly in supervised learning applications, however, relatively little has been applied in the context of hierarchical Reinforcement Learning where different time scales are important and where limited access to large training data sets and reduced feedback has made learning on these structures difficult. Moreover, the addition of partial observability and the corresponding need to encode memory through recurrent connections further increase this complexity and very limited work in this direction exists.

This dissertation investigates the use of recurrent deep learning structures to automatically learn hierarchical state and policy structures without the need for supervised data in the context of Reinforcement Learning problems. In particular, it proposes and evaluates two novel network architectures, one based on Conditional Restricted Boltzmann Machines (CRBM) and one using a Multidimensional Multidirectional Multiscale LSTM network. Experiments using a very sparsely observable version of the common taxi domain problem show the potential of the architectures and illustrate its ability to build hierarchical, reusable representations both in terms of state representations and learned policy actions.

Table of Contents

1 Introduction

Intelligent agents placed in real world scenarios are over their lifetime faced with multiple tasks and challenges that require simultaneous modeling and control of initially unknown environments with potentially very complex dynamics, observed via usually incomplete and uncertain observations and interactions. Since the world is characterized as a high-dimensional, partially observable environment with sensor and action uncertainty, delays, and with features that can be aliased, correlated or irrelevant, the search space for traditional policy learning is subject to the curse of dimensionality. For traditional MDP based Reinforcement Learning (RL), this leads to a scaling problem which makes learning of policies in these scenarios computationally intractable. In order to address this in a life-long learning setting it is imperative for the agent to efficiently acquire and reuse latent knowledge from previous experiences, such as reward structures and salient control policies, in new tasks where they are applicable. One way to address this is through Hierarchical Reinforcement Learning (HRL) which embellishes the traditional RL with a hierarchical, model-based approach to state, rewards and policy representation in order to minimize the number of decisions that have to be made and hence to reduce the policy search space and time. This proposal presents novel learning structures to automatically learn representations for use in Hierarchical Reinforcement Learning without the need for prior models or supervised training data. In particular, it first presents a model base on Conditional Restricted Boltzmann Machines (CRBM) which can learn abstract state features and policy triggers in an unsupervised way from learned policy traces. This allows it to decompose policies into separate components that could be used as reusable policies and policy triggers. However, this approach has limitations in terms of its applicability to directly learn policies in the RL framework due to limitations in the training algorithms for CRBMs, but rather lends itself mainly to the a posteriori policy an state abstractions. To address these issues, a second novel learning structure and approach based on Multidimensional Multidirectional Multiscale Long Short Term Memory (M3LSTM) is subsequently introduced which provides an advancement to hierarchical, factored state representation and policy composition for automatic state and reward abstraction and transferring of abstract policies in multitask HRL. The underlying model proposed here provides a uniform means to simultaneously learn both policies and associated abstract state features and allows to learn hierarchical skills as well as execute the learned policies within a consistent, uniform network structure. In this model, learning can be performed incrementally from basic grounded features to more complex abstract policies based on automatically extracted latent states and rewards.

Contributions are made in using both the discriminative, generative and temporal inference capabilities of LSTMs to enable seamless subsumption of abstract, goal-oriented, grounded features and action oriented (sub) policies. Here the terms partial plan, subpolicy, macro-action, and option are used interchangeably. In this dissertation we present a LSTM structure and associated training regimen to allow the learning of hierarchical control, state, and reward structures from hierarchical reinforcement learning tasks. In the proposed network structure, latent states, rewards, and strategies are formed automatically from exposures to the task executions, representing hierarchical policy and state structures in successive layers of the network. Here higher level abstractions build naturally on ones formed in the lower layers. The resulting network provides a uniform structure for both learning of state and skill abstractions, for their execution and application in new contexts, and for the prediction or simulation of policy executions and their effects on the state of the environment. We will show how the joint unit and layer learning capabilities of LSTM make this an ideal system to gradually increase the learned policy space and to gradually build hierarchical policies and representations for increasingly complex tasks.

In the remainder of this dissertation we will first introduce the basics of HRL and deep learning architectures before introducing and presenting our work on learning hierarchical state and action abstractions using a novel architecture based on Conditional Restricted Boltzmann Machines (CRBMs). After illustrating their potential on a taxi domain problem and discussing some of the challenges with this architecture in terms of using it to also encapsulate the Reinforcement Learning process, the document will discuss recurrent Neural Networks and LSTMs as an architectural element in the context of HRL. It will then introduce our specific M3LSTM structure for use in HRL and illustrate the operation and characteristics of the proposed structure with an example in a reduced scale taxi domain problem.

2 Prior Art

In real world scenarios where an agent is faced with dynamically complex, high-dimensional, and partially observable environments and with action uncertainty, the traditional enumerated MDP state space becomes easily prohibitively large for traditional policy learning, especially in the context of Reinforcement Learning (RL) settings. Addressing this curse of dimensionality in such scenarios is one of the principal challenges and motivations for Hierarchical Reinforcement Learning (HRL) approaches [22]. The prime appeal of hierarchal approaches is in the speedup and complexity reduction that is achieved largely through state and reward representations which abstract out aspects that are not relevant to subtasks [22][1][2][3], and through the transfer of skills as new macro actions which represent solutions to potential task subspaces. This ability of the learning and inference process to construct policies as sequences of subtasks which lead the system through a sequence of subspaces or subgoal states, leads to a reduction of the policy search space due to the tree-like structure of the policy hierarchy and due to guided search. This reduction is possible because entire subspaces can be delegated to subpolicies in a recursive, nested fashion, such that all of the decisions associated with the subspace can be replaced by one decision. Moreover, the set of applicable subpolicies can be pruned not just based on their utility with respect to rewards, but based on the prerequisites and likelihood of success given the current execution context and subgoal, further reducing the search space and leading to accelerated learning. If policy execution can be further parameterized with the overall goal and the lower level context, then the number of policies can be even further reduced. However, the potential power or HRL does not only lie in the use of learned policies as actions in future tasks but also in the corresponding potential for the use of increasingly more complex state features to be sufficient to represent the information necessary to select between these abstract actions as entire subspaces of the state space are handled by the policies and are thus no longer necessary for decision making the abstract policies.

Recently, a number of computational approaches have been developed to address the scaling problem in RL by building spatial and temporal abstractions of either the state space, actions, or the reward structure that serve the purpose of either clustering or factoring the policy functions into reusable subroutines. In general the purpose of these techniques is to either decompose the overall space into subspaces or to decompose a utility function over actions into factored components.

Some of the methods focus on factored or compressed state representations [1][2][5][7][10][12][17][18][18][20], while others use guided or shaped exploration strategies [8][9][13][15][16]. In both cases the issue still remains how to transfer control knowledge from previously learned to new tasks and how to generalize the policy to allow for transfer and to keep the cardinality of (macro)actions limited. To address this issue, a number of methods for hierarchical abstractions of the state and action spaces have recently been introduced [2][4][6][7][11][14][19][22][23], some of which achieve this by including the information from previous tasks [2][10][17]. A more comprehensive review of the approaches above is provided in [22]. In some cases the action abstraction is based on extracting subgoals and subgoal skills [3][5][6][8][9][11][14][24].

The complexity reduction is here achieved through the hierarchical decomposition of the original task, but if the hierarchy is rebuilt for every new task, the acceleration would only happen towards the end of learning, as most of the learning would have happened in a largely flat space. For significant speedup in building the hierarchy to occur, a mechanism to establish equivalence between (sub)tasks, and to generalize the policy needs to be inherently present to facilitate efficient transfer. However, very few control learning approaches exist that natively contain both of these abilities in a consistent, integrated framework and which allow for generative construction of representations as well as discriminative and learning capabilities. In contrast to applications in control learning, in the domain of image, video and document classification and recognition there have been a number of techniques used to automatically form object hierarchies based on the object parts models. However, these usually require labeled samples and are typically not all temporal in nature [41]. On the other hand, several models allow for generative modeling of multidimensional sequences which strongly resemble, at least on the visible level, the transitional model of a policy that involves choosing the dynamic model and the initial states in order to generate a sequence of states [26][27][28][30]. More specifically approaches to abstraction learning for sequence data could be divided into several categories based on the types of abstractions they form and the type of learning used to obtain them.

2.1 Temporal Abstractions

Learning and executing at different levels of temporal abstraction is an open challenge in problems involving long-range planning. Some of the seminal work in the area of Hierarchical Reinforcement Learning (HRL) is summarized in [38]. Among them, Sutton et al. [44] proposed the options framework, which introduced multi-step action subpolicies (options), extending the MDP framework to a semi-Markov decision process (SMDP). MAXQ framework [39] decomposed the value function of an MDP into combinations of value functions of constituent MDPs, allowing it to operate hierarchically with a set of policies taking the role of actions. Along a different line or work, Guestrin et al. [40] developed a factored MDP formulation to permit better scaling to complex domains. To address incomplete state information, Hernandez and Mahadevan [41] further combine hierarchies with memory to handle partial observations. To facilitate more automatic skill learning and hierarchical decomposition of control learning problems, McGovern and Barto [46] proposed an information theoretic approach for learning useful subgoals, and thus potentially useful subpolicies, based on fan-in and fan-out statistics of funnel states. In Huber and Asadi [2] propose a method for identifying options based on extended use of one action. More recently, several methods have been proposed to learn options in real-time by using varying reward functions [45] or by composing existing options [43]. Value functions have also been generalized to consider goals along with states [42].

2.2 Unsupervised Intrinsic Learning

Singh et al. [53] explored intrinsic reward structures in order to learn generic options that can apply to a variety of tasks. In [52] they explored an evolutionary optimization of reward functions in the context of extrinsically and intrinsically motivated behavior. Goel and Huber [48] discuss sub-goal discovery using the structural aspects of a learned policy model. Schmidhuber [51] provides a coherent formulation of intrinsic motivation, measured by improvements to a predictive world model. Rezende [49], Frank et al. [47] have recently proposed intrinsically motivated learning within the framework of mutual information maximization. Oudeyer et al. [50] categorize intrinsic approaches into knowledge based methods, competence or goal based methods, and morphological methods. These approaches derive the local reinforcement signals indirectly by estimates of how diverse the options are between each other (specialization), and how often they result in predictable outcomes (controllability).

2.3 Object-Based and Relational RL

Diuk et al. [55] and [54] propose an Object-Oriented MDP using a representation based on objects and their interactions, defining each state as a set of value assignments to all possible relations between objects. Their representation is similar to that of Guestrin et al. [56], who describe an object-based representation in the context of planning. Not directly related to planning but rather perception is Suddert [57], which introduces a Bayesian model with the addition of relative object location. Within the same area of perceptual action detection, there have been recent developments in pyramidal clustering of actions in order to recognize long term activities and in the use of Temporal Convolution [58]. However, this too requires that elemental object detection already has supervised learning object types and that supervised, labeled data is available in sufficient quantity, an assumption that is often not realistic in Reinforcement Learning applications.

2.4 Deep Q Reinforcement Learning Networks

Deep Q-learning Networks (DQN) have been successfully applied to various domains, including Atari games [75] and Go. These still perform poorly in environments that are deeply POMDP and also have sparse, delayed reward signals. One of the reasons of poor performance on tasks that have a lot of depth before reaching the goal and hence represent scenarios where the reward signal is very scarce is that DQN compresses input mostly in space due to its predominant use in domains where state is derived from images. On the other hand, DQN applications tend to be very shallow in time with time treated, if at all, as a very short spatial dimension. In other words, DQN works well only in cases where short history is sufficient to determine the next action. A second shortcoming of DQN is that it does not compress the space based on subgoals or the action space based on discovered options. This makes the learned models rigid and brittle to even the slightest changes in the rules of the game and thus not transferable to others. Some recent work by A. Graves [62] in neural Turing machines has attempted to solve the problem of long term memory. There are also some recent developments in temporal convolutional networks (TCN) [58], where higher layers have more temporally dilated filters that go further back in time at the expense of lower resolution relative to more recent input. The rate of dilation can be 2 per vertical layer which makes the depth of history grow exponentially, but at the expense of courser details in the past. Another development is clockwork networks [79] which have an explicit sampling time cadence at different levels of the network. However, while cadence can grow exponentially longer into history at higher layers in these networks, the representation may not coincide with actual clustering boundaries of higher states over time.

2.5 Current Problems with HRL and Deep Learning

While all this work has led to significant progress in HRL, most approaches still lack in clarity and uniformity, requiring mixtures of techniques or heuristics to address the different challenges or limiting their capabilities and operation to only some of the aspects of hierarchical abstraction. Issues with building hierarchical models automatically for use in HRL include that it typically comes at high computational or sampling cost. It also often requires heuristics to come up with measures of similarity or significance of states in order to derive hierarchical representations, and that the initial cost of forming hierarchies is often not recuperated when solving the next related problem. In particular most existing approaches still struggle with issues in building and transferring increasingly abstract state, action, goal, and reward representations.

2.5.1 Subgoals

One issue in HRL systems is finding virtual goals that implicitly contribute to reaching the final goal and may have a detectable feature signature. Many approaches have emerged that are intuitively and statistically plausible [38], but typically still require heuristics to discover the latent subgoal attributes or a measure of subgoal salience. In some cases the subgoals and options can be associated with certain perceptual features, such as key points, landmarks, or surprise events; in others, features are in the exocentic not egocentric or oracle view of the world based on hidden states/dynamics and memory.

2.5.2 Reuse

A second problem is that finding the (recursively) optimal hierarchy of tasks requires many trials across instances of a task, with most of the learning happening in the initial, flat state space. After learning, the topology may be optimal but overspecialized for one task which can lead to behavioral proliferation and thus an excessive space of actions, most of which are too context specific to be applicable in new tasks. Specialization of options to tasks is difficult since there are many if not infinite possible partitions of subpolicies, and still does not solve the issue in itself. To more directly address the action proliferation issue would require generalization in order for policies to be reusable, where generalizations contain parametric inputs that can dynamically change its behavior by adapting to both sublogal and current lower state information.

2.5.3 Options

A further problem is that once the subpolicies (or macro-actions) have been found, the controlling mechanism that starts, ends and transitions from one to another at runtime has to be in place and sometimes it is not clear how to deterministically ensure that the flow of control information from top to bottom as well as the flow of (abstracted) sensory information from bottom to top is taken into account without re-planning from scratch. Structuring this flow should enable options to utilize subgoals as parameters, to output to higher-level policies the summary information of lower level state, and to recognize the right points when to communicate, how to recognize the states in which to engage, and when the task has ended. Some aspects outside the rewards are a factor in determining this information flow. In particular, predictability of the end state or of the effect on the current state as well as how distinct the effects are from the ones of the other options can have an influence on some of the decisions.

2.5.4 Feature Compression

Deep network models often have issues with their representation of state-action-reward space when put in the context of HRL. Most of the DQN models tend to compress only the input sensory features and have shallow depth over time, rather than to holistically compress the dynamics of the state-action policy space into emergent structured models. At the same time deep learning models that work well for vision and speech, often do not translate directly into mapping sensory, state, and action spaces. When applied to such data, they not just lack in terms of input-output performance but also in terms of the filters' or layers' ability to extract hidden representations of the dynamics of the system such as subgoals and options, that are also human interpretable. This is evident in the somewhat related task of perceptual classification of video action/activity detection [87], where extending convolution through time does not provide the desired results but rather significant filter, layer, and module structure definitions and supervised learning are required to achieve acceptable results.

2.5.5 Feedback Loops

The structure and nature of RL control learning also introduces a number of extrinsic difficulties that are not present in the same way in supervised perceptual recognition tasks. These are further exacerbated by the fact that in RL control tasks the training set is often dynamic and due to policy changes non-stationary and imbalanced due to poor initial policy, with several feedback loops of control and a range of information flows. There are many execution/learning loops that are stochastic and depend on initial conditions, agent action and world sensor reaction. There could even be many agents that react to other agents' actions. Other components that can produce tight interleaved issues include actor critic loops, on policy and off policy values, recursive loops when recursive neural nets are used, as well as loops between exploration and exploitation. When the agent is mostly exploring, the state-action modeling can be learned, but rewards and utility functions are not efficiently exposed. Conversely, when mostly exploiting the current policy, the value function is likely stuck in a local maximum and can only be improved to a limited amount. With sparse rewards, which make it difficult to efficiently back up the value function to previous states and actions, often reward shaping and heuristic intrinsic rewards are used to accelerate the process. These, however, can lead to surprising control strategies that formally optimize the rewards but lead to rather disappointing anthropomorphized performance [83].

2.5.6 Mapping

Learning with deep and recurrent networks also yield intrinsic difficulties. The problem here is that changing the value of one state-action also inadvertently, outside of the Bellman equation, changes the value of neighboring points, which can lead to bias in exploration and over optimism in values which, in turn, leads to a positive feedback loop that further exaggerates the bias.

To address these issues, there are two elements to consider that should both be present in a successful approach, namely implicit decomposition of the task into subtasks and, just as importantly, policy synthesis where equivalent task segments can be found in other tasks and corresponding subpolicies transferred. The algorithm of building a task and policy space hierarchy thus needs an inherent mechanism for discovering the latent state features as well as the subspaces or subgoals that are useful across multiple tasks. It also needs mechanisms for determining how to build models from simpler to more complex tasks, and for generalizing reusable policies and applying them when it is useful.

Some of these mechanisms, such as decomposing a problem into subgoals, building complex policies from simple ones, and having subpolicies generalize across tasks can be implemented by adapting the actor-critic framework and adding modeling and clustering capability, as well as bottom-up and top-down signaling. Networks that can be used for modeling state-action-reward space, subgoals, options, and affordances of options, need to be layered both spatially and temporally and exhibit layered compression capabilities as well as discriminative and generative capability. Some examples of such networks are CRBM and LSTM.

3 Compositional Hierarchical Multiscale Recursive Networks for HRL

Hierarchical RL and DQN learning is a very active field where many narrow AI problems have and are being solved such as arcade video games, traditional table games, (multi)agent grid worlds, automatic robot control and navigation. The recent advances in deep, recurrent networks and hardware as well as advances in Q and actor-critic learning have made ascendance of these solutions appear within reach. Unfortunately, the reality is that few of those solutions can be practically applied to real physical world situations for several reasons. The number of trials required is time and cost prohibitive, particularly compared to humans. Once the controller or policy is learned, it overfits in a sense that it is not particularly reusable. Another problem arises in real world problem in that most of the problems are partially observable to a much larger degree than the simulated games or grid worlds and thus are better described as POMDPs. As a result, they require active attention, long term memory and some level of unsupervised learning. There is a significant explainability gap between black box solutions where mapping is learned but inner structure and logic is not clear, vs. white box solutions with clear objects and relations but where entities have to be defined first. Most of the work in HRL is based on the notion of compositional nature of the world and hence assumes that the plans could be compositional too.

The impetus for the methods introduced here is to be able to learn at different levels of abstraction, with the expectation that plans on the highest levels would be semantically plausible, and that lower level plans would be transferable by parameterization. This means learning both long term dependencies over space and time, and also hierarchical dependencies over vertical layers. The higher layers have slower cadence of changes and therefore their activation represent partitioning or summarization of the state and activity of lower layers. On the other hand, the lower layers have a more detailed view, both spatially and temporally, and provide spatio-temporal abstractions for the higher layers. This constant bottom-up and top- down flow of information gives states and actions of the lower policy the context of the higher states (subgoals) at certain times and conversely higher states get context from lower layers to transition to the next state at certain times. Some networks like CRBM, Feature Pyramid Networks (FPN), allow bidirectional flow of information between layers, and bidirectional flow between past and future states (when offline). This work proposes two architectures that use recurrent network structures to address learning policy and state hierarchies in HRL. In particular, it first introduces an architecture based on CRBMs that permits to learn state and action abstractions from task executions in an unsupervised fashion. To address some of the limitations in this approach, it then proposes an approach that uses the mechanisms found in LSTM for input, forgetting and output gating, and builds on them towards layer wise gating in order to facilitate vertical layer, temporal gating and hence unsupervised or semi-supervise clustering over space and time. In addition to gated vertical flow of information, there are other mechanisms of encapsulation of policies that are outside of pure extrinsic rewards such as intrinsic rewards and regularization that take into account the temporal and spatial characteristics of policies or options.

There are a number of the potential reinforcement regularizers that can be useful in HRL. One of them involves the prerequisite initial state to start an option. Second one is involves commitment to the option determines the duration of staying in the corresponding subpolicy and in reaching its end state. Third one is the probability distribution of the expectation of the final states after option ends, or the delta in factored states or observations from the initial state to observations at the end of an option.. While identifying subgoals could take into account characteristics of states like their input output transition flow, reward gradient, distinctive features (landmarks, novelty), delta change in factored states reflect more the affordance of the option, similar to elemental actions. Fourth, the distinctiveness of the option, how different is it from other options, can here be expressed not in trajectories or actions executed, but in terms of affordances or end effects. One difficulty in identifying subgoals is that many times solutions gravitate toward too few or too many subgoals, and require careful regularization. The same applies to options, since learning the start of it, the subpolicy, and end of it, has an inherent interplay between the upper level options, option itself and lower level options.

3.1 Conditional Restricted Boltzmann Machine (CRBM) Background

The first network structure proposed here utilizes CRBM's with their bidirectional nature and flexible gaiting mechanisms to address some of the challenges in hierarchical structures in Reinforcement Learning in a largely unsupervised fashion.

3.1.1 Conditional Restricted Boltzmann Machines

Some of the elements of learning in HRL, such as building complex policies from simple ones and having subpolicies generalize across tasks, can be implemented in Conditional Restricted Boltzmann Machines (CRBM). CRBM is a probabilistic graphical model that forms a product of experts (PoE) as an undirected multilayer graphical stochastic network [76][77]. They are derived from RBM by adding replica layers representing previous time steps, and hence the layers representing current time is additionally conditioned on previous time steps. Some distinctive characteristics of this model are that it can be used both discriminatively and generatively, as associative memory or function approximator in both top-to-bottom and bottom-up directions, and that the transfer function can be built incrementally one layer (bottom-to-top) at a time. Internal or hidden layers encode non-linearly the latent space of the visible inputs, similar to how PCA encodes the principal dimensions or features in a linear fashion. Some of these properties make CRBMs intriguing and potentially very powerful for the automatic learning of hierarchical decision policies as well as of the corresponding latent state and subgoal structures within a consistent, uniform network structure.

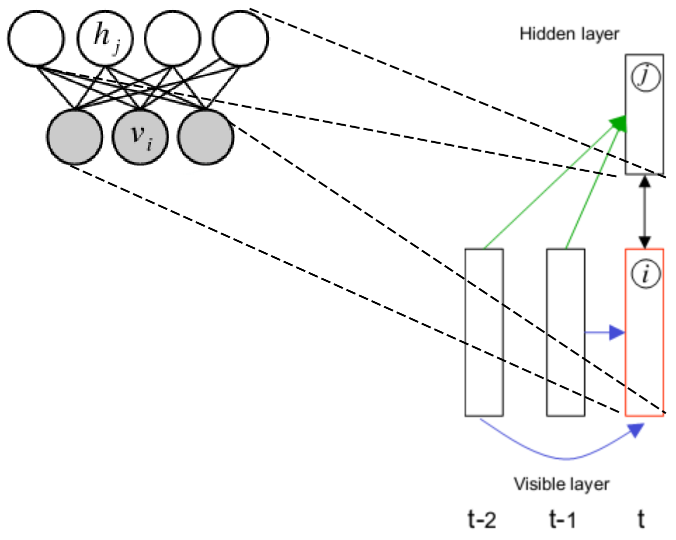

Figure 1 CRBM

Figure 1 shows a CRBM that can be used in RL if visible layers have state or observations and action as inputs, the upper left corner shows a basic RBM contained within sublayers. The t-2, t-1 are actual sublayers and not a depiction of its state over time.

3.1.1.1 RBM and Contrastive Divergence (CD)

Among many models for modeling multivariate multimodal distributions, the

Product of Experts (PoE) approach combines multiple latent-variable models by multiplication as opposed to additive aggregation like with the traditional mixture models. The probability of the data vector (*d*) is defined as

$$P(d \mid \theta) = \frac{\prod_k p_k(d \mid \theta_k)}{\sum_{d'} \prod_k p_k(d' \mid \theta_k)} \tag{1}$$

where *d*' are all possible vectors and θ_k are the parameters of the k^{th} expert. One advantage of a PoE is that the latent variables of each expert can be made conditionally independent given data and that it can produce much sharper distributions than individual experts. Provided that individual distributions are not just univariate and unimodal, the end effect is a smooth yet powerful distribution that can cover complex manifolds. Boltzmann machines represent a Markov random field which due to the probability distribution being calculated based on multiplicative aggregation of pairwise stochastic binary variable energies, implements a PoE. Even though the model is expressive, the maximum likelihood training for such a graphical model is not trivial due to conditional dependence.

There is a simple and effective alternative to maximum likelihood learning that eliminates almost all of the computation required to get samples from the equilibrium distribution and also eliminates much of the variance that masks the gradient signal. It can be shown that the negative gradient can be approximated using samples obtained by starting a Gibbs chain at a training vector and running it for one or k steps (theoretically it is ∞ steps). This means alternating between evaluating the hidden to visible direction and the visible to hidden direction.

$$\Delta w \propto \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_1 \tag{2}$$

In RBM, the nature of connections is un-directed and nodes are arranged such that layers of nodes form a bipartite graph, which ensures conditional independence of hidden nodes given data nodes. In turn, this allows, with some approximations, the max likelihood learning to be simplified to minimizing the difference between the data distribution and the model's reconstruction, and makes learning very efficient. Once one layer is learned, it acts as a visible layer to the second hidden layer that can be learned and can be shown to provide complementary priors to the first one and improve the mapping error bounds. This provides a mechanism for ever more complex and accurate transformations. The upper layers effectively encode or compress the data feature space into a more latent or abstract state space comparative to a nonlinear PCA.

Due to the undirected nature of the connections and the nature of the inference, bidirectional inference is possible based on which visible nodes are pinned to values and the rest is left to converge to the equilibrium activation. This enables the model to be used in several transformation capacities, as an associative memory, auto-encoder, classifier, or in regression. One such transform that would be relevant in the context of RL is the policy itself, given the policy (π) and state (s), infer the action, $\pi(s,a) \rightarrow [0,1]$. Another transform is to infer a higher policy ID that is being executed from a lower policy and action sequence, given state, given action sequence infer executing policy ID, $(s_1,a_1,s_2,a_2,\ldots, \pi) \rightarrow [0,1]$. Lastly, a predictive or sequencing transform, given state(s) and action infer next state $(s_{-2},a_{-2},s_{-1},a_{-1}, s_0,a_0, s_{+1}) \rightarrow [0,1]$.

3.2 A CRBM Architecture for Hierarchical Abstraction in RL

Given the capabilities of CRBMs, we propose an architecture based on multiple gaited layers of CRBMs where latent states, rewards, and strategies are formed automatically from exposures to the task executions, representing hierarchical policy and state structures in successive layers of the network. Higher level abstractions build naturally on ones formed in the lower layers. The resulting network provides a uniform structure for both learning of state and skill abstractions, for their execution and application in new contexts, and for the prediction or simulation of policy executions and their effects on the state of the environment. The incremental layer-by-layer learning capabilities of CRBMs make this a suitable system to incrementally increase the learned policy space and to incrementally build hierarchical policies and representations in increasingly complex tasks.

A modification to RBM that is needed to facilitate this is the addition of conditional nodes that represent previous node activations of the same layer and temporal connections towards current (timestamp) nodes, which gives it a sequence modeling capability. This is important in the context of policy learning as the features, including higher latent state and policy representations, are a condition for inferring the next action. Due to the temporally extended context this equates to a Markov model of higher order and consequently more information (memory) is available for more complex and accurate transforms. The same number of conditional nodes in higher layers enables even longer memory as the upper nodes may represent temporally extended states or subspaces.

Expanding from equation (1) by replacing d with state action pairs,

$$P(s,a \mid \theta) = \frac{\prod_k p_k(s,a \mid \theta_k)}{\sum_{a'} \prod_k p_k(s,a' \mid \theta_k)} \tag{3}$$

where θ_k are explicitly the parameters of the k^{th} expert, CRBMs implicitly implements PoE in a way in which hidden layer nodes present the experts

$$P(v) = \frac{e^{-F(v)}}{\sum_{v'} e^{-F(v)}} \tag{4}$$

$$F(v) = -\log \sum_h e^{-E(v,h)} \tag{5}$$

where *F* is the equilibrium free energy, and the input vector, *v*, represents the pair in the underlying state and action sequence *v*=(*s*,*a*). In this form, the network can be considered a factored representation of the policy. In the context of HRL, CRBMs represent an interesting model due to several aspects.

First, the unsupervised nature of building hidden state representations based on inputs, which in this case consist of observable state features, actions, and policy ID labels, provides an efficient means to automatically form abstractions for hierarchical policies. Second and upper layer policies can also be formed consistently were the units treat hidden state attributes as visible inputs to learn and infer higher, abstract policies. The importance of this is twofold, one the state is abstracted not just based on features but also on the dynamics of transitions and rewards, two, in the same way, the low level policies formed as experts on the first hidden layer can be used incrementally by the higher layer, abstracting policies as shared building blocks.

A second aspect that makes this network well-suited is the ability of the CRBM to perform both discriminative and generative inference. In this application this enables seamless, top-to-bottom subsumption and bottom-up switching of (sub)policies, providing important advantages. This is important as policy transitions are driven not only by the overall goal and the high level configuration of the space, but also by the low level context and outcomes. It enables use cases such as starting to execute the lower policies and classifying what the current and next best upper policies are, but also to fix the upper policy and based on the low level context infer the best intermediate policy transitions. In this form the CRBM provides the structure to perform policy execution and prediction as well as for policy and abstraction learning and inference.

A third important aspect, the representation of temporally extended MDP , achieved as indicated previously by adding nodes to layers that represent the state of the node from the previous time stamps, provides important capabilities. In particular, this is important by allowing to have higher Markov order and thus adding both shorter or longer term memory, depending on the hierarchical level at which the temporal replication occurs. This, in turn, enhances the depth of information and the decision making capability. Also, due to effective sequence modeling of world and agent, it enables short term prediction of the next states of the world given the next optimal action execution sequence. This, in a sense, can also be used as a utility or affordance predictor. In other words, by virtually playing out (macro)actions in advance of the current feature context, one can deduce which actions are likely to succeed or end up in a state with higher utility in terms of Q-values.

In the approach proposed here, which applies CRBMs in the RL domain, the basic layered structure of the CRBM as well as the Contrastive Divergence (CD) learning is unchanged. The visible and label nodes are usually Gaussian nodes due to the nature of input features, and hidden nodes are left to represent binary variables. The visible layer comprises several state feature and action nodes, while the upper layers contain labels that correspond to policy IDs and reinforcement rewards. Some label nodes can be used for policy parameterization in addition to a simple policy ID. While the reward node is not necessary since the CRBM used as the state transition and best action approximator implicitly encodes the utility (Q value) of the actions, it is nevertheless useful as a monitoring node. During learning, the policy IDs, like original actions, follow the softmax rule as only one action and one policy per layer should be active

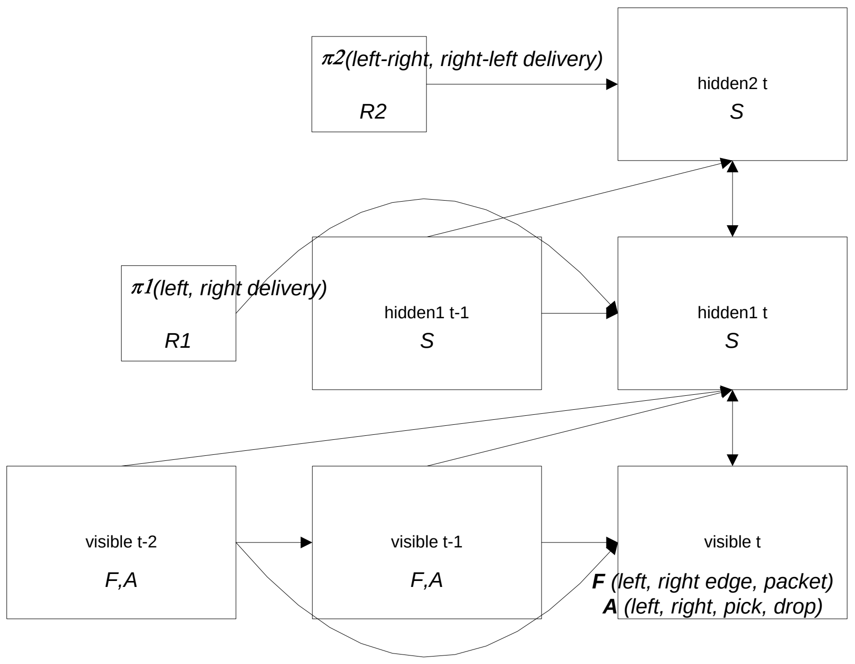

Figure 2 CRBM representing HRL

Figure 2 shows the structure of the proposed network for HRL where the visible layer has conditional replicas of the same layer's nodes shifted 1, 2 (or more) steps in time to represent past state and action features, thus facilitating abstraction of dynamics and explicit prediction. Similarly, the hidden layers have temporal replicas of themselves representing previous node values. Initially only visible and hidden layer 1 exist, later hidden layers 2, 3, and so on are added to represent incrementally higher-level abstractions and policies that can build on previously learned policies. The visible layer is presented with the feature vector F and actions A while learning, and reconstructs an action at time t given the history, hidden layer values, and F at time t. The hidden layer is presented with the policy ID, and reward signals *R*, but is during inference only initialized with 1 or 2 initial timestamp values. By setting a label node to a policy ID, setting the states to the current state ID, and the previous state at *t*-1, the network should converge to an equilibrium action *a* at current time *t*. At the same time the hidden layer should converge to the distribution of the hidden state for the current policy and current state.

In the context of a simple delivery task where a packet has to be picked up and brought to a destination, subpolicies might form that deliver to a base that is either to the right or to the left of the agent. Packet reappear anywhere on the track, and cart alternates direction looking for a packet, carrying it and delivering it at the base at left or right edge of the world. The right-delivery just means that if starting from left edge, the packet is to be picket and delivered on the right edge. Given this setup, typical operation would be setting the top policy π_2 to right-left delivery and initializing the low level features, to indicate presence or absence of the left edge. The rest of the network can be initialized to be consistent with the aforementioned settings, such as setting the lower policy π_1 to be right-delivery. Upon such initialization the network should replicate the top and middle policy by executing appropriate low level actions such as left action for example or pick action if the packet is visible or drop if the right edge is visible. Presence of the right edge in this case should also trigger the change from right-delivery to left-delivery.

3.2.1 Example

To investigate and demonstrate the capabilities at learning and representing HRL policies, a simplified cart delivery domain has been designed to represent a problem where a car operates in a 1D space, needs to pick up a packet and drop it off at a destination. The state space here is represented as a factored domain with features such as left and right edge for start and end locations, as well as the presence of a packet.

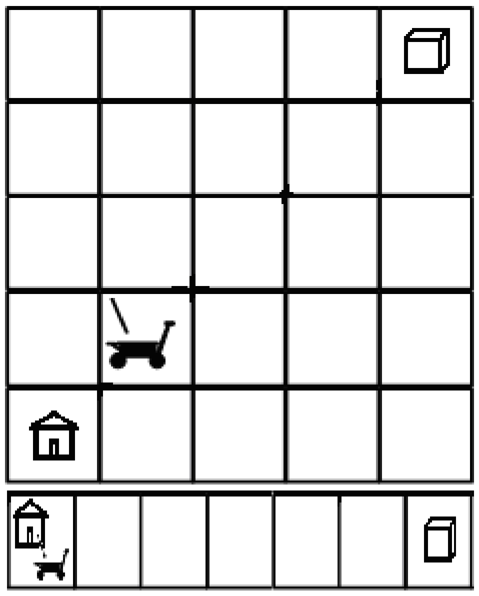

Figure 3 Taxi domain grid and linear world

The simplest low level polices are extended actions such as going from the left to the right edge, from right to left edge, picking a packet, or dropping a packet. The next level's two policies above are the ones where full right-to-left side delivery and left-to-right hand side delivery is performed. The top two policies are the ones where right-followed-by-left and left-followed-by-right delivery is performed. The intent behind the domain is to investigate incremental buildup of policies from simple to more complex while using previously acquired skills. Also, it is aimed at examining the characteristics of the latent/hidden state representation and whether the network and learning approach can elicit the utility of a subgoal in terms of an internal pseudo reward. One example is that the action of picking up a packet is a prerequisite of dropping it and in a sense has an affordance of enabling the delivery of a packet. The interesting part is that the model has to develop its own internal concept of a subgoal or, in this case, of the state of holding the packet which is not part of the feature space. At the same time, the model has to develop a long term memory of this state as the number of steps to reach the destination far exceeds the inbuilt memory encoded in the conditional part of the visible layer of the RBM, which is confined to just a small number of past timestamps. Also in training, the distances between the edges and the packet vary from short to long so that the model does not try to infer the packet possession state out of the distance alone. The rewards label is used more as a monitoring indicator to see if the model can predict the occurrence of reward rather than as a controlling label such as policy ID nodes that, if pinned to a value, would subsume the rest of the model to execute that policy.

Once the model has been primed with features and labels (in the conditional part of the network representing short node histories) that correspond to the last few steps, only the observed features are fixed for the rest of the execution. Upon every new step, the up-down inference is repeated until convergence, letting the low and high level policies execute and transition to new subpolicies. Due to the stochastic nature of the model, the policy executions and transitions are not always the same for the same input sequence, but on average follow the learned policies.

Similarly to how the reward node is predictive, the visible state action nodes could be used in a predictive manner. Instead of using the feature part of the current visible layer as an implicit conditional part in order to find the best actions, those nodes can be left unpinned and the action node's values be pinned, leading the model to in effect predict the most likely next state of the world.

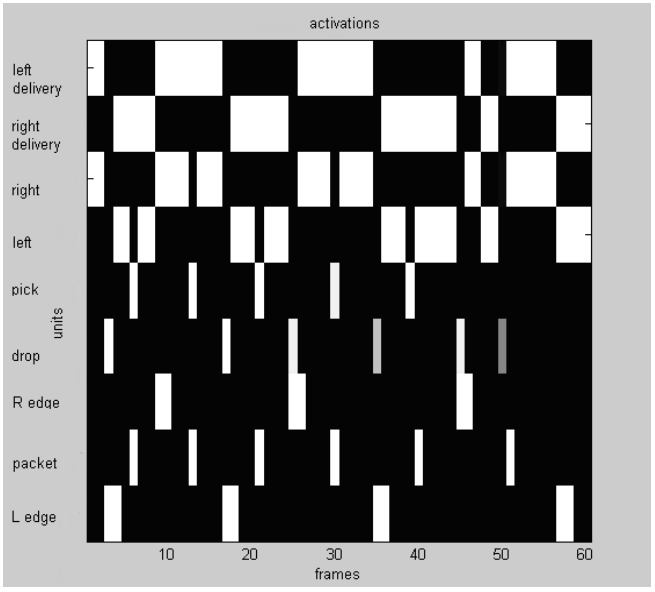

Figure 4 Taxi domain learned node activations; the bottom three rows are observations, the next three are elemental actions, and the top two are the macro action activations learned; the horizontal axis is time or steps when traversing the domain

Figure 4 shows the activations of the nodes representing features and actions after a successful policy is trained. The features represent left edge (bottom left corner), detected packet, right edge (upper right corner) and their values during a task execution are shown in the bottom three rows of the figure with white bars indicating that the features are present at the corresponding time step. The action units represent left action, right action, pick and drop and their activations are shown in the next four rows with the brightness indicating the degree to which this action is to be chosen (i.e. the action with the brightest bar is the best action at that point in time). The top two rows show the activation of the learned subpolicy nodes. The figure shows that the architecture successfully learned to represent and trigger subpolicies that successfully solve the problem. These subpolicies could now be triggered using the policy label nodes and thus the subpolicies could be transferred to and executed in new problems by triggering the corresponding label nodes. In our experiments we showed that the high level policies learned here can be applied in a larger version of this domain where the edges are further apart and will successfully solve those domains. While this is obvious to an oracle, the agent has to learn to commit to a search and delivery subpolicy, and maintain its state even its internal recursions and stochasticity, after longer periods of no changes in observations.

3.2.2 Limitations

While CRBM has many positive attributes highlighted above, there are aspects that limit its adoption in HRL.

The layers are fully connected with no structured spatial connectivity, such as e.g. convolution filters, which is adequate for relatively limited numbers of inputs that are not correlated. For images or video or 3D scans, this becomes a problem as the number of inputs it substantial (e.g. the number of pixels or voxels), and inputs are locally spatially correlated as well as potentially temporally correlated (e.g. in video). If the number of visible nodes matches the inputs, the second layer also has to have comparatively large size, and that means that the weight matrix is rather large too. This slows down contrastive convergence as there are a lot of redundant nodes activated.

It is possible to use convolutional layers to compress the input space, and have early or late mixing with other signals, but careful selection has to be made of how many layers and output nodes out of the previously trained CNN. This breaks the concept of unsupervised learning but does utilize partial transfer learning, provided that types of images have similarity between CNN and CRBM usage. This transfer applies to LSTMs too, provided that elemental convolutional filters are transferable.

The extended temporal layers or rather the number of past historical time slots have to be pre-designed and are fixed throughout the training. Based on cadence of samples and how long of a memory of events is expected, the lowest layer can become quite long. This again slows convergence.

To address some of this, it is possible to have an exponential dilation of input layers (t-1 .. t-N) to reduce complexity and help convergence, but at the expense of decimation or averaging which may dilute the signal at time t-N that had significance for time t.

While the filters evolve over the vertical layers, the activation of nodes is very distributed and stochastic and is hard to interpret as slow summarized features. Regularization such as sparsity to reduce the activity towards individual nodes alone is not sufficient to produce slow temporal and unique spatial activations. The label nodes can exhibit such behavior but those have to be pre-designed so that changes in them still break the concept of unsupervised learning and automatics clustering of activity, particularly the pattern of activity that we could associate as higher states.

3.3 Recurrent, Multiscale LSTM Networks Background

To address some of the problems and difficulties inherent in the use of undirected networks and CRBMs for HRL, the second proposed architecture utilizes directed networks, and in particular Long Short Term Memory (LSTM) networks.

3.3.1 Long Short Term Memory

Long Short Term Memory (LSTM) networks are supervised recurrent neural networks initially introduced in [64] that can store long term information, addressing the problem of diminishing or exploding gradients characteristic to traditional recurrent networks and temporal back-propagation. There are several implementations of the LSTM unit, but the main variant is that the inflow, memorization/forgetting and outflow of the information is controlled through gaiting to behave somewhat like in a finite state machines and is learned. There are several network configurations where input is always a sequence and the output is used in either a discriminative, or a generative fashion. Depending on the output layer configuration and use, the output is generating either a symbol or a sequence of symbols as in an encoder. There are also several evolutionary network configurations that deal with layering (abstraction), dimensionality (spatial compression), and direction (uni- and bi-directional).

The basic LSTM [64] has directional connections left to right (in time), representing a sequence with clear directional signal flow. Note that all the units of one layer are connected, via a signal from one step back, so when unfolded in time signals show left to right and bottom to top. Figure 5 shows a basic LSTM unit with its input, forget, and output gaits which regulate whether, and to what degree, information is used to update the information stored in the cell. Figure 6 shows the interconnectivity between LSTM units in the same layer.

Stacked LSTMs are basically layers of LSTMs stacked vertically so that individual LSTMs receive both the previous time step (sequence) and bottom to top connections (as inputs) that serve the purpose of temporal and state compression. This layout can serve the purpose of a decoder or re-encoder, and while gradients are controlled within one layer, there are some problems with gradient flow in the vertical direction, similarly to traditional recurrent networks, addressed in the grid LSTM.

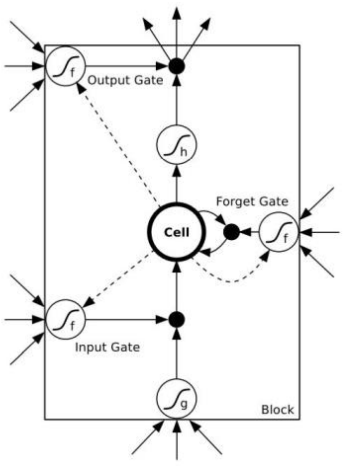

Figure 5 LSTM unit

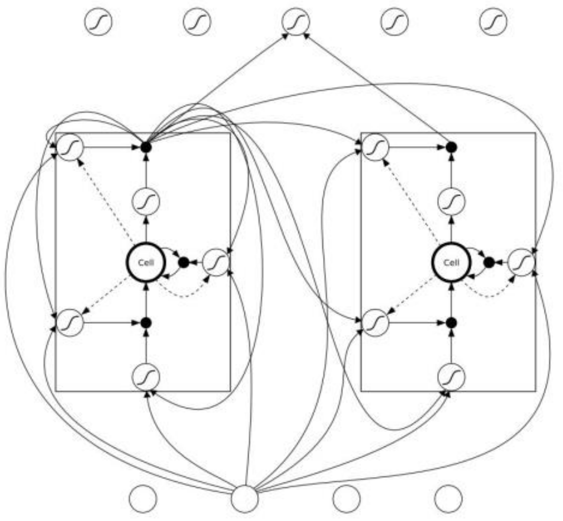

Figure 6 LSTM layer

Hierarchical LSTM [67] is also stacked but layers get progressively smaller typically at a ratio of 2 to 1 per every higher layer, similar to a pyramidal network. This allows progressive time compression as the depth of the network increases.

Multidimensional LSTM [66] extends the concept of traditional LSTM to more than one dimensions. Here the dimension of time is replaced by spatial dimensions, x and y, in the 2D case. Two directions and input from the lower layer are combined as input to LSTM units. To preserve the causality, since signals need to be ordered and cannot have direct loops, there is a traversal pattern that typically flows from one corner diagonally towards the opposite corner. However, there still is a 2^n complexity since signals still need to flow in orthogonal (90deg) directions. The original purpose of this network is the decoding of images or other grid like inputs. Figure 7 shows a stylistic representation of the information flow in a 2D Multidimensional LSTM during the calculation sweep where input from spatial neighbors and from the previous layer are combined to drive the unit.

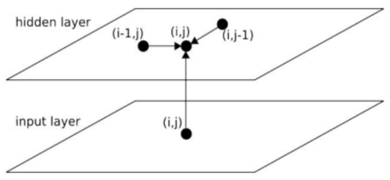

Figure 7 Multidimensional LSTM

Bidirectional LSTM [65] is a stacked LSTM where there are two layers at the same level that propagate signals along two directions before emitting a combined signal to the upper layer. The original purpose of this network is decoding sequences by having access to future information as well as past information, similar to HMMs. Figure 8 shows the basic architecture of a bidirectional LSTM with the two directional cells indicated as Forward Layer and Backward Layer. As these two directions are not interconnected except at the output, they really exist separately in the same level of the network architecture.

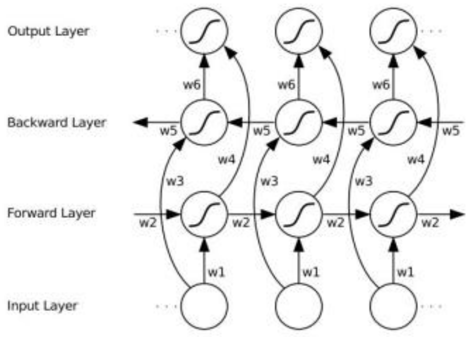

Figure 8 Bidirectional LSTM

Multidirectional LSTM [66] follows a similar pattern but with higher connectivity and are used in spatial or multidimensional LSTMs with the purpose of allowing signals from all directions to be combined at each node in the grid. Since in multidimensional LSTM signals generally flow from one corner towards the diagonally opposite corner as indicated in Figure 9 a node at a x,y is not aware only of surroundings from one side. To accomplish omni-directional neighborhood connectivity, in the 2D case 4 directional LSTM sublayers are used as shown in Figure 10. The purpose of this network is typically the decoding of images or other grid like information. The grid arrangement for sequences can be such that the input is along one dimension and the output is along another dimension.

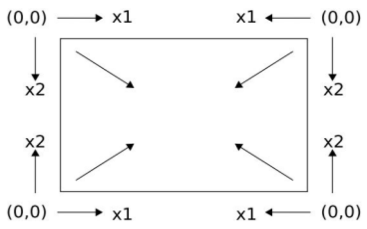

Figure 9 Multidirectional LSTM

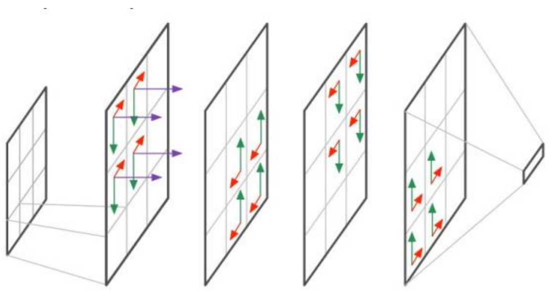

Figure 10 Multidirectional LSTM

Multiscale LSTM [72][73] is a recurrent network where there are both bottom-up and top-down connections and the vertical signal flow is governed by gates. In other words, the signal between peer nodes is mostly constant until a gate from the lower layer is triggered, and conversely a signal to the upper layer only occurs if the main gate is triggered, and that in turn changes the state of the upper level. The upper level, in turn, emits a downward signal to reset or restart the state of the lower level. The purpose of this network is to induce slower activity at higher levels and hence elicit a higher level pattern of abstraction at higher layers.

Grid LSTM [71] is a multidimensional LSTM but with some modifications as to which signals flow between LSTMs. A Grid LSTM unit is shown in Figure 11. Both the hidden (output) H and the memory (cell) C state is shared between the LSTMs in 2 or more directions or dimensions as shown in Figure 12 for a 2D configuration and in Figure

13 for a 3D configuration. In these networks, all outputs are shared among dimensions while internal memory is only shared along one dimension. The idea behind this is to regulate the flow of information in a vertical direction for two reasons, one is to solve a problem with diminishing and exploding gradients. The second is to allow the effective connectivity between layers to be learned rather than heuristically designed like inception or highway connections.

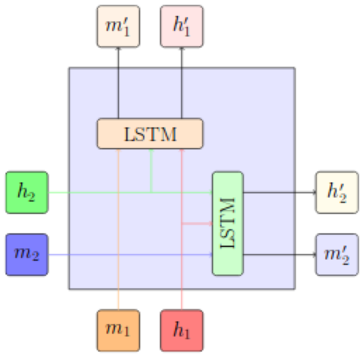

Figure 11 Grid LSTM unit

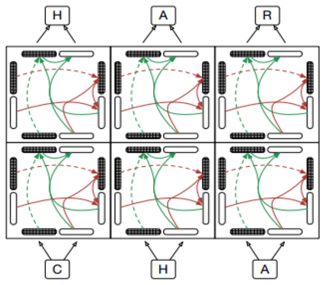

Figure 12 2D grid network

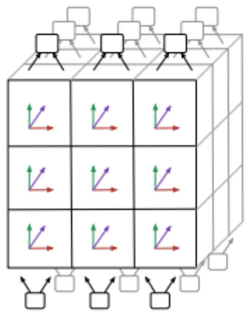

Figure 13 3D grid network

3.3.2 Multidimensional Multidirectional LSTM

In order to evaluate the multidirectional multidimensional LSTM for a task with both short and long term memory which is needed in actor and critic in the taxi domain, a simple task has been devised as supervised training. A modulo counter with 4 by 4 nodes has been constructed consisting of one main layer that in turn consist of 4 sublayers for each direction similar to Figure 9 but with 4 nodes instead of 3. For each incremental input, it is expected that the output predicts the next number, including the rollover when reaching the limit. The model is able to correctly map the input output pattern after short training. Figure 14 shows the activations during counting and illustrate the model's ability to predict counts and rollovers correctly. The learning curve in Figure 15 shows the residual error as a function of training time and demonstrates that the system can efficiently learn over time to correctly encode the problem from observations.

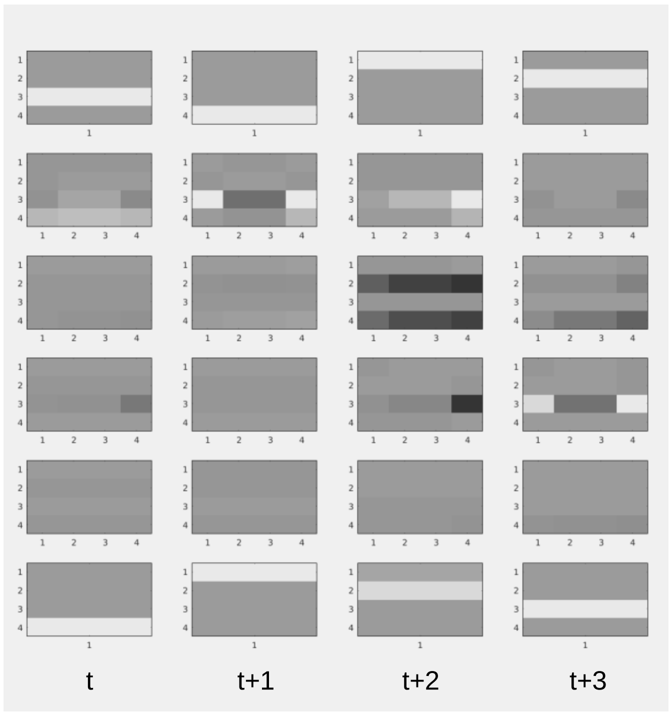

Figure 14 LSTM modulo counter, graph arranged horizontally over time, and vertically input, middle directional layers, output layer. The middle subgraphs represent activations of each node in each directional sublayer. Top subgraphs are the prediction in of the input number at the very bottom line of subgraphs.

Figure 14 LSTM modulo counter, graph arranged horizontally over time, and vertically input, middle directional layers, output layer. The middle subgraphs represent activations of each node in each directional sublayer. Top subgraphs are the prediction in of the input number at the very bottom line of subgraphs.shows the activity of the inner nodes. Each column represents a time slot or step. Within each column to bottom and top row represent input and output numbers with one hot bit, and second to fifth row within the column represent each sublayer. Each sublayer has 4 by 4 nodes, which change activity over 16 internal corner to corner propagation steps (for each input step), and the final activity at the end of 2D propagation before all are summed towards the output layer are depicted.

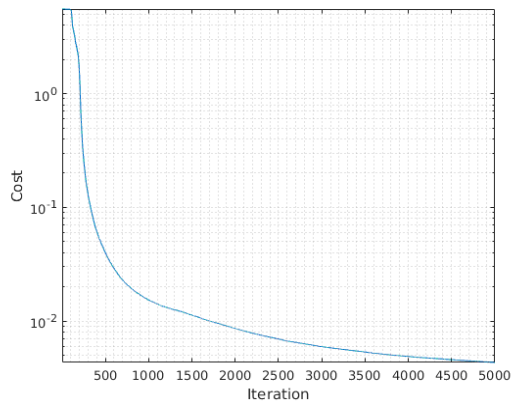

Figure 15 Training error over iterations

The multidimensional multidirectional LSTM provides the system with the ability to reason over spatial dimensions of the problem domain. In the linear taxi domain task that we used in the CRBM case and will again in the LSTM-based model as a base proof of concept problem, the domain is mostly 1D in terms of space and observations provide only one block of the grid visibility at a time. For these reasons applications in that domain will not use 2D connectivity as in the example presented here. This design is a more natural fit for the matrix taxi domain task, even though this LSTM can be used with one input to one output sequences. However, the training time of a 2D multidirectional network would be significantly higher than for a 1D model and thus the dimensionality of the model should be adjusted to the problem domain. The increase in complexity stems from the fact that for each step all directions (corners) need to propagate throughout the entire set of 2D nodes, before proceeding to the next step. In a 1D model that propagation only occurs all at once and is thus significantly faster.

3.3.3 Deep Q-learning Networks (DQN)

There are several mayor differences between traditional supervised and unsupervised or reinforcement learning networks when it comes to using deep neural networks and particularly recurrent networks for mapping the value function or policy. The differences originate partially in the dynamic nature of the error signal where, unlike in supervised learning, input output combinations constantly change due to the changing policy. Partially due to the mapping produced by the network where, unlike for tabular values, change in one state/action weight can influence other outputs, and partially due to the exploration exploitation strategy, there could be artificially optimistic Q values for frequently executed state action pairs relative to other less frequently followed but potentially better trajectories. These challenges, colloquially known as “deadly triad”, have been reviewed in Sutton and Barto [91].

Double networks [81] deal with mapping state values where lifting the value of one state (Q value) or action (policy) also inadvertently lifts neighboring states or actions because of the smooth nature of how neural networks map input output transfer functions. This positive feedback loop where exploration in turn visits those states more often, also leads to over-optimistic values of the corresponding states or actions. The idea to address this here is to have two networks, a target and a predictor network where the network anneals by freezing and using the target as a temporarily fixed policy and learning the predictor from that policy and rewards. After a number of iterations, the target is updated from the predictor.

Dueling networks [82] decompose the mapping of the Q values into a value of the state V and a value of the action A relative to other actions (advantage) in that state and predict them separately in two different higher layers of the network. The rationale here is that in some bad states the choice of action is irrelevant, and in others where the state is too early relative to the end state, the current state might not really have any effect. Analogous to classical state space control system controllability, the control signal or an action, unfolded in time does not have much effect on the future output variable. On the network mapping side, having two separate layers allows for more focused mapping and less susceptibility to fluctuation in V value. Moreover, the decomposition used addresses that actions in one state need to be normalized by value due to the effect of the network mapping similar inputs to similar values of outputs as mentioned under double networks. This is especially useful in the taxi domain where many steps may have the same external observation, and while the value function may be incorrect as an absolute value, the relative values of actions, important for argmax, are still correct.

Experience replay [87] is a means of stabilizing and accelerating learning particularly at early stages of training where there is predominance of wrong actions, exploration randomness, and scarcity of rewards, at least in the worlds where reward is given only when the end goal is accomplished without any intermediate hints. The successful trajectories are replayed in order to follow repeated positive gradients when there was reward, and new trajectories are evaluated based on state and action visitation statistics.

Prioritized sweeping [88] is a means of updating the value of those states that are likely mostly to be changing since they are most closely related by transition functions of the current trajectory. This updates the states that are most likely to be relevant to the current policy more frequently and accelerates the value iteration.

Curriculum learning [89] is another way of guiding learning through gradually more complex tasks. The idea here is that simpler tasks are tractable for learning and that related, more complex tasks can use transfer learning to evolve a working policy. If a policy is to be learned immediately in the context of a bigger or more complex task, the training or rather the policy would not have any elemental subpolicies to rely on to reduce the dimensionality or size of the search space, but would execute in an entirely flat state/action space. Using transfer to those domains can exxelerate learning by being able to build on already learned knowledge.

Intrinsic rewards [85] are a way of guiding the search under scarcity or absence of rewards, particularly at the early stages of training, but also later if the network happens to converged to local minima. The main idea here is to use not just plain state/action visitation statistics or some other subgoal statistics like landmarks, funnels, but use instead the notion of predictability of modeling the world, to explore areas of the state/action space or transitions that appear surprising and hence may provide some informational gain that helps to not repeat the same (optimistically) good trajectories that may turn out to just be only the local minima, in terms of Q values and policy.

Multiple agents [92] are a programmatic way to accelerate training by exploring the search space in parallel. Frequently sharing the gradients and learning rate so that the direction of weight deltas does not diverge is important. This is a common problem in supervised learning when the world or samples are partitioned between agents and the cumulative batch size becomes many times larger. In our case this is not a problem since agents operate in the same space and just explore random trajectories. The potential problem is diversity of trajectories since the agents do not coordinate exploration, so the efficiency of parallelism is possibly not proportional to the number of agents. This is commented on in the results section where towards the end of training when the model is more reliable and exploration is negligible, parallel agents are probably going over the same trajectories. Figure 16 shows the gradient sharing among multiple learning agents over time.

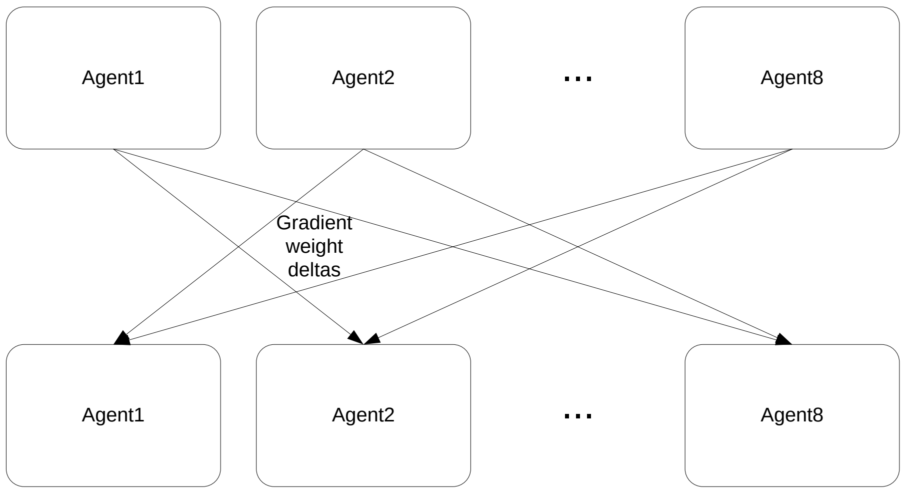

Figure 16 Multiple parallel agents training

3.3.4 Limitations

There are some limitations to traditional LSTM. One is that the feature size of the input layer even with grid LSTM is limited and in the cases of 2D or 3D image or voxel data this usually means that it needs convolutional layer to reduce the feature size. This adds complexity to choosing the right base convolutional network and adds complexity in jointly training via gradient descent. A second limitation is that all types of LSTM networks are no bidirectional in the same sense as RBMs where learned labels at the higher layer can be pinned to a value to make the lower layers generate the corresponding action sequence. This is because LSTM networks have strict input and output definitions whereas in RBMs inputs and outputs are interchangeable, thus allowing a node that was initially learned (and thus a hidden node or output) to be later set to a value (and thus* become an input). Typically when used in a generative way, LSTMs will be trained in a supervised fashion to convert an input sequence to an output sequence or an input label to an output sequence in an encoder/decoder framework. This, in turn does not allow any layer to abstract some upper hierarchical label in an unsupervised way.

3.4 M3LSTM RL

3.4.1 Contributions

The main contributions of this proposed architecture in this dissertation are in the area of topology of the hierarchical model and how it is trained, and at inference ability to elicit boundaries of salient upper or abstract states and corresponding subpolicies based on both the observable and hidden state and actions. The hierarchy or abstract features can be readily observed in some other networks like visual convolutional nets in the form of filter activations common across a diverse set of imagery, typically associated with edges and textures. In video, due to its three dimensions in terms of space plus time, spatio-temporal features are less commonly identifiable. In the control or policy domain, the dimensionality and diversity is even higher, partially due to diversity of sensors, actions and tasks, partially due to the actor constantly actively changing and traversing different parts of the world and hence changing observations, and partially due to the MDP or shallow POMDP nature or heuristics of elemental controllers, actors and critics both in time and in terms of flat policies without automatic subsumption. In the case of the deceptively simple taxi domain task, the problem is made more difficult in order to mimic the complexity of bigger domains by impoverishing the sensory input to detecting objects only when directly above those fields/objects. To accommodate this, the actor has to develop internal memory and the ability of counting, or, in other words, a hidden representation of the world without being given hints as to its proximity or intermediate rewards. It has to evolve a long term search strategy and also a long term hold and release strategy as it does not receive information indicating whether it is holding a packet or not. More tactical is obstacle avoidance, not repeatedly bumping into edges or performing oscillating actions. This goes for 1D and gets progressively more difficult for 2D and 3D versions, which when embellished with obstacles or other intermediate objects starts to exhibit the same complexities as solving multi-step strategic video games, proven to be very difficult for RL to learn. While some games (Atari) have been solved by DQN (DeepMind), those games needed shorter temporal focus and few multi-step strategies, often using only 4 consecutive frames when learning. In these applications it has been observed that even a small change in rendering may require re-learning or learning a new policy from scratch [86]. Some of the more complex games like Montezuma's revenge while graphically simple, require much longer term multi-stage policies and are still an open-ended RL benchmark problem. While the game is 2D the progression of the game due to barriers and corridors is often in 1D.

In the taxi domain world, which is heavily POMDP, one problem is that the search space is still large even for the 1D case as each step can have four actions and, without accounting for oscillations like forward/backward reversal in positions or repeated picking and dropping of the packet, the searching and reaching of the goal (with packet) is still exponentially high with an exponent of 4. In the 2D case it is at least to the power of 6 and in the 3D case to the power of 8. This is why even though the world looks simple from an oracle perspective, where the oracle has an extrinsic view (world map) and a priory context which actions are beneficial and what are the affordances of objects/locations, it is as comparatively complex as video gaming in the RL model free context due to the dramatic lack of observability. Another variation to the task would be to not pick up but rather push the packet in 2D. However, we did not consider this here.

The second problem this work is addressing is common to RL and related to the fact that the domain studied here has very sparse rewards, a highly limited view and movable objects which change their location in the world. The result of this that what makes this a challenging problem is not just that it is partially observable and provides very limited feedback but the world changes or reacts to actions. So the agent or rather the critic in the learning system has to build joint state of both the visible features and hidden state and of the world transitions with the agent acting in it.

The third problem that this work has to address to operate on this class of POMDP RL problems is to address not just the interplay and feedback loops between actor, critic and environment, but also the dynamics of LSTMs that are recursive themselves, and particularly of M3LSTM that has bottom up and top down connections.

Lastly when a larger number of LSTMs are interacting in different capacities such as actor, critic, predictor networks, the overall training dynamics can be quite challenging. The choice of hyper-parameters becomes an even bigger tunable matrix than with one network and the update dynamics between the different networks can cause significant issues and prevent convergence. In the architecture introduced here, the system will contain three different LSTM networks to form an actor, a critic, a world state predictor, and an intrinsic reward generator. To address the challenges of convergence, a number of techniques will be applied that might also be applicable to other architectures.

Most of the methods and techniques mentioned above and in Section 3.3.1 are complementary to each other and have been used in this work. Advantage learning alone does not require two networks, but double/dueling does require duplicate network weights. These networks are particularly important at the onset of training rather than during later stages because the size of the search space and the initial absence of guidance from the policy, as well as the fact that all the layers of interaction loops in the actor and critic are initially untrained and unaligned.

3.4.1.1 Bottom-up and Regularization of Information/Signal Flow

The main motive here is to have both a good mapping capability between input and output, as well as automatic extraction of meaningful upper states and conversely policies which may help in several ways. This requires i) that the upper context can influence lower policies and switch policies, ii) that once the upper, high level control policy has formed, this policy may be applicable for transfer learning, permitting it to be activated in new contexts from the higher level without the need to retrain the lower level action strategies, and iii) that ideally the system should move from being pure black box towards gray or white box, providing some level of explainability and human introspection into the controller behavior.

The importance of regulating the spatial flow and temporal history within the (horizontal) layer are that long span and long term dependencies can be learned automatically without resorting to special filters, numerous layers and heuristics, like with CNNs and TCNs. The horizontal flow over time is already part of native LSTM operation. For multidimensional spatial features, having multiple directions per time instance allows the flow of information in LSTM from all sides not just one. This is done via LSTM nodes arranged in a matrix formation, like in multidimensional multidirectional grid LSTM.

The importance of regulating the vertical connectivity between layers is that higher layers may evolve semantically compressed latent activations that may represent a more interpretable summary of the underlying activity. However, as such, progression of these higher level representations often occurs at a slower pace or cadence and often is better expressed in an event-driven rather than time-driven setting. Part of the vertical connectivity and in particular its gaiting is thus to permit arbitrary, potentially irregular time dilations between layers. This element of the network is not supervised even though it is part of the supervised input output network, and is not heuristically designed but learned, so it might be more generalizable across multiple tasks. This is opposite to CNNs (inception, highway) which have pre-designed filters and connectivity and activation regularization like sparsity, or to CRBM where activity is unsupervised but is highly distributed and very transient/spiky. The vertical regulation is accomplished via multiscale LSTM with special gate or indicator nodes and resembles a global/layer LSTM forget and output gate. The indicators are both indicative of observation and internal state transitions and also in control of global internal state transitions.

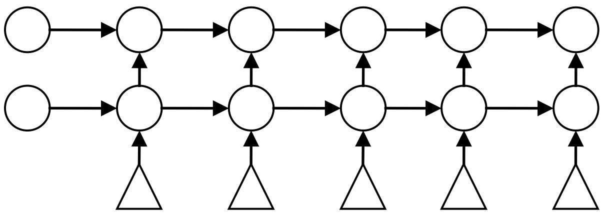

Figure 17 Traditional LSTM with input and previous state propagating each time step

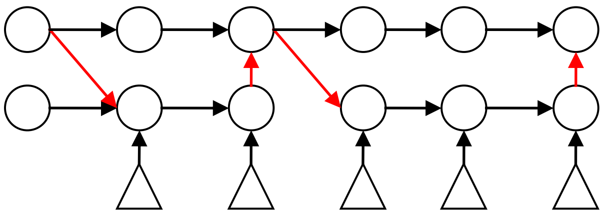

Figure 18 Gated propagation of signals over layers over time

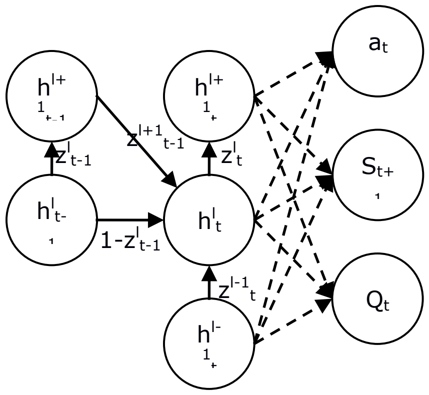

Figure 19 Detailed gating over nodes and sharing base network

Figure 17 shows the signal propagation with traditional layer LSTM connections depicted in black while Figure 18 shows the new M3LSTM connections in red. These are gated connections where a global layer gait/indicator is triggered based on information from activations of all adjacent layers, its own history, as well as the bottom and top layer. This ensures that here is a fuller context rather than just its own history of activations.

Figure 19 shows in more detail the flow of regular time (*t*-1) and bottom up (*l*-1) signals, but also the top down and gating signals *z.* The diagram also shows that the same base network can be shared among several outputs to compute controller action *a*, the critic value of the state (and action) *V*, and the prediction of the next states. Equations (6)-(10) detail how the different parts of the units in this LSTM-based network are calculated.

$$h_t^l, m_t^l, z_t^l = F^l(m_{t-1}^l, h_{t-1}^l, h_t^{l-1}, h_{t-1}^{l+1}, z_{t-1}^l, z_t^{l-1}) \qquad (6)$$

$$m_t^l = \begin{cases} f_t^l \circ m_{t-1}^l + i_t^l \circ g_t^l & if\ (z_{t-1}^l = 0\ \&\ z_t^{l-1} = 1) \quad update \\ m_{t-1}^l & if\ (z_{t-1}^l = 0\ \&\ z_t^{l-1} = 0) \quad copy \\ i_t^l \circ g_t^l & if\ (z_{t-1}^l = 1) \quad flush \end{cases} \tag{7}$$

$$h_t^l = \begin{cases} h_{t-1}^l & if\ (copy) \\ o_t^l \circ \tanh(m_t^l) & else \end{cases} \tag{8}$$

$$\begin{pmatrix} f_t^l \\ i_t^l \\ o_t^l \\ g_t^l \\ z_t^l \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ \tanh \\ hardsigm \end{pmatrix} F_{slice}\left(s_t^{recurrent(l)} + s_t^{topdown(l)} + s_t^{bottomup(l)} + b^{(l)}\right) \tag{9}$$

$$\begin{aligned} s_t^{recurrent(l)} &= U_l^l h_{t-1}^l \\ s_t^{topdown(l)} &= z_{t-1}^l U_{l+1}^l h_{t-1}^{l+1} \\ s_t^{bottomup(l)} &= z_t^{l-1} W_{l-1}^l h_t^{l-1} \end{aligned} \tag{10}$$

Here *h* denotes hidden, *m* the cell state, *z* the boundary detector, *f* the forget gate, *i* the input gate, *o* the output gate, and *g* the cell proposal vector. *W* and *U* are the weights and *b* is the bias, at layer *l* and time *t*. One difficulty with the top-to-bottom flow is that it breaks the conditional independence of nodes in layers, so learning considers the partial derivative in the gradient derivation. A second difficulty caused by the vertical flow is that treating the indicator *z* as a hard gate imposes some difficulty on the gradient of the discontinuous transfer function which is then approximated as a straight-through estimator as suggested by Hinton [93].

3.4.1.2 Decoupling of the Q Value

In the context of the actor-critic framework, it is important that the critic can estimate the value of the state and action, and that the actor can estimate the next best action. In model-based settings it is also important that the next states can be generatively predicted. The value of that state-action can be treated as the advantage function [82] and the value of the state, often only the advantage function is important. These values can be split either into two networks or split into two separate parallel layers towards the top of the network. This split is in order to decouple the value of the state from relative advantage of the actions, in order to avoid the problems with unintended coupling of the two described earlier. This is especially important in partially observable case where there is aliasing in external features, meaning that features may not change even though there has been change in position, just not observable, and the Q value may not be accurate. However, as long as the advantage value *A* is accurate, the policy is still correct. Here lower layers can be shared since there are some common elements of the world and agent (action) dynamics and of the world features that are compositional. Figure 20 shows the dueling network architecture to represent advantage and Q-value learning where the initial layers are shared but then the Advantage value (top branch) and the state value (bottom branch) are calculated separately in a way that would permit them to be recombined into the Q-values (green, virtual connections on the right).

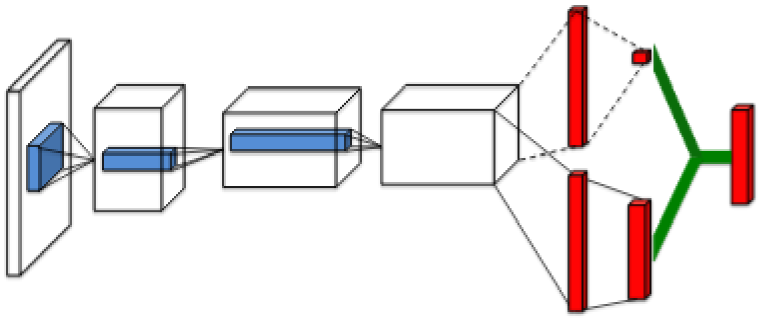

Figure 20 Dueling network for advantage learning

The importance of estimating the model of the world as a separate network (or separate higher output layer) is that an accurate long term model can help with replay and rollout into the future, by acting generatively. It can also help exploration by minimizing actual physical executions in the real world while still permitting value function and thus policy updates based on predicted outcomes. The more immediate use, however, is to facilitate an internal reinforcement signal based on novelty measured as an error in estimating the next state. In the initial exploration the effect is that the transitions that have not been explored are also less likely to have been modeled well, so it helps with guiding the search. This does not necessarily mean that the less understood parts of the world are more valuable than what the Q value is showing, but it may indicate unusual or interesting places that may not give immediate reward but may act like landmarks in the overall trajectory to the goal as described in Section 3.4.1.3.

Equation (11) show the formulation of the RL problem with the state value function V and the state/action utility Q while the decomposition of the Q value function into state value and advantage function is shown in Equation (12). In the approach presented here, this decomposition is used and represented with dueling LSTM networks.

$$\pi(s,a) = P(s \mid a)$$
$$R(s) = \sum_r P(r \mid s)$$
$$V(s) = \max_a (R(s) + \gamma \sum_{s'} P(s' \mid s,a) V(s'))$$
$$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) V(s')$$
$$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha(R(s) + \gamma \max_{a'}(Q^\pi(s,a') - Q^\pi(s,a)) \tag{11}$$
$$Q(s,a) = V(s) + A(s,a) \tag{12}$$

Here *V* is the average for the state and *A* the advantage of an action relative to other actions in the policy base on value iteration.

3.4.1.3 Intrinsic Rewards and Regularizers

Intrinsic rewards are becoming an increasingly important mechanism of exploration/exploitation and subgoal and options discovery, particularly in the presence of no or very sparse rewards. While intermediate rewards can be added heuristically or through inverse RL, many times determining these can become increasingly complicated and leads to unintended behavioral adaptations and poor performance. Since there are infinitely many partitions of subgoals and options possible, there is more than one regularizer needed, especially ones that do not just focus on rewards but information gain.

The commitment to a subgoal or an option/sub-policy in the proposed network is gated by the bottom-up gating mechanism described earlier. This is mostly unsupervised in a sense that there is no external signal governing the end of the option, but is still indirectly part of supervised learning that optimizes the input output mapping and output loss.

The predictivness of an option in terms of either the outcome state or a difference in factored state/observation between beginning and end of an option $\Delta s = s^{end} - s^{start}$, is an extension to one step predictor. This can be split between mean delta and variance in factored observations and goes beyond the one step predictivness used in basic intrinsic reinforce. In essence this gives the option nature similar to the one of elemental actions in terms of predictability of the effects of each action given previous start state and parameters. For options to be valuable they need to be predictable and parametric.

The distinctiveness of the option is another important attribute for good learning results and the potential for future transfer. In other words for an option to be valuable for transfer and avoid multiplicity of instances but rather have a flexible parameterization, it is important to increase the difference in terms of the effects that different options achieve. The aspect of divergence (KL) regarding the trajectory of states and relative difference between start and end state are of importance here rather than the actions that were taken. This also has an impact on interpretability of macro-actions.

Both the predictability and distinctiveness have to do with the notion of affordances, or what objects afford what subgoals, or rather what contexts afford actions and policies to succeed. The affordance is associated with the perceptual signature of a starting point. In other words, an option start point has to be recognizable and the end point also has to be such that it can be associated with certain subgoals.

In the current implementation, the options and subgoals are not explicitly identified as objects or landmarks, but rather elicited as transitions in the activity/state of higher layer gating nodes commonly referred to as indicators. They both mark the boundaries of states and boundaries of policy changes which are commonly interpreted as subpolicies.

3.4.1.4 Compositional Q Values

In the context of the critic and options it is important that both extrinsic and intrinsic rewards can be combined and also that the cumulative reward can be decomposed. There are a number of reasons. In the cases where rewards are extremely rare, the percolation of the utility value would diminish less over the sequence of options than over the sequence of elemental actions and thus not correctly capture the values. If it is decomposed, the utility value can be reconstructed from individual option contributions and be less prone to network mapping issues and exploration issues. To achieve similar increased robustness and usability, the rewards towards the subgoal or end of an option can be decoupled from the main goal. Here estimating the model as opposed to just direct reward also helps since short term rollouts can simulate the states and hence future rewards. The episodic reinforcement can be used without having to reconstruct the global value function. Equation (13) illustrates this decomposition

$$
\begin{aligned}
V^s &= \sum_{l=1}^{L} V_l^s \\
V_l^s &= \sum_{o(start)>s}^{O} V_{l-1,o}^s + \sum_{t>s,t<o(end)}^{S} r\gamma^t
\end{aligned}
\tag{13}
$$

where *s* denotes the current state at level *l*, *L* is the number of layers, *O* is the number of options at level *l*, *r* is the reward, and γ is the discount factor.

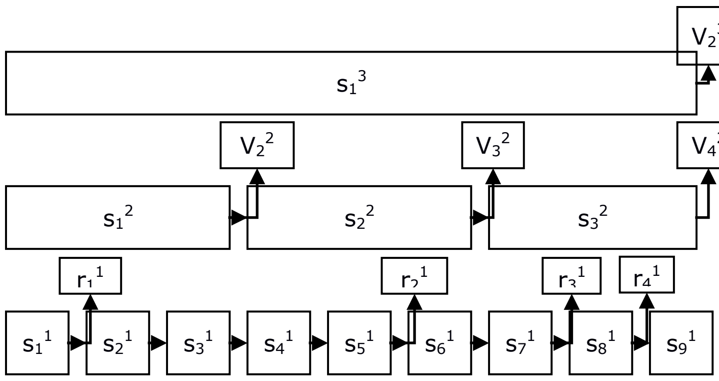

Figure 21 Compositional reward and utility/Q signals

In the current implementation, since the critic is a recursive layered deep network, parts of the value function are not explicitly defined as separate models constructed to map the additive nature of the utility or sum rewards, but are rather an input output mapping of the critic to reconstruct the overall *Q* value. Figure 21 shows the compositional nature of the rewards and value functions across different levels of abstraction, represented by different layers of the network.

3.4.2 Architecture

In Section 3.4.1.1 the implication is that layer gating via boundary detectors directly connects to state, action, and rewards nodes. The actual implementation is a little more subtle in that it works in the actor-critic context. There is the critic that estimates the value of the state in a compositional way and there is the actor that estimates the advantage function. However, to permit the overall capabilities, there is also a state-action and reward predictor that estimates the model and can be used for rollouts. What Section 3.4.1.1 implies is that lower level states that compress the space may be shared among different, more specialized layers atop or even between different sub-networks that specialize in *s*,*a*->*s*, *A*(*s*), or *r*(*s*).

There could also be other sub-networks that specialize in subgoal and option regularization based on information gain, state transition fan-in fan-out, predictiveness of options and rewards, or distinctiveness of options.

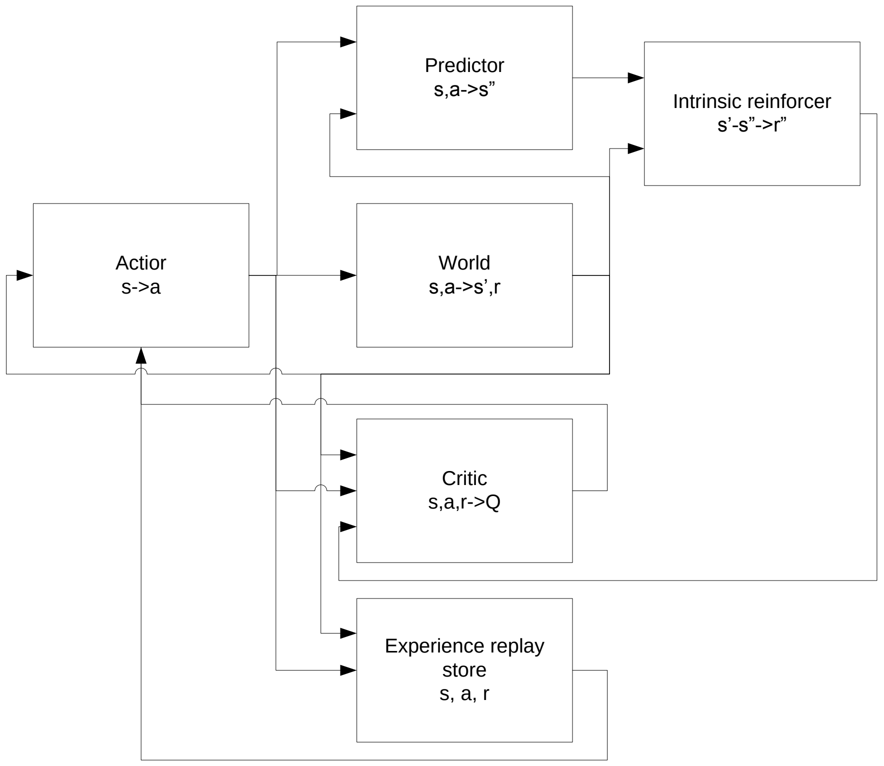

Figure 22 Actor, critic, predictor and intrinsic reinforcer architecture

Figure 22 shows the models and modules of the M3LSTM architecture proposed here, where *s* denotes state, *a* action, *r* reward, and *Q* the state/action value function. The internal subgoal *g*, option *o*, goal *G*, options *O*, as well as network layers and time are omitted here for clarity. There are two expectation maximization optimization loops. One between actor and critic in a sense that the critic estimates the current performance of the actor, and the actor estimates the actions based on the critic's values of states and actions. The second loop is between actor-critic and state-action predictor (model of the world), in a sense that the mapping of the state-action predictor will change based on where the actor has traversed, but also that the actor will be encouraged to explore based on performance of the state-action predictor. As shown in Figure 19, the networks don't have to be separate but can share the inputs and the base model and then differentiate the outputs. This can be beneficial when trying to extract the deeper hidden abstract state jointly, but can also be problematic when actor-critic and predictor look at different transition complexities. In other words, the observable world may be complex to fully predict in terms of all features but easy to control since many features are irrelevant. Conversely, the features may be simple, distinct, very rare, but hard to control, requiring memory, similar to taxi domain. Here the packet is not observable once on the cart so has to be memorized.

In the current implementation the critic including both the value and advantage Q values, and the actor share the same base network. The predictor shares the same network topology but has twice the inputs (*s* and *a*) and is separate. The resulting network architecture that was used for the taxi domain experiments presented in Section 3.4.5 is shown in Figure 23 and for this domain has 4 input units (8 for the prediction network), 2 layers with 5 hidden LSTM nodes each for each network, 2 gating indicators, one per layer, and 4 output units for each of the networks.

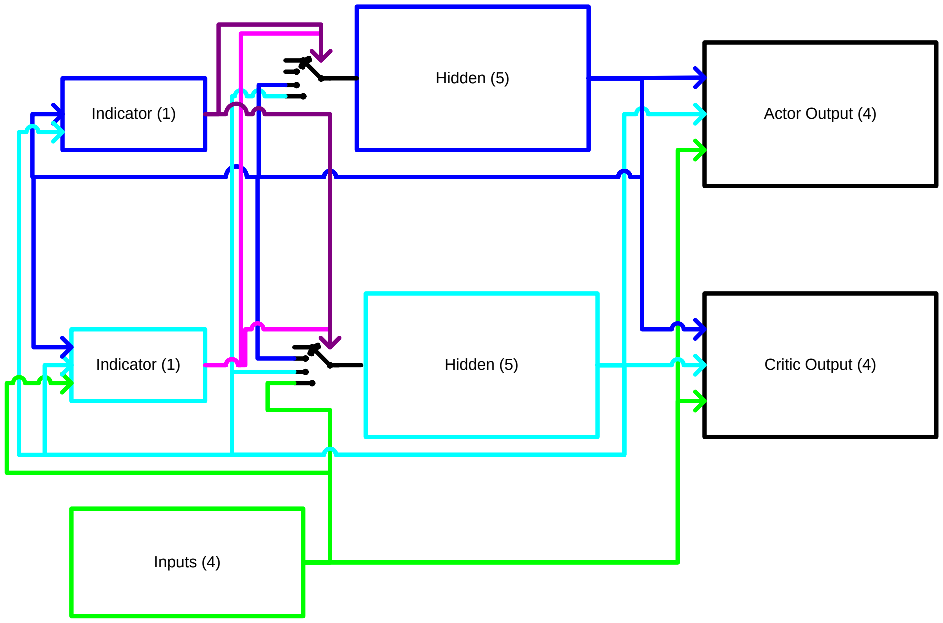

Figure 23 Topology of the actor critic network,

predictor is similar with larger number of inputs

3.4.3 Training

Training consists of initialization phase 1-3) where all three models topology, weights, optimizer are initialized and respective learning hyper parameters, and the training phase. Training phase consist of an outer loop 3-18) that iterates over trails and an inner loop 5-12) consisting of actor actions and world updates. The critic gets the observation s and outputs the both the value and advantage values, actor based on same observation outputs action activations, and predictor based on observations and action, the estimated new observation 10). On policy max actions are replaced with experience replay store actions based on epsilon randomness that decreases over the course of outer iterations. Upon exceeding the number of steps per trail or reaching the goal 12), the Q values are calculated in reverse with penalty and discount factor. The is used as a basis for an error signal for the critic, maxarg advantage as basis for error for actor, and difference in predicted vs. actual next observation as error for predictor. The mean error signals are fed into BP optimizer and deltas applied to weights. Here the double network is used and target network weights updated periodically including the deltas from other agents (not listed). The temperature of the noise for exploration is decreased over outer iterations.

Algorithm 1 High level pseudo code for training

```
 1)  Initialize networks, actor, critic, predictor, replay store, Adam optimizers
 2)  Initialize learning rate 0.001, gamma 0.95, penalty -0.05, max num of steps, experience
     replay max size and mixing ratio
 3)  For trail in trails
 4)     Initialize world to length of track, random positions of cart and packet and base with
        base and cart towards the beginning of the track, initial state s0
 5)     For steps until goal or max steps
 6)        s = get observable features of external part of factored state
 7)        q = max(Q(s, [a])=V+A) with some noise added in [q]
 8)        a = argmax(A(s)) or a = good trajectories(s0, s)
 9)        s' = world(a)
10)        s", r = S(s, a) predictor
11)        r' = r + c * error(s – s')
12)        If goal, break, record trajectory (s0, [a])
13)     For reverse steps
```

```
14)       $q(s,a,t)^* = r(t) + \gamma * \max(q(s,a,t+1))$
15)       $error(q, t) = q(s,a,t)^* – Q(s,a,t)$
16)       $error(a, t) = a(t) – \text{argmax}(A(t))$
17)       $error(s, t) = s(t) – S(t)$
18)    Gradient descent Q, A, S, with errors above
19)    Adjust exploration noise, mixing of experience replay
```

3.4.4 Model Training Challenges and Inference Performance

The current implementation is centered around Actor, Critic, Predictor and Intrinsic rewards module of the architecture as shown in Figure 22. All share the same type of network, in particular the L3STM structure. In this case, two layers with peepholes with boundary gating are used. It takes input vector of binary values [0, 1] and predicts Q values, next action, and next likely state in a sequence. As mentioned, this only appears as an elemental problem when observed from an oracle view perspective, but, due to the minimal observability of the domain, requires spatial and long term temporal processing as well as hidden states, and affords experimentation with the complexity of rules of the world and the length of sequences and values of hyperparameters. In practical real-world problems if inputs are visual, there would be additional DNN embedding layer between pixels and inputs of the LSTM.

The multiscale implementation of the indicator nodes posed a number of challenges both in terms of implementation and tuning. There are several loopbacks, namely the inner loop of the LSTM memory unit, the recursive loop from the previous time step, the feedback loop between lower and upper layer, and the outer loops between actor, critic, and intrinsic reinforcer. The multidimensional and multidirectional nature of the layer also requires iterative propagation of signals in the diagonal fashion from all corners (4 if 2D, 8 in 3D) before summing the signals to be sent towards upper layer. This 2D network which was previously demonstrated in Section 3.3.2 does not explicitly lend itself to the simplified 1D taxi domain since the nature of the input does not conform to a grid configuration. As a result, for the result in the taxi domain, only a 1D LSTM network was used. In terms of tuning, there is a tradeoff between straight-forward input-output performance on one task with more traditional feed-forward networks vs. the segmentation and generalization performances of the network for multitask learning.

Fine tuning the subgoal regularization for the predictor, had some challenges in terms of how to quantify and utilize the predictability. Subgoals will be derived as activities of the layer wise gating mechanism described above. The additional regularization is to be accomplished via additional sub-networks such as a novelty (intrinsic reward) network measuring how well a state can be predicted, and other metrics like state and option transition statistics. The prediction is accomplished through the *s,a->s* modeling sub-network, and the estimation of the distribution over end states can be done via rollouts or a more complex $sl,_{o}^{start}$->$s_{l,o}^{end}$ sub-network. The distinctiveness is somewhat less tangible, likely implementable using mapping KL divergence of the factored delta between start and end states, and was not utilized.

Due to the nature of LSTM and RL, and of the interplay between inner and outer direct or indirect feedback loops, which is not easily parallelizable in any DL/DNN framework, the training and inference was rather slow. Multiple agents and experience replay was used but the aforementioned feedback loops are still at play.

Current implementation is on top of TensorFlow implementing all node and gate recursions. At the same time, due to recursive and top to bottom loops, this is not easy to implement or optimize in traditional DNN frameworks and not easily parallelizable on GPUs. One reason is that it does not follow supervised learning where input and output is known and fixed and can be batched; rather, there is interaction between actor and the environment, which is active/reactive and changes observations in every step and there is also stochasticity in both actions for exploration purposes and potential effects of actions. There is also a nuance in treating the gradients in the top-down direction where conditional independence does not hold and in computing and treating the gradients over a discrete function (*z*).

Currently training can be done in parallel with multiple near identical agents (8), where one agent per hyper-thread share gradient updates periodically, and where the search space is not exclusive to each agent but overlaps. At later stages where there is less randomness, the search is not as parallel as at the beginning of training where trajectories are much more random and less likely to repeat.

3.4.5 Experiments

The experiment is situated in the simplified grid world where taxi needs to find the packet, retrieve and deposit it at the base. The task is made difficult by having a random positioning of the packet and no observability as to the proximity of the packet or whether the packet is off or on the cart. As a result, the agent has to evolve hidden representations and a number of strategies. The first strategy would be one to search which involves committing to one direction until either packet or edge is encountered. Further strategies would involve turning around towards the base, a tactic to turn around at the edge, and a strategy to drop the packet only at the base. To be able to learn these and decide among them, a number of hidden state features also have to be learned, including memory of whether it picked the packet has to be formed in the hidden layer since the packet is only observable on the ground. To learn the strategies and the memory concepts poses a chicken and egg problem for the learner since the strategies depend on the system's ability to distinguish whether they are useful and the usefulness of attributes in the hidden representation depend on the actions to be used based on them. As a consequence, they have to be learned simultaneously in the architecture which should thus autonomously form representation and skill hierarchy.

Figure 24 shows an example situation in a taxi world of length10 and Table 2 shows the corresponding observation vectors. Important to note here is the sparse observability that makes most states' observations identical (no observation in those states). To solve the problem, the system has thus to build its own hidden state information to encode its location and whether the packet has already been picked up.

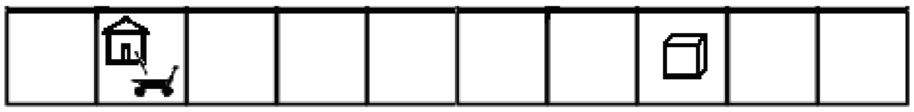

Figure 24 Graphical depiction of initial state of the world

Table 2 Initial world state (oracle view), 1-edge, 2-packet, 4-base, 8-other edge

Position	0	1	2	3	4	5	6	7	8	9
Initial world	1 (0000) edge	4 (0010) base cart	0 (0000)	0 (0000)	0 (0000)	0 (0000)	0 (0000)	2 (0100) packet	0 (0000)	8 (0001) edge

Table 3 State of the world over time, agent observations, automatically learned indicators, and actions over time. Features in 0000 depicted as edge, packet, base, edge, only features visible to agent. Actions 0-left, 1-right, 2-pick, 3-drop

(Position of cart)	1	2	3	4	5	6	7	7	6	5	4	3	2	1	1
Observation	0000	0000	0000	0000	0000	0000	0100	0000	0000	0000	0000	0000	0000	0010	0110
Action	1	1	1	1	1	1	2	0	0	0	0	0	0	3	1
Indicator	0.36	0.47	0.50	0.50	0.50	0.50	0.50	**0.86**	0.50	0.52	0.49	0.48	0.49	0.49	0.36
Indicator	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.47	0.55	0.54	0.52	0.51	0.51	0.50	0.52
(Has Pkg)	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0

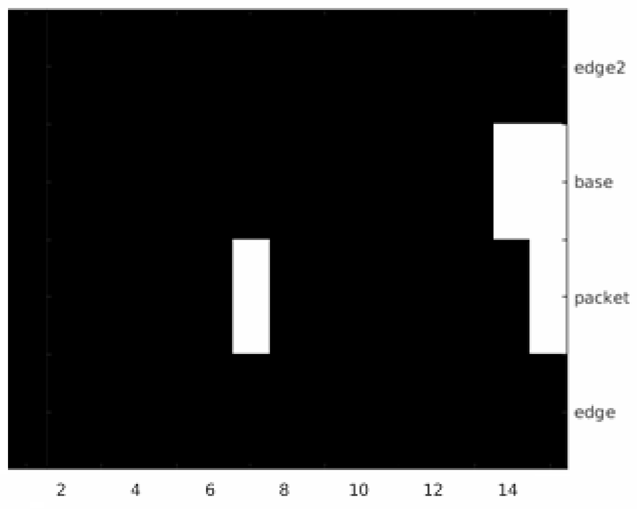

Figure 25 External features visible to the agent (0-edge, 1-packet, 2-base, 3-edge)

Table *3* shows the execution of the learned policy and Figure 25 External features visible to the agent (0-edge, 1-packet, 2-base, 3-edge) shows the corresponding observations. Here the action sequence shows that the system successfully executes and optimal policy, moving towards the package, picking it up, returning to the base, and immediately dropping it. The indicator variables in the table show the values of the indicators in the LSTM layers underlying the actor and critic networks. The spike in the top indicator at position 7 after it picked up the package shows that the agent is correctly able to identify and encode a subgoal and change the policy when the packet is encountered. Figure 26 shows the Q values predicted by the critic network for all of the actions during execution, while Figure 27 shows the state of the indicator variables during the execution in more detail, clearly displaying the spike in the first indicator. This illustrates that the system successfully learns to predict correct actions at the low level and subgoals and subpolicies at the high level.

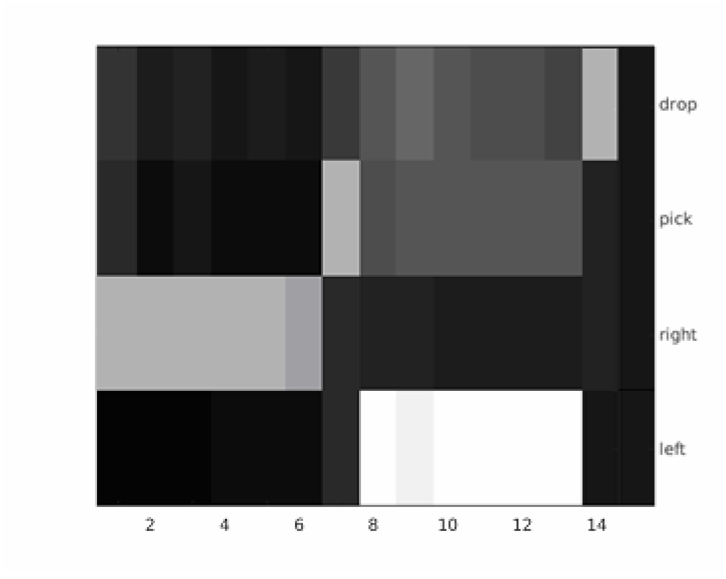

Figure 26 Q values (over steps) per each elemental action

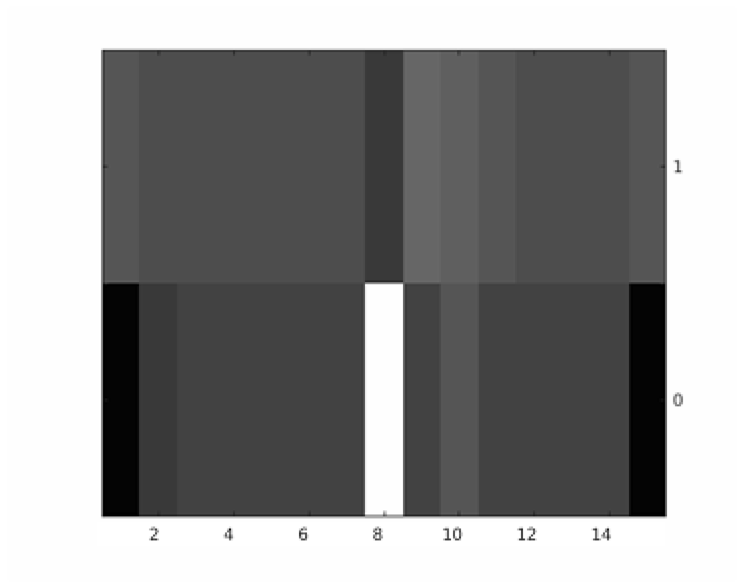

Figure 27 Internal agent indicators/states that signify change in subpolicy/option

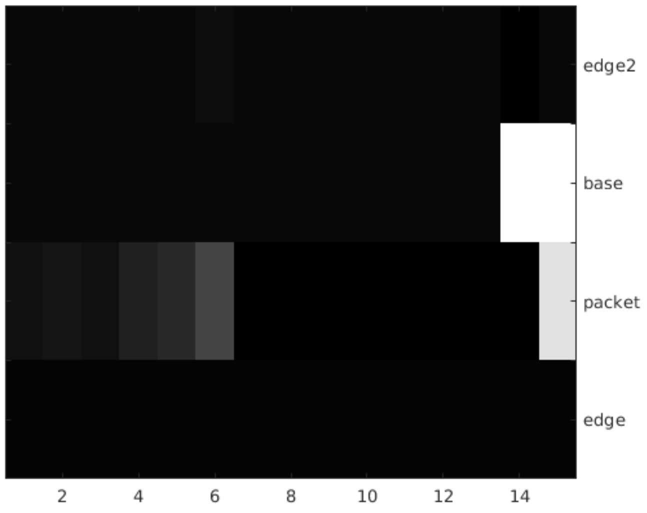

Figure 28 Predictor activations with packet expectation increasing over time

Figure 28 shows step ahead estimation of a predictor. The prediction of observing a packet after it is dropped is close to deterministic, but more interesting is the increasing expectation of observing a packet over time if one was not observed yet, which is plausible and coincides with both the Bayesian probability given fixed optimal policy and intuition that the longer the search is the higher the probability of finding the packet is.

It is important to note that in this experiment the gating (*z*) is not discrete 0 or 1 but is gradual ranging from 0-1. Since gating indirectly alters the behavior of the rest of the network significantly, as well as directly alters the lower and upper layers, discontinuous values tend to be harder to learn. Typically only when the network exhibits no errors in actions taken and there are only temporally prolonged actions, the indicator settles to a stable value. Figure 29 shows the situation when the indicator variable is discrete and illustrates the subpolicy state in more detail. This graph clearly shows the two discrete phases / subpolicies being executed prior and after the package is detected and picked up. However, during learning having a gradual indicator is advantageous because convergence is much faster/sooner. However, the disadvantage is that subpolicy/option cannot be easily identified as fixed discrete indicator values, but as a spike in the indicator, which induces other parts of network to change mode, then the trigger comes back to steady state.

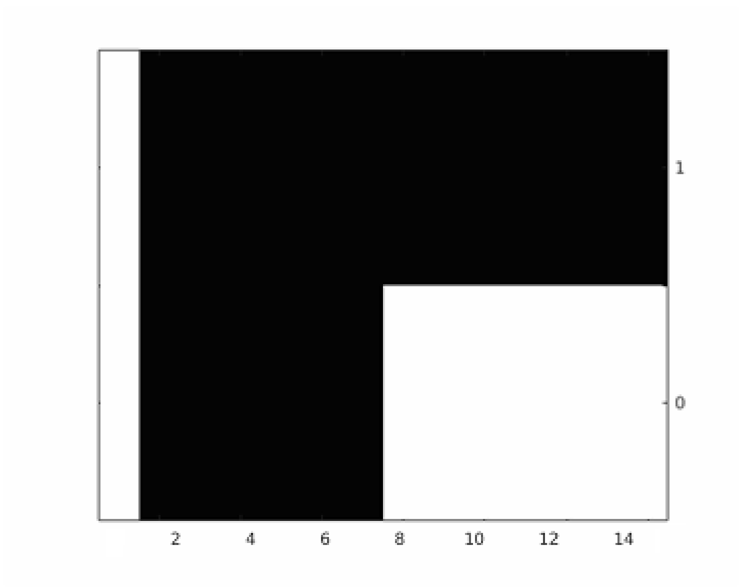

Figure 29 Internal indicators when gates are discrete functions

In contrast to the situation where the package is present, either edge, crossing over the base location, or being at same location when there is no packet, does not evoke any indicator activity. Figure 30, Table 4, Table 5 and Figure 31 show the corresponding situation. The agent is still able to turn at the edge but there is no attempt to drop at the base location, which indicates that there is an internal trigger to turn around and but it does not have the same benefit as when carrying a packet.

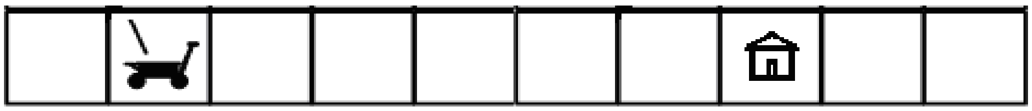

Figure 30 Graphical depiction of world state (oracle view) with no packet

Table 4Initial world state without packet oracle view), 1-edge, 4-base, 8-other edge

Position	0	1	2	3	4	5	6	7	8	9
Initial world	1 (0000)	0 (0000) cart	0 (0000)	0 (0000)	0 (0000)	0 (0000)	0 (0000)	4 (0100) base	0 (0000)	8 (0001)

Table 5 Change of world features, agent state, automatically learned indicators, and actions over time. Indicators not triggering on base of edge, drop not triggered, triggered turn at the edge

(Position)	1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1	0
Observation	0000	0000	0000	0000	0000	0000	0010	0000	0001	0000	0010	0000	0000	0000	0000	0000	0000	1000
Action	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1
Indicator	0.50	0.50	0.51	0.51	0.51	0.51	**0.51**	0.50	**0.51**	0.47	0.48	0.50	0.51	0.50	0.50	0.50	0.50	0.50
Indicator	0.51	0.51	0.51	0.51	0.50	0.50	0.50	0.51	0.51	0.57	0.55	0.53	0.55	0.55	0.56	0.56	0.57	0.57
(Has Pkg)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

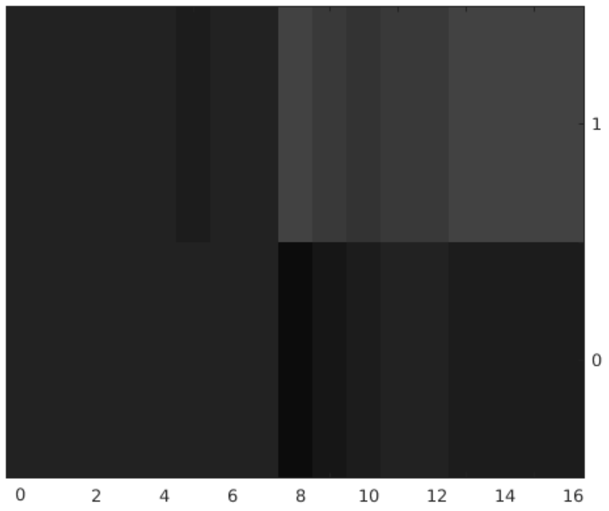

Figure 31 Internal agent indicators/states that signify change in subpolicy/option when there is no package, and agent turns around at the edge.

Due to the exploration/exploitation nature of reinforcement learning, the recursive nature of LSTM, and the closed loopback nature of interacting networks, among other learning strategies, curriculum learning had to be employed. This means that first shorter, simpler worlds need to be trained on before proceeding with larger and/or complex worlds. The initial trail in a smaller world (6 blocks) is depicted in Figure 34 which shows gradual decrease in length of trajectories relative to optimal length. The second Figure 35 shows when that learned model was transferred with longer track (up to 15 blocks). There is initially more noise at the beginning not just because the world is new but because the exploration has initially higher noise rather than later towards the end of simulation. The corresponding error in Q relative to best known Q* target value up to that iteration is depicted in Figure 32. This shows slow convergence to on policy Q value. Figure 33 shows the same learned Q value vs. the final optimal Q value which is not available to the agent.

Note that the iteration numbers in the figures are additive per all agents and typical training was done with 8 agents. In terms of convergence vs. number of cumulative iterations, the result is worse than for one agent since there is uncoordinated overlap in search trajectories, but in terms of wall time, the convergence is close to 8 times faster.

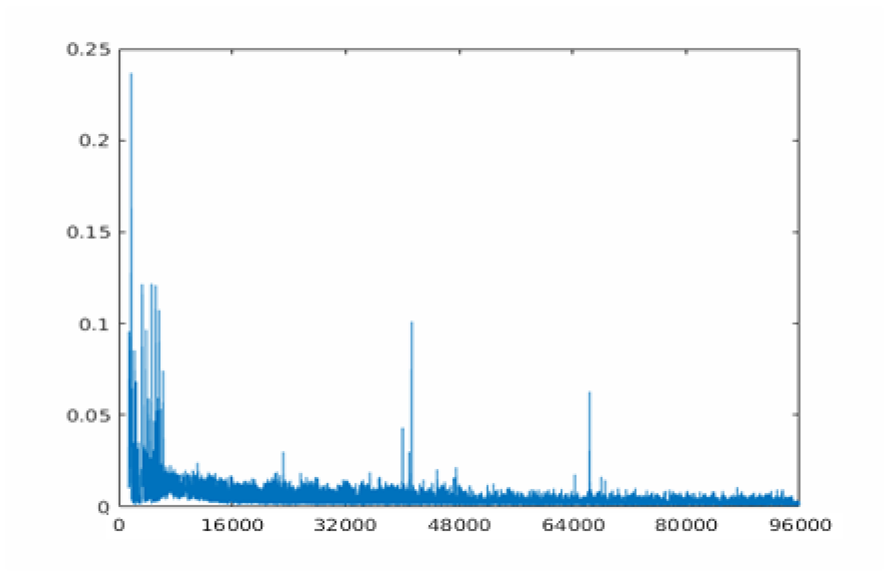

Figure 32 Critic Q error values between current optimal Q* and Q network, Bellman error, averaged over states/actions

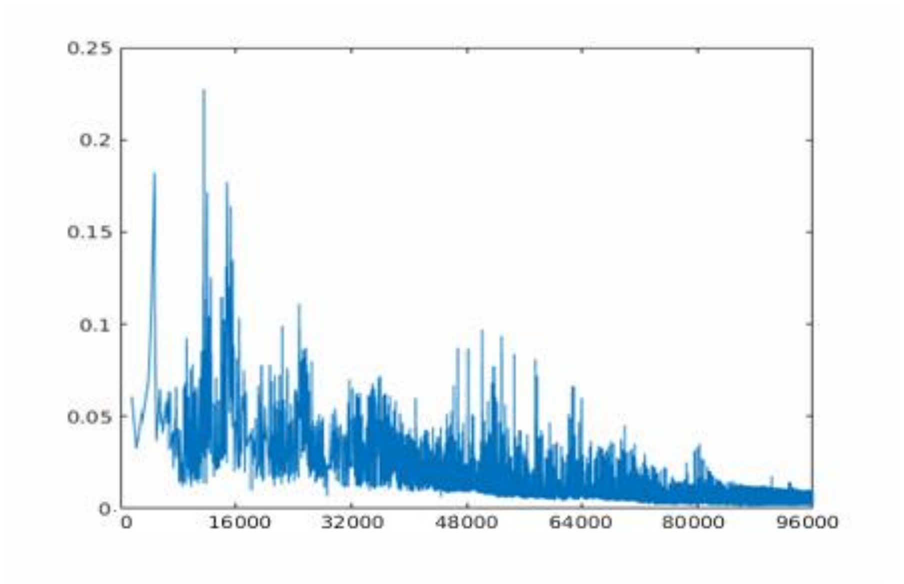

Figure 33 Critic Q error values between current Q** final ideal and Q network, averaged over states/actions

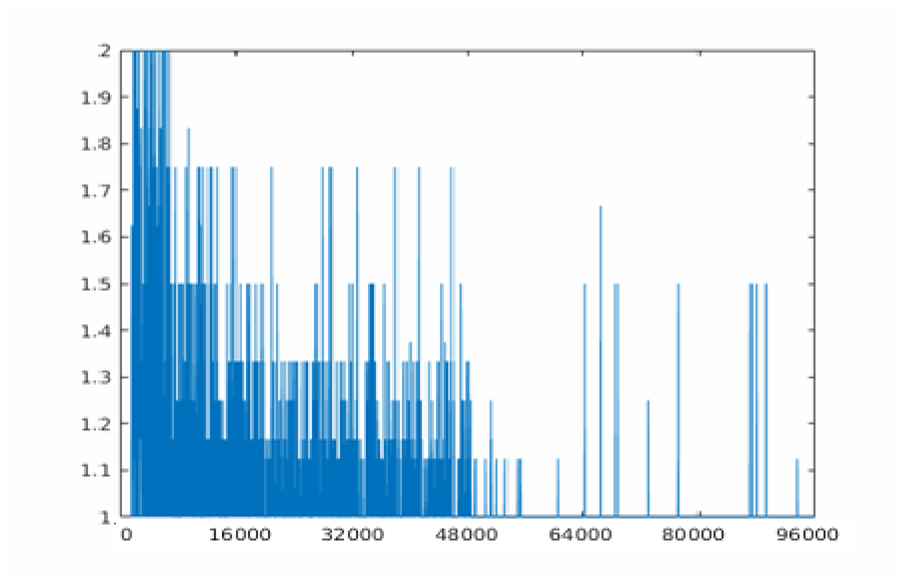

Figure 34 Sequence length ratio relative to optimal sequence length
(values over 2 clipped)

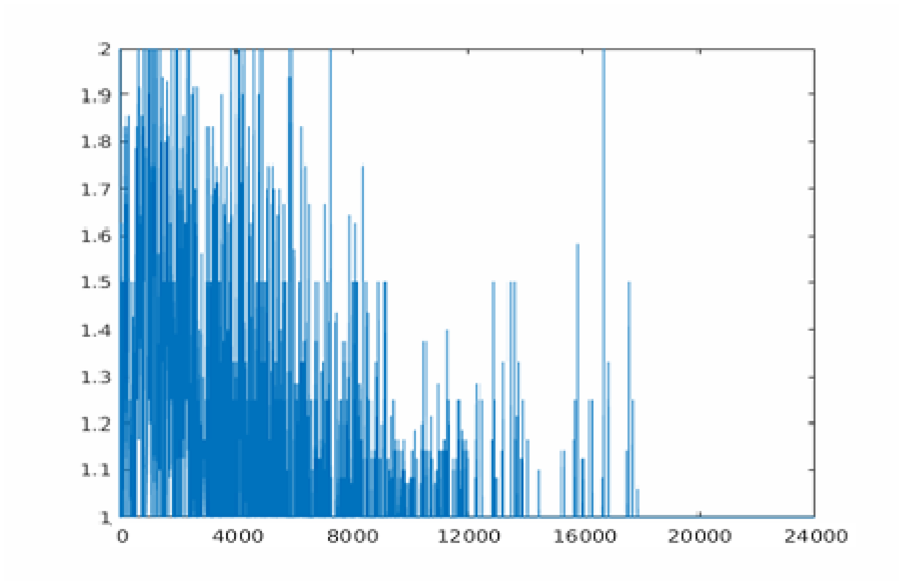

Figure 35 Sequence length ratios relative to optimal sequence length on a transferred domain
(longer bidirectional sequence)

4 Discussion

The simulation results show the capability of the network to exhibit the automatic latent, saliency abstraction behavior in a sense that the indicator node activation patters coincide with some pivotal points in policy behavior. This coincides to a large extent with what a human would identify as phases or options or subplicies in the overall solution strategy or policy, with the distinction that humans already have context or concepts of what/how each action performs and what the important map places and objects are. The agent has to evolve its own internal state space and state transition and utility representation without any priors and with very limited outside observations and feedback. It has been observed in gaming that when the common objects are given uncommon appearance or uncommon rules, humans also tend to perform worse in multi-step long term games.

The simulations also show that there is a certain amount of transfer learning happening. When going through curriculum leaning from simpler, smaller to more complex, larger domain, the system exhibits faster adaptation to the larger worlds. Due to the initial large search space it is more efficient to first learn an initially agnostic, basic agent and then to move on to more advanced skills. For example such development and transfer from simple to complex could be starting to train on a smaller world with both packet and house on one side of the initial agent position and agent being able to observe the packet it carries, and moving from there to later having random length, random positions of cart, packet, and base, and not seeing but having to remember holding the packet. Due to curriculum learning, there is not one continuous loss/error graph but several discontinues ones, which have higher error at the beginning of every new complexity level than on-policy would suggest due to high exploration at the beginning of training. The other aspect of transfer is that once the subgoal that delineates two subpolicies emerges, the actor-critic model can be used as base model for two separate actor-critics that can be from that point on be trained and specialized on their respective subspaces.

At the end, the agent is able to evolve a strategy that resembles temporally extended options or subpolicies that have some distinctiveness to what they react to and how long they last. Some parts of these policies are beneficial for new similar tasks, with the understanding that there is no single node that identifies with a single policy other than a discrete indicator node (which is hard to learn) or a gradual one which is triggered at transitions. The rest of the actor control or policy is still distributed among many node activations. This resonant behavior between layers also causes training to be longer and less stable relative to feed forward unidirectional supervised networks that are only trained with the input output mapping on a single task in mind. In our case the task is not just one world but multiple worlds as in combinations of positions of objects which are solved jointly while at the same time learning which objects afford success.

5 Conclusions

The motivation for this work was to investigate and prove the possibility of using two novel hierarchical recursive networks in the context of task decomposition, compositional plans, and transfer. The design impetus was the ability to solve extremely partially observable multi-step reinforcement learning tasks, and at the same time, without supervision or heuristics, elicit salient policy options and subgoals that can be helpful in generalizing and transferring to closely related tasks. Another goal was to provide introspection into the workings of the traditionally black box networks, an attribute increasingly important in explainability of the models. By using additional top-down connections, layer gating, and additional intrinsic reinforcement networks, the main network is able to capture the structure of the task by identifying pivotal subgoals and solve associated subpolicies/options. The higher structure of the task manifested by indicator changes exhibits plausibility from a human actor perspective and has proven transferable to closely related task, which was the initial premise of the work.

Future extensions to this model would entail handling 2D and 3D spaces with bigger observation aperture, trading larger local observability for much bigger search space and much longer step sequences with multiple subgoals. This would take advantage of multidimensional multidirectional LSTM with more layers for both actor, critic and predictor. The system can also utilize some of the regularizers such as the predictivness of longer term actions in terms of end effects or the overall delta between the beginning and end state, or the distinctiveness of policies to decide when to bifurcate the model into submodels corresponding to subpolicies/options. Once the indicators are formed, the model can be bifurcated into multiple more specialized models and continue training on subspaces.

M3LSTM provides the potential to capture long term and hidden dependencies particularly when combined with other specialized modules/models (intrinsic RL), but also requires careful hyper parameter selection and a more methodical training regime which, paired with parallel agents, can lead to containable, tractable training and models.

6 References

[1] Papudesi V., Huber M., Learning Behaviorally Grounded State Representations for Reinforcement Learning Agents, Epibot 2006.

[2] Asadi M, Papudesi V, Huber M, Learning Skill and Representation Hierarchies for Effective Knowledge Transfer, Proc. IJCAI-07, 2007.

[3] Goel S, Huber M, Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies, FLAIRS, 2003.

[4] Dean T., Givan R., Leach S., Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes, Proc. UAI-97, 1997

[5] Butz M., Swarup S., Goldberg D., Effective Online Detection of Task-Independent Landmarks, Tech. Report 2004002, 2004.

[6] Bakker B., Schmidhuber J., Hierarchical Reinforcement Learning, based Subgoal Discovery and Subpolicy Specialization, Tech.Report IDSIA, 2003.

[7] Bulitko V., Sturtevant N., Kazakevich M., Speeding Up Learning in Real-time search via Automatic State Abstraction, IJCAI 2005, 2005.

[8] Shestorov A., Stone A., Improving Action Selection in MDP's via Knowledge Transfer, Proc. AAAI-O5, 2005.

[9] Taylor M, Stone P, Behaviour Transfer of Value-Function-Based Reinforcement Learning, Proc. AAMAS'05, 2005.

[10] Jong N., Stone P., State Abstraction Discovery from Irrelevant Variables, Proc. IJCAI05, 2005.

[11] McGovern A., Barto A., Accelerating Reinforcement Learning through the Discovery of Useful Subgoals, Proc. 18th ICML, 2001.

[12] Simsek O, Barto A, Using Relative Novelty to Identify Useful Abstractions in Reinforcement Learning, Proc. 21th ICML, 2004.

[13] Hayes G, Konidaris G, Anticipatory Learning for Focusing Search in Learning Agents, Proc. ABiALS 2004.

[14] Konidaris G, Barto A, Building Portable Options: Skill Transfer in Reinforcement Learning, Tech.Report 2006-17.

[15] Konidaris G, Barto A, Autonomous Shaping : Learning to Predict reward form Novel States, 26th ICML, 2006.

[16] Konidaris G, Hayes G, Estimating Future Reward in Reinforcement Learning using Associateive Learning, Proc. 8th Intl.Conf. on the Simulation of Adaptive Behavior, 2004.

[17] Andre D, Russell S, State Abstraction for Programmable Reinforcement Learning Agents, Tech. Report UCB//CSD-02-1177, 2001.

[18] Drummond C, Composing Functions to speed up reinforcement Learning in a Changing World, Proc. 10th ECML, pp 370-381, 1998.

[19] Ravindan B, Barto A, Model Minimization in Hierarchical reinforcement Learning, Proc. SARA 2002.

[20] Mahadevan S, Spatiotemporal Abstractions of Stochastic Sequential Processes, Proc. SARA, 33-50, 2002.

[21] Rosenstein M, Cohen P, Continuous Categories for a Mobile Robot, Proc. 16th Nat. Conf. on AI, 1999.

[22] A.G. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning, DEDS, (13)4:341–379, 2003.

[23] Thrun S, Schwartz A, Finding structure in reinforcement learning, In Advances in Neural Information Processing Systems 7, 1995.

[24] McGovern, A., Barto, A. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. ICML 2001,pp.361-368.

[25] Andrew Kachites McCallum, Reinforcement Learning with Selective Perception and Hidden State, Ph.D. thesis, 1996.

[26] G. E. Hinton and R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, 313(5786):504-507, 2006.

[27] G. E. Hinton, S. Osindero, and Y. W. The, A fast learning algorithm for deep belief nets, Neural Comutation, 18:1527-1554, 2006.

[28] G. E. Hinton, Learning multiple layers of representation, Trends in Cognitive Sciences, 11(10):428-434, 2007.

[29] B, Sallans, G.Hinton, Reinforcement Learning with Factored States and Actions, Journal of Machine Learning Research, 5:1063-1088, 2004

[30] Geoffrey E. Hinton, Training Products of Experts by Minimizing Contrastive Divergence, Neural Computation, 14(8):1771-1800, 2002.

[31] Dietterich,T.G., Hierarchical reinforcement learning with the MAXQ value function decomposition, Journal of Artificial Intelligence Research, 13:227-303, 2000.

[32] Georgios Theocharous, Sridhar Mahadevan, & Leslie Pack Kaelbling, Spatial and Temporal Abstractions in POMDPs Applied to Robot Navigation, Technical Repot MIT-CSAIL-TR-2005-058, Computer Science and Artificial Intelligence Laboratory, MIT, 2005.

[33] Amy McGovern, Andrew G. Barto, Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density, in ICML, 361-368, 2001.

[34] Victoria Manfredi, Sridhar Mahadevan, Hierarchical Reinforcement Learning Using Graphical Models, Proceedings of the ICML05 Workshop on Rich Representations for Reinforcement Learning, 39–44, 2005.

[35] Richard S. Sutton, Doina Precup, Satinder Singh, Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning, Artificial Intelligence 112:181-211.

[36] Joelle Pineau, Sebastian Thrun, An integrated approach to hierarchy and abstraction in POMDPs, Technical report, CMU-RI-TR-02-21, 2002.

[37] Laurent Charlin & Pascal Poupart, Automated Hierarchy Discovery for Planning in Partially Observable Environments, Proc. Advances in Neural Information Processing Systems (NIPS), 2006.

[38] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems, 13(4):341–379, 2003.

[39] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition.J.Artif. Intell. Res.(JAIR), 13:227–303, 2000.

[40] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored mdps. Journal of Artificial Intelligence Research, pages 399–468, 2003.

[41] N. Hernandez-Gardiol and S. Mahadevan. Hierarchical memory-based reinforcement learning. In Advances in Neural Information Processing Systems, pages 1047–1053, 2001.

[42] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pages 1312–1320, 2015.

[43] J. Sorg and S. Singh. Linear options. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pages 31–38, Richland, SC, 2010.

[44] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1):181–211, 1999.

[45] C. Szepesvari, R. S. Sutton, J. Modayil, S. Bhatnagar, et al. Universal option models. In Advances in Neural Information Processing Systems, pages 990–998, 2014.

[46] Amy McGovern, Andrew G. Barto, Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. 2001 International Conference on Machine Learning.

[47] M. Frank, J. Leitner, M. Stollenga, A. Forster, and J. Schmidhuber. Curiosity driven reinforcement learning for motion planning on humanoids. Intrinsic motivations and open-ended development in animals, humans, and robots, page 245, 2015.

[48] S. Goel and M. Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In FLAIRS conference, pages 346–350, 2003.

[49] S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In Advances in Neural Information Processing Systems, pages 2116–2124, 2015.

[50] P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. Frontiers in neurorobotics, 1:6, 2009

[51] J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). Autonomous Mental Development, IEEE Transactions on, 2(3):230–247, 2010.

[52] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. Autonomous Mental Development, IEEE Transactions on, 2(2):70–82, 2010.

[53] S. P. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In Advances in neural information processing systems, pages 1281–1288, 2004.

[54] L. C. Cobo, C. L. Isbell, and A. L. Thomaz. Object focused q-learning for autonomous agents. In Proceedings of AAMAS, pages 1061–1068, 2013.

[55] C. Diuk, A. Cohen, and M. L. Littman. An object-oriented representation for efficient reinforcement learning. In Proceedings of the International Conference on Machine learning, pages 240–247, 2008.

[56] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational mdps. In Proceedings of International Joint conference on Artificial Intelligence, pages 1003–1010, 2003.

[57] Erik B. Sudderth, Describing Visual Scenes using Transformed Objects and Parts. International Journal of Computer Vision, special issue on learning for vision and vision for learning, 2007.

[58] Lea, C., Vidal, R., Reiter, A., & Hager, G. D., Temporal convolutional networks: A unified approach to action segmentation. In European Conference on Computer Vision(pp. 47–54), 2016.

[59] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:1602.01783, 2016.

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[61] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

[62] Alex Graves, Greg Wayne, Ivo Danihelka, Neural Turing Machines, 2014.

[63] Djurdjevic P., Huber M., Deep Belief Network for Modeling Hierarchical Reinforcement Learning Policies, Int. Conf. on System Man and Cybernetics '13, London, 2013,

[64] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. Neural Computation, 9(8):1735–1780, 1997.

[65] Alex Graves, Santiago Fernández, Jürgen Schmidhuber, Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition, ICANN 2005.

[66] Alex Graves, Santiago Fernandez, Juergen Schmidhuber, Multi-Dimensional Recurrent Neural Networks, 2007.

[67] Alex Graves, Supervised Sequence Labeling with Recurrent Neural Networks, preprint, 2009.

[68] Wonmin Byeon, Thomas M. Breuel, Federico Raue, Marcus Liwicki, Scene Labeling with LSTM Recurrent Neural Networks. 2015.

[69] Zhen Zuo, Bing Shuai, Gang Wang, Xiao Liu, Xingxing Wang, Bing Wang, Yushi Chen, Convolutional Recurrent Neural Networks: Learning Spatial Dependencies for Image Representation. 2015, CVPR2015.

[70] Zhen Zuo, Bing Shuai, Gang Wang, Xiao Liu, Xingxing Wang, Bing Wang, Learning Contextual Dependencies with Convolutional Hierarchical Recurrent Neural Networks, 2015.

[71] Nal Kalchbrenner, Ivo Danihelka, Alex Graves, Grid Long Short-Term Memory, arXiv:1507.01526, 2016

[72] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio, Gated Feedback Recurrent Neural Networks, 2016.

[73] Junyoung Chung, Sungjin Ahn, Yoshua Bengio, Hierarchical Multiscale Recurrent Neural Networks, 2016.

[74] LSTM: A Search Space Odyssey, Klaus Greff, Rupesh Kumar Srivastava, Jan Koutnik, Bas R. Steunebrink , Jurgen Schmidhuber, arXiv:1503.04069, 2017.

[75] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing Atari with Deep Reinforcement Learning, arXiv:1312.5602, 2013.

[76] Graham W. Taylor, Geoffrey E. Hinton, Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style, ICML 2009.

[77] Graham W. Taylor, Geoffrey E. Hinton and Sam Roweis, Modeling Human Motion Using Binary Latent Variables, NIPS 2006.

[78] Colin Lea, Rene Vidal, Austin Reiter, Gregory D. Hager, Temporal Convolutional Networks: A Unified Approach to Action Segmentation, arXiv:1608.08242, 2016.

[79] Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J.. A clockwork RNN. In Proceedings of the 31th international conference on machine learning, vol. 32 (pp. 1845–1853), 2014.

[80] Sara Sabour, Nicholas Frosst, Geoffrey E. Hinton, Dynamic Routing Between Capsules. 2017.

[81] Hado van Hasselt, Double Q-learning, NIPS 2010.

[82] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas, Dueling Network Architectures for Deep Reinforcement Learning, 2005.

[83] Alex Irpan, Deep Reinforcement Learning Doesn't Work Yet. https://www.alexirpan.com/2018/02/14/rl-hard.html

[84] Satinder Singh, Andrew G. Barto, Nuttapong Chentanez, Intrinsically Motivated Reinforcement Learning, NIPS, 2004.

[85] Yuri Burda, Harrison Edwards, Amos Storkey, Oleg Klimov, Exploration by Random Network Distillation, arXiv:1810.12894, 2018.

[86] On solving Montezuma's revenge, https://medium.com/@awjuliani/on-solving-montezumas-revenge-2146d83f0bc3

[87] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver, Prioritized Experience Replay, arXiv:1511.05952, 2015.

[88] Moore, A., Atkeson, C.. Prioritized sweeping: Reinforcement learning with less data and less realtime, Machine Learning, 13:103–130, 1993.

[89] Bengio, Y., Louradour, J., Collobert, R., and Weston, J., Curriculum learning. In Proceedings of the 26th annual international conference on machine learning, pp. 41–48.ACM, 2009.

[90] Joao Carreira, Andrew Zisserman, Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset, arXiv:1705.07750, 2017.

[91] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. The MIT press, Cambridge MA, 2018.

[92] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, Asynchronous Methods for Deep Reinforcement Learning, arXiv:1602.01783, 2016.

[93] Hinton et al. Coursera Lecture 9c.