Towards Practical Robot Manipulation using Relational Reinforcement Learning

Richard Li and Allan Jabri and Trevor Darrell and Pulkit Agrawal

Fig. 1: We present a simple yet effective reinforcement learning system that can stack 6 blocks in a tower without requiring any demonstrations or task-specific engineering. Our method also exhibits zero-shot generalization and is capable of configuring blocks into previously unseen configurations of multiple towers and pyramids without any training (last two rows). See the videos here: https://richardrl.github.io/relational-rl

Abstract—Learning robotic manipulation tasks using reinforcement learning with sparse rewards is currently impractical due to the outrageous data requirements. Many practical tasks require manipulation of multiple objects and the complexity of such tasks increases with the number of objects. Learning from a curriculum of increasingly complex tasks appears to be a natural solution, but unfortunately, only provides a small performance gain. We hypothesize that the inability of the state-of-the-art algorithms to learn from a curriculum stems from the absence of inductive biases for transferring knowledge from simpler to complex tasks. We show that graph-based relational architectures overcome this limitation and enable learning of complex tasks when provided with a simple curriculum of tasks with increasing numbers of objects. We demonstrate the utility of our framework on a simulated block-stacking task. Starting from scratch, our agent learns to stack six blocks into a tower. Despite using sparse rewards, our method is orders of magnitude more data-efficient and superior in performance to the existing state-of-the-art methods relying on either human demonstrations or task-specific engineering. Furthermore, the learned policy exhibits zero-shot generalization, successfully stacking blocks into taller towers and into previously unseen configurations such as pyramids, without any further training.

I. INTRODUCTION

The main idea in reinforcement learning is to incentivize actions that maximize rewards. Unlike video games, where rewards are readily available, for manipulation tasks, a reward function must be manually constructed. One such construction is to reward the agent only when it completes the job (i.e., sparse rewards). Sparse rewards are somewhat straightforward to define, but, because many tasks require execution of a long sequence of actions, sparse rewards drastically complicate the challenges of exploration and credit-assignment. Training with sparse rewards, therefore, either completely fails or requires massive amounts of data.

An alternative is to reward the agent for completing intermediate sub-goals (i.e., reward-shaping or dense rewards). However, in this scenario, an agent can get stuck in a local minimum of only ever achieving a few sub-goals and never completing the overall task. It is well known in the literature that intuitively reasonable reward functions can often result in unexpected or bad behaviors [1]. In general, the reward function and the learned behavior share a complex relationship which makes reward design very challenging.

Despite these challenges, reinforcement learning (RL) has been used for solving a wide variety of robotic manipulation tasks such as stacking blocks in a tower [2], [3], [4], opening a door [5], flipping a pancake [6], hitting a ball, orienting a cube [7] and other dexterous manipulation tasks [8]. However, to overcome challenges discussed above, these systems relied on either, (a) human demonstrations; (b) sim2real transfer; (c) careful environmental instrumentation to simplify the task or, (d) careful reward design.

Many works have sought to overcome these problems using: better optimization methods [9], [10], [11], combining model-based and model-free learning [12], hierarchical learning [13], design of better exploration methods [14], [15],
A few recent works have also exploited the compositional structure of the task and environment to improve the data efficiency of RL algorithms [19], [20], [21]. In the related field of supervised deep learning, transfer of knowledge by pre-training on a source task followed by finetuning on a target task [22], [23], [24] has been very successful in reducing the data requirements. However, in the context of RL, learning from multiple tasks and transferring this knowledge to reduce data requirements for a new task remains an open challenge [25], [26], [27], [28].

In this paper, we take a complementary approach of learning from a task curriculum. We propose a reinforcement learning system that can perform long horizon, multi-object manipulation tasks using only sparse rewards. Our system does not require human demonstrations or task-specific reward design.

Learning of a complex task can be simplified by first learning simpler tasks that in turn provide information about solving the complex task. This idea, known as curricular curriculum learn has

An intuitive solution is to present the agent with a task curriculum, where the agent stacks an increasing number of blocks starting from just a single block. However, learning from a curriculum exposes the inability of current deep networks to tackle shifts in the input data distribution. In the running example of block stacking, after an agent has mastered the stacking of two blocks, the introduction of the third block in the environment significantly changes the distribution of the agent’s input. In the absence of appropriate inductive biases, the agent is unable to cope up with such drastic changes in the input data distribution and resorts to treating the new data distribution as a new learning problem and starts learning from scratch. Therefore the agent is unable to use the knowledge of stacking two blocks to stack three blocks.

One well-known method to tackle changing data distribution is to train with data-augmentation, which has translated into the idea of domain randomization in the RL settings [29]. In the running example, training with randomization would involve sampling the number of blocks from a uniform distribution in every episode. However, because, in most episodes, the agent would be tasked to stack multiple blocks, learning in such a setup remains very challenging. This discussion suggests that the major hindrance in learning from a curriculum may not be in the design of the curriculum, but the inability of learning systems to transfer knowledge across the different tasks in the curriculum.

In this work we show that training a policy represented by a attention-based graph neural network using a straightforward curriculum is sufficient to learn to stack six or more blocks from scratch. The curriculum strategy is to increase the number of blocks presented to an agent after it masters the target task with a fewer number of blocks. This strategy can be applied to many multi-object manipulation tasks. To the best of our knowledge, ours is the first work to solve this problem without requiring any task-specific reward design, special instrumentation of the environment, or expert demonstrations. In fact, our system is orders of magnitude more efficient at stacking blocks as compared to previous work that used demonstrations [4]. Furthermore, our system can build towers that are taller than training time and also configure blocks in different configurations such as pyramids without any additional training (i.e. achieving zero-shot generalization). While we present results on the task of stacking blocks in different configurations, the approaches developed in this work do not make any task-specific assumptions and are therefore generally applicable to a wide range of tasks involving manipulation of multiple objects.

II. RELATED WORK

Our work is broadly related to techniques for scaling reinforcement learning algorithms to more complex robotic manipulation settings, as well as the use of relational and curricular inductive biases in machine learning.

Relational Inductive Bias: The use of relational inductive biases has a long history in reinforcement learning [30], [31], [32], and more broadly in logic and machine learning [33]. Recently, there has been great interest in the use of Graph Neural Networks (GNNs) for representing graph data structures, which are especially suitable for object-oriented environments [34], [35], [36], [37], [38], [39]. In the context of RL, a key motivation for relational representation is to support generalization by exploiting higher-order (i.e. pairwise) features of entities that appear in the state of the environment. This is usually achieved by defining predicates for useful relations between entities [31]. GNNs provide a framework for implicitly learning this relational structure end-to-end. In the past, GNNs have been studied in context of learning and transferring policies for locomotion across agents with variable morphologies [20], [21].

Closest to our work is past research combining GNNs with policy learning for manipulation tasks. However these works either rely on tens or hundreds of thousands of expert demonstrations [40], [41] or exclusively whos results on video games[19]. Furthermore, while these works have considered GNNs to improve efficiency of solving a single task, we combine GNNs with learning from a curriculum of increasingly complex tasks to solve long-horizon and sparse reward manipulation problems that cannot be solved directly by current methods.

Curriculum Learning: Curriculum learning addresses the effect of data sampling strategies on learning, under the presumption that proper sampling of tasks can allow for more sample efficient learning and avoidance of local minima [42]. In particular, prior work has shown that ordering tasks by heuristic measures of difficulty can be effective [43], [44]. A line of work has studied automatic discovery of curricula based on learning progress [45], adversarial self-play [18], [46], or backtracking [47]. So far, these methods have not yielded curricula capable of automatically discovering tasks of the complexity we consider. In this paper, our contribution is not in proposing a new algorithm or heuristic for choosing the task curricula, but to demonstrate the usefulness of
For example, to stack blocks into a tower, previous relied on human demonstrations [4], [48], or heavy reward engineering [3], [49], and/or carefully designed curriculum [3], [2] of reaching, picking and placing blocks. Such a design of curriculum and reward functions is a hard problem with no principled solutions.

**Hierarchical Reinforcement Learning** Hierarchical reinforcement learning (HRL) aims to address the scaling and generalization problem in RL by decomposing problems into smaller subproblems. Examples of HRL frameworks include the “options” framework [13], feudal learning [50], [51] and the MaxQ framework [52]. A key unsolved challenge is joint end-to-end learning of multiple levels of control, while avoiding degenerate solutions that lack hierarchical abstraction. Most successful instantiations of hierarchical RL make use of domain knowledge to construct a hierarchy [53]. To our knowledge, no hierarchical reinforcement learning algorithms have demonstrated the ability to solve stacking tasks of the complexity we consider [54].

### III. Experimental Setup

Figure 1 shows our simulated robotic environment consisting of a 7-DoF Fetch robot arm equipped with a two-fingered parallel jaw gripper. The robot can manipulate blocks kept on a table. Each block is a 5 cm cube. All our experiments require the robot to manipulate 1-9 blocks. This environment is based on OpenAI’s FetchPickAndPlace [55] and is implemented in the MuJoCo physics engine [56]. The robot’s action space is 4D, consisting of relative 3D position of its end-effector and a scalar value representing the distance between two fingers in the gripper.

**Observations:** The agent receives the location of the gripper and features representing N blocks as input. The block features are denoted by \(X^f: x_1^f, x_2^f, ..., x_N^f\), where \(N \in [1, 9]\) and \(x_i^f\) is the feature representation of the \(i^{th}\) block. Each block is represented by a 15-D vector consisting of 3D position \((x_i^p)\), 3D orientation expressed as Euler angles, 3D position relative to the gripper, 3D cartesian velocity and 3D angular velocity. The goal is expressed as set of 3D block positions, \(X^g: x_1^g, x_2^g, ..., x_N^g\). The overall input to the agent is therefore \(\{X^f, X^g\}\). At the start of every episode, the initial block positions are randomly initialized on the table and the goal positions are sampled using a pre-determined distribution. The maximum length of every episode is 50 \(\times N\) steps, where \(N\) is the number of blocks.

**Reward:** We use a sparse step-wise reward function where the robot is only rewarded when it places the \(i^{th}\) block within a distance \(\delta\) from its desired goal location. The overall reward for placing \(N\) blocks is given by: \(\sum_{i=1}^{N} \mathbf{1}_{\|x_i^p - x_i^g\| > \delta}\). We noticed that with this reward function, the robot learns to hold the top two blocks in its gripper instead of placing them and moving its hand away. To encourage this behavior, we added an additional term in the reward function that encourages the robot to move its hand away from all the blocks. Specifically, the agent was provided with a negative reward when the hand was at a distance less than \(2\delta\) from all the blocks. The negative reward was only provided when all the blocks are stacked. The overall reward is therefore given by, \(r_t = -\sum_i \mathbf{1}_{\|x_i^p - x_i^g\| > \delta} + \mathbf{1}_{\text{grip away}}\), where \(\mathbf{1}_{\text{grip away}}\) is the \(+1\) term mentioned above. Following [55], we set \(\delta = 5\text{cm}\), the size of each block.

### IV. Preliminaries

#### A. Reinforcement Learning

A typical RL agent acts within an environment \(E\), modeled by a discrete-time Markov Decision Process (MDP) described by state space \(S\), action space \(A\), transition function \(T\), reward function \(r(s,a)\), and discounting factor \(\gamma\). The aim of the agent is to maximize the expected cumulative reward along states \(s_1:T\) caused by a sequence of actions \(a_1:T-1\), by learning a suitable policy \(\pi(s_t, a_t)\), i.e. \(\max_{\pi} \mathbb{E}_{\pi,s_t \sim \mathcal{T}} \left[ \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t) \right]\).

A relatively efficient class of policy search algorithms is off-policy reinforcement learning. Q-learning [57] is a well known choice for off-policy learning, wherein the aim is to model the Q-function, i.e. \(Q(s_t, a_t) = r(s_t, a_t) + \sum_{t=i+1}^{T} \gamma^{t-i-1} r(s_t, a_t)\). In principle, the optimal Q-function is found by solving the Bellman equation [58]. In practice, we approximate the Q-function with a function approximator (i.e. a neural network) parameterized by \(\theta\) by minimizing the Bellman error \(\mathcal{E}(\theta) = \frac{1}{2} ||Q_{\theta}(s_{t+1}, a_{t+1}) - (r_t + \gamma \max_{a_t} Q_{\theta}(s_t, a_t))||^2\).

#### B. Goal-Conditioned RL

While the above formulation is appropriate for a single goal, for solving multiple tasks, it is necessary to provide a task description as input [59], [23], [60]. Goal conditioned policies are expressed as \(a_t = \pi(s_t, s_g)\), where \(s_g\) represents the goal state. The learning problem is expressed as:

\[
\max \pi \sum_{n=1}^{N} \mathbb{E}_{s_g^n \sim \rho(s_g | n), a \sim \pi, s_T \sim \mathcal{T}} \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t, s_{g}^n) \tag{1}
\]

where \(n\) is the task id and goal \(s_g^n\) is sampled from a goal distribution \(\rho(s_g | n)\).

#### C. Graph Neural Networks (GNN)

The central computation in a GNN is message passing between 1-hop vertices of a graph performed by a graph-to-graph module. This module takes as input a variable-size vertex set \(v = \{v_i\}_{i=1}^{N_v}\) and outputs an updated set \(v' = \{v'_i\}_{i=1}^{N_v}\) where \(N_v\) is the number of vertices in the input graph. \(v_i, v'_i\) denote feature vectors of the \(i^{th}\) node before and after a round of message passing. In each message passing round, each vertex sends a message to every other vertex. In attention-based GNNs, the incoming messages are weighted by a scalar coefficient (computed by attention) according to their relevance to the receiving vertex. The new feature representation of the vertex is the weighted sum of incoming messages. Message passing is typically performed multiple times. After message passing, the entire graph is represented as a fixed-sized embedding by pooling features across all vertices.
Mathematically, let the feature representation of the $i^{th}$ vertex at time-step $t$ be $\vec{v}_i^t$. In every message passing round, each vertex generates a query $\vec{q}_i^t$, key $\vec{k}_i^t$ and a message $\vec{m}_i^t$ using independently-parameterized functions $\vec{q}_i^t = \phi_q^t(\vec{v}_i^t)$, $\vec{k}_i^t = \phi_k^t(\vec{v}_i^t)$, and $\vec{m}_i^t = \phi_m^t(\vec{v}_i^t)$. Each vertex in the graph receives a message from all the vertices and computes it’s feature representation, $\vec{v}_i^{t+1} = \sum w_{ij} \vec{m}_j^t$, where $w_{ij}$ are the attention weights and are computed as following: $w_{ij} = \text{softmax} \left( V^T \tanh(\vec{q}_i + \vec{k}_j) \right)$. We use a residual connection and layer normalization between the output of message passing round $t$ and the input of message passing round $t + 1$ to ease optimization.

### V. Method

We present a simple, but effective method for solving long-horizon sparse reward tasks using reinforcement learning. Our core contribution is to equip the RL agent with inductive biases of relational reasoning in order to enable learning from a curriculum of tasks of increasing complexity. We use Soft-Actor Critic (SAC; [10]) as our base learning algorithm because it more robust to choice of hyperparameters and random seeds as compared to alternative off-policy learners such as DDPG [61]. To use the same policy for multiple tasks, we modified SAC to be goal-conditioned [59], [23], [60]. For better sample efficiency, we also incorporated the idea of goal re-labelling via hindsight experience replay (HER; [60]). Details of SAC and HER can be found in the respective papers and are not directly relevant to our work. While we use SAC + HER for policy learning, our contributions are not specific to these algorithms and are applicable to any policy learning method.

We represent both the actor and critic in SAC using the graph neural network architecture described in Section IV-C. The various components of the GNN ($\phi_q^t, \phi_k^t, \phi_m^t$) use 64D linear layers. We use separate weights for each round of message passing and terminate the message passing after 3 rounds. We call this agent architecture as ReNN. We compare the performance of ReNN against the baseline system that constructs the actor and critic using four layers of 256D fully connected layers (referred to as MLP in rest of the paper).

#### Training Curriculum:

We trained the robot to stack multiple blocks using three different curricula of tasks:

- **Direct**: The robot was directly tasked to learn a policy to stack six blocks starting from scratch.
- **Uniform**: At every episode, the number of blocks was uniformly sampled between 1 and 6.
- **Sequential**: The robot must first pick and place a block at goal positions sampled 50% of the time on the table and 50% of the time in the air. The robot then must pick and place 2 blocks, where one block goal is sampled on the table and the second block is sampled the same as the previous stage. Thereafter, the robot is tasked with stacking blocks in a single tower configuration sitting atop the table. After the robot perfected stacking (N-1) blocks, it was given N blocks to stack. N was sequentially increased from 3 to 6. The transition points in this curriculum are manually chosen based on return and success rate curves.

### A. Testing Details

We evaluated the generalization capability of the policy trained for stacking a single tower by evaluating its performance on the following tests:

- **Single Tower**: A single point is uniformly sampled on the table to serve as the base of a block tower. The goal positions of the blocks corresponded to translation along the z-axis from the base.
- **Multiple Towers**: Few points ($k \in \{2, 3\}$) were sampled on the table to serve as the base location of multiple towers. Each block was randomly assigned to a tower to produce towers of approximately equal height.
- **Pyramid**: A uniformly sampled point on the table served as a corner point for pyramid configuration. Figure 5 shows different Multiple Towers and Pyramid goal configurations for varying number of blocks.

We report performance of ReNN_Sequential (referred to as ReNNin later text) across three seeds. For other methods we report performance on a single seed. Success rate is reported as accuracy of completing a task averaged over 100 episodes. A task is considered complete when each block is within its goal position in the final timestep of the episode rollout.
VI. RESULTS

TABLE I: Comparing the performance of our method against the previous state-of-the-art [4] that makes use of human demonstrations on the block stacking task. Each entry, $p\%$ ($\pm$ $s$), denotes that the method achieved accuracy of $p\%$ after $s$ number of environment steps. We report the mean and standard deviation over 3 seeds for our method. Our method is both superior in performance and orders of magnitude more efficient.

<table>
<thead>
<tr>
<th>Task</th>
<th>Single Tower 4</th>
<th>Single Tower 5</th>
<th>Single Tower 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nair'17 [4]</td>
<td>$91% (850M)$</td>
<td>$50% (1000M)$</td>
<td>$32% (2300M)$</td>
</tr>
<tr>
<td>Ours</td>
<td>$93% \pm 4% (23M)$</td>
<td>$84% \pm 6% (27M)$</td>
<td>$75% \pm 4% (30M)$</td>
</tr>
</tbody>
</table>

Figure 2 shows that $ReNN$ trained with the sequential curriculum (green line; section V) succeeds at stacking six blocks in the tower. Standard $MLP$ architectures or $ReNN$ trained to directly stack 6 blocks without the sequential curriculum fails. Our experiments revealed that training with $Uniform$ curriculum was also insufficient. These results show that both $ReNN$ and the sequential training are critical for success. To best of our knowledge, ours is the first paper to show that is is possible to train a RL agent to stack six or more blocks in a tower after starting from scratch, without the need of task-specific reward and environment design, nor expert demonstrations.

The training curve of our method in Figure 2 has several fluctuation in performance. Many of the significant dips in performance correspond to increments in the number of blocks that are presented to the agent for stacking as the sequential curriculum (see Section V) progresses. In most cases, dips in performance were corrected with little additional experience. The only notable exception is the performance dip at 9M steps that correponds to transitioning from 1 to 2 blocks. This was the first time the $ReNN$ were faced with multiple entities and thereby this was the hardest stage of the training that required significant experience in the new environment for recovery. It can also be seen that variance in performance increase with the number of blocks in the environment. We hypothesise that this is because stacking a larger number of blocks in a tower requires delicate balance that gets harder to achieve with more blocks. We also found that SAC converged faster, albeit with higher variance when it’s exploration was augmented to take a random action with probability of 0.1.

Our method achieves a success rate of 75% at stacking 6 blocks in 30 million timesteps. In comparison, the existing state-of-art method [4], that makes use of human demonstrations and resets, achieves only a success rate of 32% after over 2.3 billion timesteps (see Table I for a detailed comparison) to stack 6 blocks. While the base learning algorithm used by [4] is DDPG + HER, in comparison to SAC + HER used by us, the orders of magnitude difference in performance cannot be attributed to the choice of using SAC instead of DDPG. Figure 2 clearly shows that SAC + HER, without the sequential training and the $ReNN$ architecture proposed in this work, is inept at stacking blocks. Additionally, we found that it is very difficult to properly weight the behavioral cloning loss of demonstration-guided methods such as [4] with the entropy term in SAC, rendering SAC unusable as a replacement for the original DDPG used in [4].

A. Zero-shot Generalization

It is desirable to learn policies that are not only adept at the task they were trained on, but can be re-purposed for new and related tasks. If our $ReNN$ architecture indeed provides a good inductive bias, then it should be possible solve different block configuration tasks with high-accuracy. To test this, we evaluated the performance of the learned policy, without any fine-tuning on previously unseen block configurations (i.e. zero-shot generalization) described in Section V-A. The results of this analysis are summarized in Figure 3.

**Single Tower Evaluation:** First we evaluated whether a policy trained to stack $i$ blocks in a single tower could stack up to 8 blocks in a single tower without requiring any further fine-tuning. The bars in different shades of red color in Figure 3 illustrate this evaluation for six policies, each of them trained to stack a single tower containing $4$ to $6$ blocks respectively. We found that it is possible to stack towers with $i + 1$ blocks without any further training.

Despite the success on $i + 1$ blocks, our system rarely succeeded on stacking $i + 2$ or more blocks. One possible reason for this failure is that our system is unable deal with larger number of blocks from the perspective of large changes in statistics of the input distribution. The other possibility is that it becomes progressively harder to stabilize
larger number of blocks in a tower and the robot needs to substantially refine its strategy to stack more blocks. To tease apart the reason for failure, we tested zero-shot generalization on the task of stacking larger number of blocks, but in multiple towers of smaller heights.

**Multiple Towers Evaluation:** The second sub-figure in Figure 3 illustrates the performance of a system trained on \( i \) blocks in stacking up to 9 blocks into multiple towers. It is clear that significantly higher success rates are possible for multiple towers with higher numbers of blocks than the single tower task with higher numbers of blocks. These results suggest that our system can easily handle a larger number of blocks, but stacking a taller single tower without additional training is non-trivial due to the difficulty of stabilizing a taller stack of blocks. The performance improves slightly for Multiple Towers 7, 8 and 9 as the training task gets more blocks. A reason for why the improvement is not more significant is that the system may start overfitting to the single tower task as the number of blocks gets higher.

**Pyramid Evaluation:** To stress test our system further, we evaluated its performance on placing blocks in a pyramid configuration (see Figure 5). Note that the robot never saw pyramids during training. Stacking blocks in pyramid is different than a tower, because now blocks may need to be balanced on two supporting blocks instead of only being stacked vertically. Bar plots in the third sub-figure in Figure 3 show that our system is able to generalize and manipulate larger number of blocks than seen in training into substantially different pyramid configurations. Interestingly, the agent trained on Single Tower 4 performs better on the difficult Pyramid 5 and Pyramid 6 tasks than the agent trained on Single Tower 6. This is again likely due to overfitting to the single tower training task as the target pyramid task becomes more and more dissimilar as the number of blocks grows. To the best of our knowledge, we are unaware of any prior work in RL that has shown such zero-shot generalization results for manipulating blocks into various configuration of stacks. At the same time, there is substantial room for improvement and we have highlighted some exciting directions of future research in Section VII.

### B. Analyzing the learned representations

In order to gain some insights into why ReNN leads to faster convergence and better generalization, in Figure 4 we have visualized the pattern of attention as the robot stacked six blocks to form a tower. The first row of the figure shows the images corresponding to key steps in stacking of blocks. Each column in second row is a \( 6 \times 6 \) matrix \( (E) \). Each entry in the matrix, \( e_{ij} \) represents the normalized relevance score of the \( i^{th} \) block on the features of the \( j^{th} \) block (see equation at the bottom of Section IV-C) computed by the final layer. Figure 4 clearly shows that maximum attention is to paid to the block that is to be placed and the attention on existing blocks in the stack decreases from the top-most to bottom-most block. Such a pattern of attention suggests that our system has learned to focus on the blocks most-relevant to the sub-problem (i.e. stacking of individual blocks) the robot is solving. This is turn suggests that our system has learned to decompose a complex problem into simpler sub-problems. We hypothesize that such decomposition is the reason why our system is easily able to learn from our curriculum of tasks and perform zero-shot generalization.

### VII. Discussion

We have presented a framework for learning long-horizon, sparse reward tasks using deep reinforcement learning, relational graph architecture and curriculum learning. While we are orders of magnitude more sample efficient than the the existing state-of-the-art, our method would still require a few dozen robots (corresponding to our 35 workers) and several days (assuming each action takes .25 seconds) of real world training to achieve a comparable environment step complexity. And while block stacking is representative of long-horizon, multi-object manipulation tasks, it is important to scale our method to tasks involving more complicated object geometries and more granular manipulation. For example, we may consider a real world cooking task with tools and ingredients.

We present a simple curriculum based on the reasonable principle that smaller sets of objects are easier to learn to manipulate than larger sets of objects. However, more complicated and effective curricula could exist along axes of variation beyond just the object cardinality, and discovering these curricula automatically is an interesting direction for future research. One point of concern with relational architectures is that the computation time is quadratic in the number of entities. Developing compute efficient methods is therefore important to scale these methods to environments with much larger numbers of objects. Finally, while we have presented results from state observation, in the future we would like to scale our system to work from visual and other sensory observations.

### VIII. Acknowledgements

AWS, DARPA. Trevor Darrell’s group is supported in part by the US Department of Defense, and the BAIR and BDD industrial consortia.

### References


Fig. 4: Analysis of internal representation used by our system to solve the task of stacking six blocks into a single tower. The first row shows the key step of this task. The second row, shows for each of these key frames, a $6 \times 6$ matrix, whose $(i, j)^{th}$ entry represents the influence of feature representation of $j^{th}$ block on the $i^{th}$ block after three message passing rounds. The labelled index along the y-axis and x-axis correspond to the object ID. Object IDs 1 - 6 correspond to green, yellow, blue, pink, red, and black blocks in order. The attention map reveals that our agent pays attention to the sub-set of blocks relevant to solving the sub-problem (stacking one block) at hand. We speculate that such an attention map suggests that the agent has learnt to decompose the complex block-stacking task into simpler sub-problems.


APPENDIX

Fig. 5: Top: successful block configurations for Pyramid 4 to Pyramid 7. Bottom: successful block configurations for Multitower 4 to Multitower 9.

TABLE II: Experiment Hyperparameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of workers</td>
<td>35</td>
</tr>
<tr>
<td>Replay buffer max size</td>
<td>1E5</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>3E-4</td>
</tr>
<tr>
<td>Epsilon for uniform action sampling</td>
<td>.1</td>
</tr>
<tr>
<td>Discount factor</td>
<td>.98</td>
</tr>
<tr>
<td>Batch size</td>
<td>256</td>
</tr>
<tr>
<td>HER fraction of re-labelled goals</td>
<td>.8</td>
</tr>
<tr>
<td>SAC target entropy</td>
<td>4</td>
</tr>
</tbody>
</table>

TABLE III: ReNN Architecture Hyperparameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding dimension</td>
<td>64</td>
</tr>
<tr>
<td>Number of graph modules</td>
<td>3</td>
</tr>
<tr>
<td>Graph module weight sharing</td>
<td>False</td>
</tr>
<tr>
<td>Post-readout MLP hidden layers</td>
<td>3</td>
</tr>
<tr>
<td>Input normalization</td>
<td>Shared mean/stddev for each block</td>
</tr>
<tr>
<td>Activation function</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Activation function (attention)</td>
<td>tanh</td>
</tr>
</tbody>
</table>