Grid-to-Graph: Flexible Spatial Relational Inductive Biases for Reinforcement Learning

Zhengyao Jiang
Centre for Artificial Intelligence
University College London
zhengyao.jiang.19@ucl.ac.uk

Pasquale Minervini
Centre for Artificial Intelligence
University College London
p.minervini@cs.ucl.ac.uk

Minqi Jiang
Centre for Artificial Intelligence
University College London
msjiang@cs.ucl.ac.uk

Tim Rocktäschel
Centre for Artificial Intelligence
University College London
t.rocktaschel@cs.ucl.ac.uk

ABSTRACT

Although reinforcement learning has been successfully applied in many domains in recent years, we still lack agents that can systematically generalize. While relational inductive biases that fit a task can improve generalization of RL agents, these biases are commonly hard-coded directly in the agent's neural architecture. In this work, we show that we can incorporate relational inductive biases, encoded in the form of relational graphs, into agents. Based on this insight, we propose Grid-to-Graph (GTG), a mapping from grid structures to relational graphs that carry useful spatial relational inductive biases when processed through a Relational Graph Convolution Network (R-GCN). We show that, with GTG, R-GCNs generalize better both in terms of in-distribution and out-of-distribution compared to baselines based on Convolutional Neural Networks and Neural Logic Machines on challenging procedurally generated environments and MinAtar. Furthermore, we show that GTG produces agents that can jointly reason over observations and environment dynamics encoded in knowledge bases.

KEYWORDS

Relational Inductive Bias; Reinforcement Learning; Graph Neural Network

1 INTRODUCTION

Reinforcement Learning (RL) has seen many successful applications in recent years. However, developing agents that can systematically generalize to out-of-distribution observations remains an open challenge [3, 9, 35]. Relational inductive biases are considered important in promoting both in-distribution and systematic generalization, in supervised learning and RL settings [28, 34, 35].

Battaglia et al. [1] define relational inductive biases as constraints on the relationships and interactions among entities in a learning process. Traditionally, relational inductive biases have been hard-coded in an agent's neural network architecture. By tailoring the connections between neurons and applying different parameter sharing schemes, architectures can embody various useful inductive biases. For example, convolutional layers [15] exhibit locality and spatial translation equivariance [21], a particularly useful inductive bias for computer vision, as the features of an object should not depend on its coordinates in an input image. Similarly, recurrent layers [13] and Deep Sets respectively exhibit time translation and permutation equivariance [1, 32].

In this work, we introduce a unified graph-based framework that allows us to express several useful relational inductive biases using the same formalism. Specifically, we frame the computation graph underlying a neural architecture as a directed multigraph with parameter sharing groups denoted by common edge labels connecting shared parameters in each group. This formalization allows us to define specific inductive biases as comprising rules that generate edges and edge labels. The computation is then implemented by Relational Graph Convolutional Networks [R-GCNs, 22], a type of Graph Neural Networks [GNNs, 37] that dynamically construct a computation graph based on a relational graph.

We make use of this formalism to introduce Grid-to-Graph (GTG), a mapping from grid structures of discrete 2D observations to relational graphs, based on a set of *relation determination rules* that generate effective spatial relational inductive biases. Given a feature map with entities (nodes) arranged in a lattice where each entity corresponds to a feature vector, the relations encoded by GTG constrain the flow of information between these entity feature vectors when these features are processed by R-GCNs. We refer to the resulting approach as R-GCN-GTG.

We evaluate R-GCN-GTG in eight tasks: five MinAtar games [30], a procedurally-generated LavaCrossing environment [2], a box-world environment [33] requiring complex relational reasoning, and a symbolic variant of Read to Fight Monsters [RTFM, 36], an environment that provides knowledge bases (KBs) describing environment dynamics that change in every episode. On RTFM, we demonstrate that R-GCN-GTG can not only exploit the spatial information in a feature map, but in addition also relational information without modifying the neural architecture. Our experiments show that R-GCN-GTG produces better policies than Convolutional
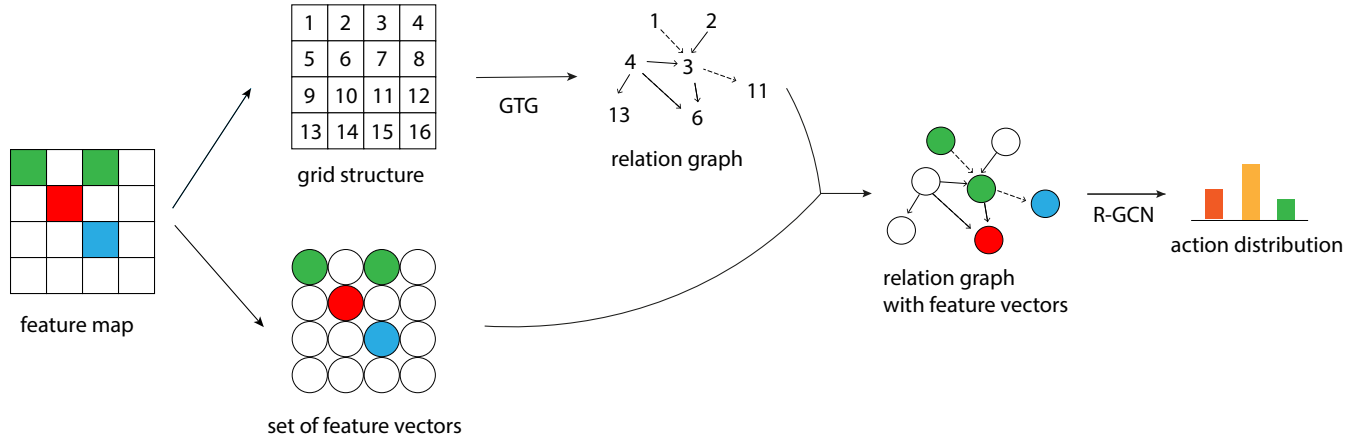
Figure 1: A high-level overview of GTG-R-GCN. We abstract away the grid structure of the feature map and turn observations into a spatial relational graph with GTG. The vectors of the feature map are attached to nodes in the relational graph. We then use an R-GCN to reason over the relational graph and node feature vectors to produce an action distribution.

Neural Networks (CNNs) or Neural Logic Machines [NLMs, 6], a state-of-the-art neural-symbolic model for relational reinforcement learning.

In summary, our main contributions are: *i)* we propose a principled approach for expressing relational inductive biases for neural networks in terms of relational graphs, *ii)* we introduce GTG to transform grid structures represented by a feature map into relational graphs that carrying spatial relational inductive biases, *iii)* we empirically demonstrate that in comparison to CNNs and NLMs, R-GCN-GTG generalizes better in both in- and out-of-distribution tasks in a diverse set of challenging procedurally-generated grid-world RL tasks, and finally *iv)* we show that GTG is able to incorporate external knowledge, enabling R-GCN-GTG to jointly reason over spatial information and relational information about novel environment dynamics without any additional architectural modifications.

2 RELATED WORK

Graph Neural Networks in RL. To our knowledge, NerveNet [28] is the first work that used GNNs to represent an RL policy. Their model follows a similar message-passing scheme as GCNs, maintaining only node feature vectors. NerveNet has been bench-marked on MuJoCo environments, Snake and Centipede, and achieves better in-distribution performance and generalization than Multi-Layer Perceptrons (MLPs). In these environments, NerveNet controls multi-joint robotic avatars. State information and output actions are represented as a graph structure where each node corresponds to a movable part of the agent and local state and actions are attached to each node. Their generalization tests cover size, disability transfer, and multi-task learning. While achieving good systematic generalization performance, Wang et al. [28] focus on the continuous control setting and abstract the graph structure from the morphological information of the avatar. Kurin et al. [17] propose Amorpheus, a transformer architecture for continuous control without relying on morphological information, outperforming NerveNet.

Our work focuses on a different class of tasks for which the input can be represented as a feature map.

Neuro-Logic Models for RL. A separate branch of relational neuro-symbolic models builds directly upon first-order logic, for example, ∂ILP [8] and Neural Logic Machines (NLMs) [6]. Neural Logic Reinforcement Learning [NLRL, 14] applies a modified version of ∂ILP on a varied set of block world tasks and grid-world cliff-walking tasks, displaying robust generalization properties. While ∂ILP has useful strong inductive biases that allow it to generalize well once it has learned a good policy, it suffers from poor scalability and proves difficult to train for more complex logical mappings. Therefore, we use Neural Logic Machines [NLMs, 6], a more expressive and scalable neuro-symbolic architecture, as our baseline. Besides supervised concept-learning tasks, NLMs also have been applied to simple reinforcement learning tasks [6]. In Dong et al. [6], an NLM-based agent is trained to generalize to procedurally generated block world environments and algorithmic tasks. In these tasks, the NLM-based agent surpasses a Memory-Augmented Neural Networks baseline [24] both in terms of in-distribution generalization and out-of-distribution generalization to larger problem sizes. NLM treats relations as inputs and reasons about them in a soft, differentiable manner. However, inductive biases for NLMs remain hard-coded in the architecture. In contrast, R-GCN uses learned relations to express relational inductive biases that constrain message passing.

Works Focusing on Symmetries. Symmetries, especially equivariances, form an important class of relational inductive biases. Some previous works [4, 27] focus on how prior knowledge about symmetries can be incorporated into a model. Group equivariant convolutional networks can represent arbitrary group symmetries, say, rotation and flipping [4]. In RL, MDP Homomorphic Networks [27] express symmetries in the joint state-action space. For these methods, certain relational inductive biases outside of symmetries, like locality in CNNs, still must be hard-coded into the architecture. R-GCN-GTG cannot express equivariance in the state-action space,

but it can recover some symmetries like translation equivalence. However, it is unclear whether it can represent all group symmetries.

3 BACKGROUND

Relational Graphs. We define a relational graph as a labeled, directed multi-graph, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{V}$ is the set of nodes (representing entities), $\mathcal{R}$ is the set of relation labels (representing relation types), $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ is the set of relations (labeled directed edges). Each relation is represented by a tuple (a, r, b) with $a \in \mathcal{V}, r \in \mathcal{R}$, and $b \in \mathcal{V}$, and represents a relationship of type r between the source entity a and the target entity b of the edge.

Relational Graph Convolutional Networks. R-GCNs [22] extend GNNs to model relational graphs. R-GCNs represent a map from the set of feature vector nodes to a new set of feature vectors, conditioned on the relational graph $\mathcal{G}$ and the parameters W_r attached to each relation label. The update rule for a feature vector $\mathbf{x}_a$ of node a is given by:

$$\mathbf{x}'_a := \sigma\left(\sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{N}_a^r} \frac{1}{c_{a,r}} \mathbf{W}_r \mathbf{x}_b + \mathbf{W}_0 \mathbf{x}_a\right), \quad (1)$$

where σ is a non-linearity (such as the ReLu function), $\mathcal{N}_a^r$ denotes the neighboring nodes of a under relation type r, $\mathbf{W}_r$ is the weight matrix associated with r, and $c_{a,r}$ is a normalization constant. In this work, we use $c_{a,r} = |\mathcal{N}_a^r|$. R-GCNs were introduced to deal with graph structured data [22]. Our work presents a new perspective on R-GCNs: we view relational graphs as representing the connectivity and parameter sharing scheme for the model, thereby encoding a prior relational inductive bias.

4 METHODOLOGY

In this section, we describe how relational graphs used by R-GCNs can be adopted to formalize two constraints commonly used in neural architecture design: sparse connectivity and parameter sharing. We introduce GTG, a set of relation determination rules for representing spatial relational inductive biases. GTG strictly generalizes the inductive bias underlying convolutional layers.

Finally, we propose two ways of enabling R-GCNs to jointly reason with visual information restructured according to GTG and potentially additional, external relational knowledge.

4.1 Expressing Relational Inductive Biases Using Relational Graphs

In R-GCNs, message passing is explicitly directed by the relational graph rather than implicitly by the model architecture. Removing the non-linearity and representing $W_0 \mathbf{x}_b$ as a self-loop edge [1], we obtain the following simplified R-GCN update rule:

$$\mathbf{y}_a = \sum_{r \in \mathcal{R} \cup \{0\}} \sum_{b \in \mathcal{N}_a^r} \frac{1}{c_{a,r}} \mathbf{W}_r \mathbf{x}_b. \quad (2)$$

The set of edges determines whether there is message passing between each pair of entities, thereby encoding the connectivity of

---

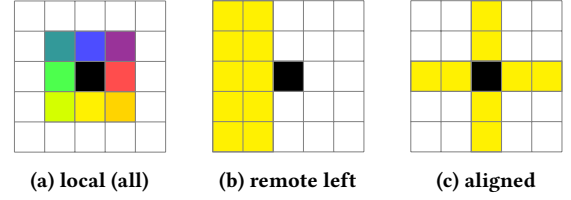**(a) local (all)** **(b) remote left** **(c) aligned**

Figure 2: Three subsets of GTG relation determination rules: the black tile is the target node, and the other colored tiles are source nodes. Tiles with the same relationship to the target node share the same color.

the model. The relationship labels indicate the specific pattern of parameters to be used by the message-passing functions. By making use of different relational graphs, R-GCNs can represent many common neural architectures, including MLPs, CNNs, and DeepSets. We provide a formal description of the neural architectures that R-GCN can represent in Appendix A.3.

To construct the relational graph, we make use of relation determination rules, each defined in the following form:

$$r(a, b) \leftarrow \text{condition},$$

where $r(a, b)$ is a relation from entity a to b with label r, and *condition* is a logic statement. If the condition holds true, relation label r will be appended into $\mathcal{R}_{ab}$, the set of all relation labels of relations between entities a and b. The relation determination rules then express the relational inductive bias by controlling the sparsity and parameter sharing patterns of a feed forward neural network.

4.2 Grid-to-Graph: Spatial Relational Inductive Biases

We now introduce a set of relation determination rules that can be used to construct spatial relational inductive biases. We start by replicating the relational inductive biases of CNNs, and then introduce new biases to address the limitation of CNNs.

4.2.1 Local Directional Relations. The number of possible definitions of spatial relationships between objects is very large, and it may not be feasible to enumerate each of them, let alone empirically evaluate them all. We, therefore, start by mimicking the inductive biases encoded by CNNs, which have been shown to be effective in computer vision tasks and Deep Reinforcement Learning tasks with visual inputs [16, 20, 23]. This provides us with a set of local directional relations. Each local directional relation specifies the relative position of two adjacent entities. A graphical illustration is shown in Fig. 2 (a), where a selected target node is painted black and source nodes are painted with different colors, each corresponding to different relation labels. For clarity, we picked a single node as the target node, though this may not be generally the case. Visualizing all the local directional relations would result in a mesh of edges connecting all nodes to each other.

Consider two entities a and b, and their coordinates x_a, y_a, x_b, y_b. The determination rules for local directional relations are as

---

[1] All the self-loop edges share the same relation label, and term $W_0 \mathbf{x}_b$ is included in the summation.

follows:

$$\text{rightAdj}(a, b) \leftarrow (x_a = x_b + 1) \wedge (y_a = y_b),$$
$$\text{leftAdj}(a, b) \leftarrow (x_a = x_b - 1) \wedge (y_a = y_b),$$
$$\text{topAdj}(a, b) \leftarrow (y_a = y_b + 1) \wedge (x_a = x_b),$$
$$\text{bottomAdj}(a, b) \leftarrow (y_a = y_b - 1) \wedge (x_a = x_b),$$
$$\text{topRightAdj}(a, b) \leftarrow (x_a = x_b + 1) \wedge (y_a = y_b + 1),$$
$$\text{topLeftAdj}(a, b) \leftarrow (x_a = x_b - 1) \wedge (y_a = y_b + 1),$$
$$\text{bottomRightAdj}(a, b) \leftarrow (x_a = x_b + 1) \wedge (y_a = y_b - 1),$$
$$\text{bottomLeftAdj}(a, b) \leftarrow (x_a = x_b - 1) \wedge (y_a = y_b - 1).$$

By only applying these local directional relations, the computation of the associated R-GCN model would be equivalent to that of a convolutional layer with a 3×3 kernel (up to a normalization constant).

4.2.2 Remote Directional Relations. One limitation of convolutional layers is the difficulty of message passing among remote entities. In order to pass information to another node N blocks away, N layers are needed for a CNN with strides length of 1. This problem can be alleviated with larger strides, pooling layers, or dilated convolutions [11, 31], but the model will still require a large number of layers. For less deep CNNs, such as the baseline model used in this work, long-distance message passing is accomplished using dense layers following the convolution layers, as there is no message passing between distant entities. However, dense layers exhibit only a weak relational inductive bias, which can hurt generalization performance. With only local directional relations, R-GCNs inherit the same long distance message passing problem as convolution layers. We, therefore, introduce remote directional relations, which capture the notion of relative positions between objects. We visualize one such remote directional relation, *left*, in Fig. 2 (b). We express remote directional relations using the following rules:

$$\text{right}(a, b) \leftarrow x_a > x_b,$$
$$\text{left}(a, b) \leftarrow x_a < x_b,$$
$$\text{top}(a, b) \leftarrow y_a > y_b,$$
$$\text{bottom}(a, b) \leftarrow y_a < y_b.$$

4.2.3 Alignment and Adjacency Relations. Besides these directional relations, we also add two auxiliary relations, *aligned* and *adjacent*:

$$\text{aligned}(a, b) \leftarrow (x_a = x_b) \vee (y_a = y_b),$$
$$\text{adjacent}(a, b) \leftarrow (|x_a - x_b| \le 1) \wedge (|y_a - y_b| \le 1).$$

Aligned relations indicate if two entities are on the same horizontal or vertical line, visualized in Fig. 2 (c). *Adjacent* relations indicate whether two objects are adjacent to each other, which, unlike local directional relations, carry no directional information.

4.3 Jointly Reasoning with External Knowledge

GTG expresses spatial relational inductive biases in the form of a relational graph. As R-GCN was originally designed to reason over knowledge graphs, it may be tempting to let R-GCN jointly reason over spatial inputs and a task-relevant external relational knowledge graph by simply merging some graph representation of each without further architecture changes. We introduce two ways
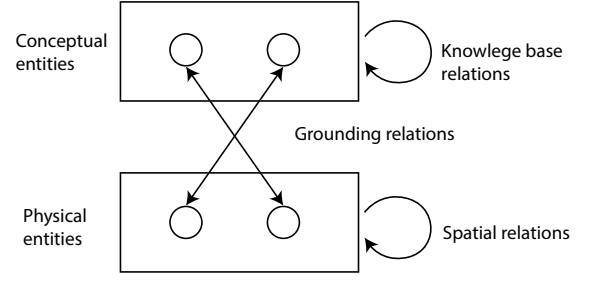

Figure 3: Reasoning with an external knowledge base

of incorporating external relational knowledge: one-hop relations between physical entities and grounding relations with a knowledge graph. Examples of these two approaches applied to RTFM can be found in Section 5.4 and Fig. 5.

4.3.1 Relations between Physical Entities. We refer to each cell in the feature map a physical entity. We can then straightforwardly introduce relational knowledge by adding relations between physical entities. However, this limits the knowledge that can be expressed, as this approach cannot represent more abstract knowledge that describes relations between concepts rather than specific entities, e.g., a shinning weapon can kill fire monsters.

4.3.2 Working with External Knowledge Bases. For enabling the inclusion of external knowledge in our model, we maintain two sets of entities: conceptual entities, which exist in the knowledge base (e.g. the class of an object) and physical entities, which exist in the environment (e.g. a specific object in the environment, such as a monster). We can then link the two graphs corresponding to these two sets of entities with grounding relations so that information can flow between them.

A graphical illustration of this approach is presented in Fig. 3. The grounding relations assign conceptual counterparts to the physical entities. In this work, these relations are handcrafted.

5 EXPERIMENTAL SETTING

5.1 MinAtar

MinAtar [29] is a collection of miniature Atari games. Each observation is a 10×10 feature map, where each cell represents a single object in the game. Unlike conventional Atari games, the environments in MinAtar games are stochastic – for instance in Breakout, the ball starts in a random position. MinAtar environments also set a 10% sticky action [12] probability by default. Sticky actions force the agent to take the action taken in the last step. We enforce a 5,000 steps limit, as some agents can play indefinitely on some of the MinAtar games.

5.2 Box-World

Box-world is a grid-world navigation game introduced by Zambaldi et al. [34]. To solve the game, the agent must collect the gem using the correct key. However, the key is in a locked box, which needs to be unlocked with a separate key. There are also distractor branches

which will consume the current key and produce a key that cannot unlock the gem box. As the game is combinatorially complex, the chance of hitting the correct solution by random walk is low. Zambaldi et al. [34] demonstrated that their RL models required between 2×10^8 to 14×10^8 steps to converge in this environment. Due to limited computational resources, we only train models for 10^7 steps in each environment and further reduce the difficulty of the Box-World environment: *1)* the field size is reduced to 10×10, *2)* the number of distractor branches is set to 1, *3)* the length of distractor branches is set to 1, *4)* the goal length is set to 2. Although we reduced the difficulty, we still preserve all core elements of the Box-World environment. We expect this simplified experimental setup to still allow us to compare the relational reasoning capabilities of different models while reducing the total steps needed for training to convergence.

5.3 LavaCrossing

LavaCrossing is a standard environment from MiniGrid [Minimalistic Gridworld, 2]. The agent must navigate to the goal position without falling into the lava river. The game is procedurally generated, and there are 3 difficulty levels available for each map size, making this is an ideal RL environment to test combinatorial and out-of-distribution generalization of learned policies. MiniGrid environments are by default partially-observable, but we configure our instances to be full-observable. Also, by default, an agent can turn left, turn right, and move forward, which requires the agent to know its direction when navigating to a particular position. We adjust the action space, making the agent able to move in all four directions without turning left or right. The number of lava rivers generated equals the level number. To test the out-of-distribution generalization, we train the agent on difficulty level 2 and test on difficulty levels 1 and 3.

We also design a Portal-LavaCrossing task, illustrated in Fig. 4 (a), to test whether R-GCN-GTG can generalize to non-Euclidean spaces without retraining. After training on difficulty level 2, we transfer the agent to Portal-LavaCrossing where there are no gaps in the lava river. For each side of the lava river, a teleportation portal is placed in a random position: when the agent moves into the portal, it is then placed on the other side of the lava river, and this is the only way to cross it. As no such portals exist in training levels, the agent must be able to generalize to a new environment with non-Euclidean space, leveraging novel test-time spatial relations, or resort to moving around randomly until stepping into the portal. In this task, the CNN baseline is not aware of the portal at all, and can only reach the goal if it moves into the portal by chance. For R-GCNs and NLMs, we append new spatial relationships between portals and other grids: incoming relations to one portal all connect to the paired portal on the other side of the lava river, and outgoing relations from one portal are kept the same. These relations are shown in Fig. 4 (b) and (c).

5.4 Read to Fight Monsters

Read to Fight Monsters [36] is a grid world game, where each level includes a text document providing information about per-episode game dynamics. Each map contains two monsters and two weapons, each randomly generated and positioned. Each weapon

(a) Initial state   **(b) Incoming edges**   **(c) Outgoing edges**
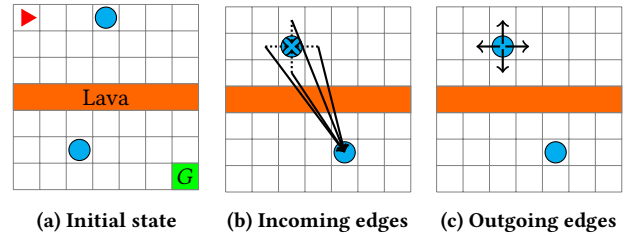
Figure 4: Portal-LavaCrossing tasks. (a) One possible initial state, where blue circles are portals, the red triangle is the agent, and the green block is the goal position. (b) How incoming edges to the portal are attached to the paired portal. Dashed arrows represent the original edges and solid arrows, new edges. (c) Outgoing edges from the portal remain the same.

has a modifier and each monster has an element property. The agent must defeat the monster of a specific element, which can only be defeated with weapons with a specific modifier. The relations describing which modifiers defeat which elements are procedurally generated at the start of each episode and described by the document. Furthermore, each monster belongs to a team. The text document also describes which team must be defeated. Without the text document, the agent can only pick a weapon and attack an arbitrary monster, which leads to an overall win probability of less than 50%. We construct a knowledge base which contains the same information as the original RTFM document based on the grammatical rules that RTFM uses to generate the text document.

We test two approaches of introducing external knowledge to RTFM, shown in Fig. 5. The easier physical-entities-only approach uses two relation labels *target* and *beat*, ignoring the concept of modifiers, elements, and teams. *target(a)* is a unary atom indicating *monster a* is the one the agent must defeat and is appended to feature vectors. The relation *beat(a,b)* (binary atom) means *weapon a* defeats *monster b*. If the agent carries the weapon, *entity a* corresponds to the agent itself. A more complex approach introduces conceptual entities and uses multi-hop reasoning to solve the problem. Along with the physical objects in the environment, this approach considers the conceptual entities of teams, modifiers and elements. This approach introduces additional grounding relations: The relation *assign(a,b)* assigns *modifier a* to *weapon b* or *element a* to *monster b*. The relation *belong(a,b)* indicates that the *monster a* belongs to *team b*. The relation *beat(a,b)* states that *modifier a* defeats monsters of *element b*. The relation *target(a)* means that the agent must defeat *team a*. Finally, *hold(a)* indicates the agent currently holds a weapon with *modifier a*.

5.5 Architecture Overview

In this subsection, we describe how GTG and R-GCN can be incorporated into an RL policy. First, the state of the environment is rendered as a feature map. Specifically, each tile in a grid world is represented as a binary-valued feature vector $\mathbf{x}$. These feature vectors $\mathcal{X}$ are attached to nodes and GTG generates edges between these nodes, forming a relational graph $\mathcal{G}$ that represents particular relational inductive biases. If required, extra knowledge about
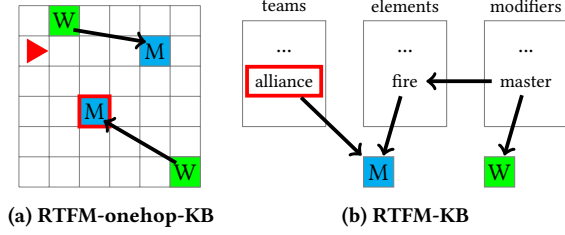
(a) RTFM-onehop-KB

(b) RTFM-KB

Figure 5: RTFM tasks with KB encodings of the text document. The red triangle represents the agent; the blue block, the monster; and the green block, the weapon. Figure (a) shows the one-hop reasoning version of KB encoding, where arrows indicate beat relations, and the red frame indicates the target monster. Figure (b) illustrates the multi-hop reasoning version. Arrows indicate a relation between two entities. The red frame indicates the opponent team. Relation labels are not represented explicitly in the graph.

game dynamics expressed as a knowledge base can be merged into this multigraph. Subsequently, the R-GCN acts on this multi-graph and associated feature vectors. After processing using R-GCN, we apply a feature-wise max-pooling to all node feature vectors. The outputs are then fed to dense layers that output per-action logits. The graphical illustration of the whole process can be found in Fig. 1. The probability of actions can thus be written as:

$$P(\mathbf{a}|\mathcal{X}) = \text{softmax}(\text{MLP}(\text{maxpool}(g(\mathcal{X}, \mathcal{G}; \boldsymbol{\theta}_1)); \boldsymbol{\theta}_2)), \quad (3)$$

where g is the stack of R-GCN layers and $\boldsymbol{\theta}$s are neural network parameters.

A separate head performs value estimation:

$$\hat{v}(\mathcal{X}) = \text{MLP}(\text{maxpool}(g(\mathcal{X}, \mathcal{G}; \boldsymbol{\theta}_1)); \boldsymbol{\theta}_3)). \quad (4)$$

During training, the agent samples actions according to this resultant action distribution. During testing, we take the action with maximum probability. In our experiments, we make use of IMPALA [7], a policy-gradient algorithm to train our RL models. We based our IMPALA implementation on TorchBeast[18] and our R-GCN implementation, on Pytorch-Geometric[10]. Our implementation is available at https://github.com/ZhengyaoJiang/GTG.

6 RESULTS AND DISCUSSION

In this section, we report our empirical results comparing R-GCN to baseline methods regarding in-distribution performance, out-of-distribution combinatorial generalization, and the ability to incorporate external knowledge. We also report ablations probing the effectiveness of different relation determination rules and components of R-GCN-GTG.

6.1 In-Distribution Generalization

Fig. 6 shows the in-distribution performance (training curve) of CNNs and relational models on MinAtar and LavaCrossing tasks. For each model, we run 5 trials, each with different random seeds. The thin, opaque lines in the plot represent training curves corresponding to each run, and the bolded lines represent mean episodic return averaged over all five runs. Here, NLM and R-GCN use GTG

with all three classes relationships, namely, local directional, remote directional, and auxiliary relations. We can see that R-GCN-GTG models consistently perform either better or on-par with CNNs and NLM-GTGs across all eight environments. For Asterix, Seaquest, Box-World and Breakout, R-GCN-GTG outperforms CNNs by a significant margin. As the CNN baseline in the RTFM environment is unable to access information in the knowledge base, it acts as an informative baseline, for which only visual information is available. NLM-GTGs also achieve good performance on Seaquest and Breakout, but they are inferior to CNNs on other MinAtar tasks. We also mark the best performance of original MinAtar baselines [30] as a red horizontal dashed line. It is worth noting that we use a deeper network than these baselines, which include a deep Q-network [19] and actor-critic with eligibility traces [5, 25], trained on twice as many steps. Thus, our CNN agent acts as a much stronger baseline than the original MinAtar models.

There are two potential reasons for the large performance gain of R-GCN-GTG over CNNs. Firstly, R-GCN-GTG does not make use of the absolute position of objects, but instead only taking into account the relative positions between objects. In contrast, CNNs employ dense layers to reason globally. Further, these dense layers have a weak relational inductive bias which can negatively impact sample efficiency and generalization. Secondly, GTG provides more flexible message passing than conventional CNN layers in terms of long-range dependencies. We further study the roles of various relation determination rules and max-pooling after convolutions in our ablation studies.

Although the NLM-GTG models in our experiments make use of relational graphs determined by GTG, it uses this information in a less structured way. Specifically, NLM-GTGs encode such relational information as dense vector representations, whereas R-GCN-GTG uses this information to construct a GNN, thus directly determining the computation graph and flow of messages. The performance gain of R-GCN-GTG over NLM-GTG suggests that using GTG to determine the specific relational inductive bias, and therefore guide message passing in a structured way, results in better in-distribution performance.

6.2 Out-of-Distribution Systematic Generalization

In Table 1 and Table 2, we show how policies learned by our relational models can generalize to environments outside of the training distribution. For our LavaCrossing experiments, we train the agent on difficulty level 2 and test the policy on difficulty levels 1 and 3. Table 1 shows average returns over five training runs. Each model is evaluated on 200 test episodes. The relative performance change with respect to the training environment is shown in parentheses. All the models generalize and perform optimally on difficulty level 1. However, when generalizing to difficulty level 3, the relational models perform significantly better than CNNs. R-GCN-GTG generalizes the best among all the models we tested.

Table 2 demonstrates the win rate for each model in the symbolic variant of the RTFM tasks. Again, we report mean returns averaged over five training runs. Each section of the table represents a different task variation. We transform the text document into a symbolic knowledge base of triples for RTFM-KB and RTFM-onehop-KB.
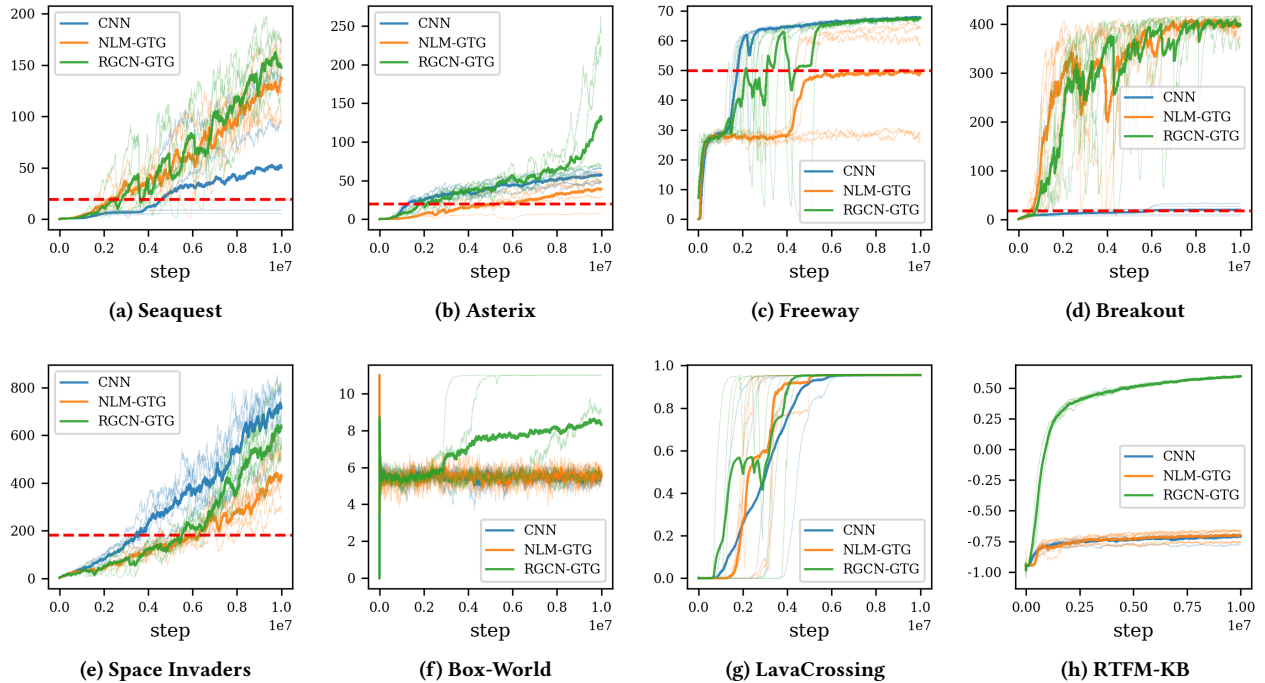
Figure 6: Training curves of CNN and relational models. The opaque lines represent the returns of individual runs, while the bolded lines, the average of 5 runs. Red dashed lines mark the final performance of the best model reported by the original MinAtar baselines.

Model	Level 2	Level 1	Level 3	Portal
CNN	0.958	0.960	0.790(-17.5%)	0.040(-95.8%)
NLM-GTG	0.955	0.960	0.918(-3.9%)	**0.158**(-83.5%)
R-GCN-GTG	0.958	0.960	**0.942**(-1.7%)	0.096(-90.0%)

Table 1: Out-of-distribution generalization performance on LavaCrossing. The agent is trained on difficulty level 2.

task	room size model	6×6	10×10
RTFM-KB	NLM-GTG	21.0%	19.6% (-6.7%)
	R-GCN-GTG	86.3%	96.6% (+12.0%)
RTFM-onehop-KB	NLM-GTG	93.0%	99.4% (+6.9%)
	R-GCN-GTG	93.0%	98.2% (+5.6%)
RTFM-text	txt2π	55%	55% (+0%)

Table 2: In-distribution and out-of-distribution generalization in RTFM variants. Figures report the win rate increment between 10×10 environments and 6×6 environments

This makes the task easier compared to the original RTFM-text task (last row) as models do not have to learn to encode information presented as textual inputs. Further, the RTFM-onehop-KB is easier than RTFM-KB as the RTFM-KB require multihop reasoning. We also put the performance of the model proposed by the RTFM paper [36], txt2π in RTFM-text environment into the table. We train on environments with a grid size of 6×6 and test generalization performance on environments with a grid size of 10×10. An optimal policy in the 10×10 environments should achieve better performance compared to that on the smaller environments, as the agent has more space to evade monsters. We observe that NLM-GTG and R-GCN-GTG generalize well to the larger environments. However, R-GCN-GTG performs much better than NLM-GTG in the harder RTFM-KB environment, both in terms of in-distribution (6×6 grid) and out-of-distribution generalization (10×10 grid).

6.3 Incorporating External knowledge

The Portal-LavaCrossing and RTFM experiments demonstrate the flexibility of GTG in incorporating different kinds of external knowledge. In Table 1, we can see that with spatial information provided by the KB, the NLM-GTG and R-GCN-GTG agent managed to generalize to Portal-LavaCrossing in a zero-shot manner. The RTFM results in Table 2 show how GTG enables the relational model to jointly reasoning with both spatial information and environment dynamics information when represented as a KB. The R-GCN-GTG agent performs well both in multi-hop reasoning and one-hop reasoning variants of RTFM-KB, but NLM-GTG only performs well in the easier one-hop variant.
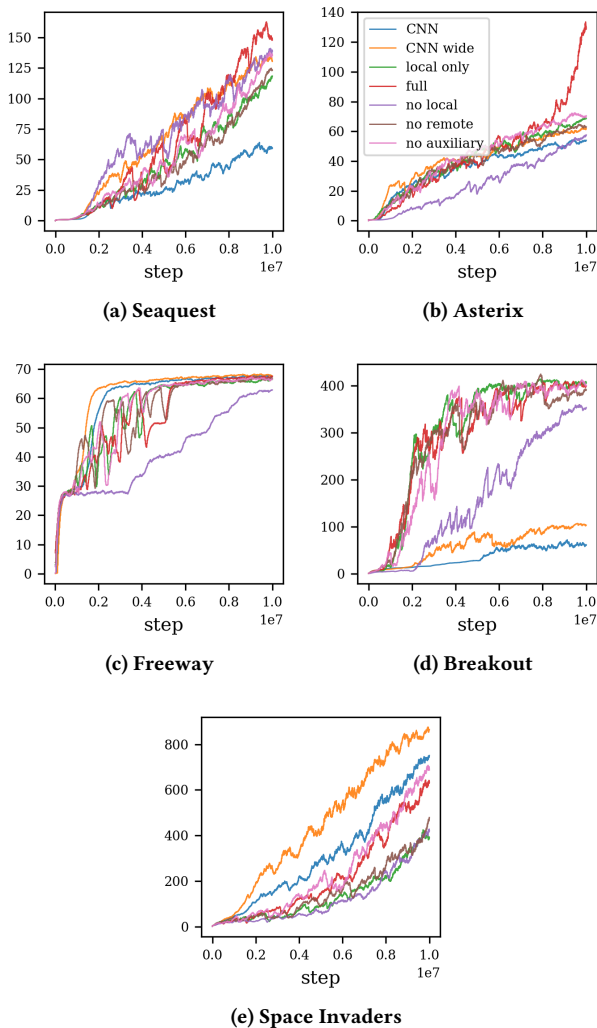
(a) Seaquest

(b) Asterix

(c) Freeway

(d) Breakout

(e) Space Invaders

Figure 7: Ablation of R-GCN-GTG on MinAtar tasks.

6.4 Ablation Study

Fig. 7 presents the results of an ablation study of three relational inductive biases encoded using GTG (local, remote, and auxiliary relations). Each line shows the smoothed training curve averaged over five training runs. The red lines represent the training curve of R-GCN using the full set (local, remote, and auxiliary relations) of relational inductive biases. The green lines show the performance of R-GCNs using local directional relations only, whose convolution computation is equivalent to that of the image convolution in Section 4.2. A notable difference among these models is the information aggregation method between convolution layers and dense layers: the CNN model concatenates all output feature vectors (i.e. flattened), while the R-GCN model applies a max-pooling layer. The flattened vector in the CNN model tends to have high dimensionality, thereby increasing the total number of parameters in the adjacent dense layer. Therefore, when constraining the number of parameters of the two architectures to be approximately equal, most

of the parameters of the R-GCN model reside in convolution layers, whereas more parameters of the CNN model reside in the dense layers. This explains the performance difference between CNN and R-GCN-GTG with local directional relations only. In Seaquest and Breakout, R-GCNs yield better performance than CNNs, while in Space Invaders CNNs outperform R-GCN-GTG. The two models achieve similar performance in Asterix and Freeway.

To further investigate the role of max-pooling, we evaluated a wider CNN with the same number of parameters as the convolution layer in the local-only R-GCN-GTG model. This wider model flattens features before dense layers rather than applying max-pooling, resulting in 876k parameters (the full R-GCN-GTG model has 149k). We use this wider CNN model to isolate the performance improvement that results from max-pooling. Comparing CNN-wide and R-GCN-GTG local-only, we see mixed results: the two models achieve comparable performance on Seaquest, Asterix and Freeway; local-only R-GCN-GTG performs better in Breakout, while the wider CNN model performs better in Space Invaders. This shows that max-pooling by itself does not outperform flattening if we do not care about the number of parameters.

Unsurprisingly, removing local directional relations undermines the performance of R-GCN-GTG in almost all of the tasks, which shows the importance of the relational inductive bias of locality. We also assessed the impact of remote and auxiliary relations, observing that using both sets of relations improves performance on Seaquest, Asterix and Space Invaders. The benefits of introducing these additional relational inductive biases are robust in the sense that they do not degrade performance any environment. In contrast, this is not true for max-pooling. These improvements demonstrate that we can go beyond the relational inductive bias of CNNs by using the relation determination rules of GTG, which provide a flexible framework for expressing many useful connectivity and parameter sharing constraints

7 CONCLUSION

This paper introduced Grid-to-Graph, a principled framework for representing relational inductive biases. GTG is based on a set of relation determination rules, which act on inputs in the form of a feature map corresponding to discrete 2D state observations. Using these relation determination rules, GTG transforms 2D observations into a multigraph input for an R-GCN model. The resulting architecture, R-GCN-GTG outperforms both CNNs and Neural Logic Machines, the previous state-of-the-art in deep relational RL, on MinAtar and a series of challenging procedurally-generated grid world environments, both in terms of in-distribution performance and out-of-distribution systematic generalization. Our results further show that GTG provides an effective and straightforward interface for incorporating various forms of external knowledge without any architectural modifications.

REFERENCES

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018).

[2] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

[3] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. 2020. Leveraging Procedural Generation to Benchmark Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 2048–2056. http://proceedings.mlr.press/v119/cobbe20a.html

[4] Taco Cohen and Max Welling. 2016. Group Equivariant Convolutional Networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 2990–2999. http://proceedings.mlr.press/v48/cohenc16.html

[5] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. 2012. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*. IEEE, 2177–2182.

[6] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural Logic Machines. In *ICLR (Poster)*. OpenReview.net.

[7] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. 80 (2018), 1406–1415.

[8] Richard Evans and Edward Grefenstette. 2018. Learning Explanatory Rules from Noisy Data. *Journal of Artificial Intelligence Research* 61 (Jan 2018), 1–64. https://doi.org/10.1613/jair.5714

[9] Jesse Farebrother, Marlos C. Machado, and Michael Bowling. 2018. Generalization and Regularization in DQN. *CoRR* abs/1810.00123 (2018). arXiv:1810.00123 http://arxiv.org/abs/1810.00123

[10] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[12] Matthew J Hausknecht and Peter Stone. 2015. The Impact of Determinism on Learning Atari 2600 Games.. In *AAAI Workshop: Learning for General Competency in Video Games*.

[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[14] Zhengyao Jiang and Shan Luo. 2019. Neural Logic Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 3110–3119. http://proceedings.mlr.press/v97/jiang19a.html

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[17] Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. 2020. My Body is a Cage: the Role of Morphology in Graph-Based Incompatible Control. In *International Conference on Learaning Representations (ICLR)*.

[18] Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. 2019. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552* (2019). https://github.com/facebookresearch/torchbeast

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

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb 2015), 529–533. https://doi.org/10.1038/nature14236 arXiv:1312.5602

[21] Markus Püschel and José M. F. Moura. 2008. Algebraic Signal Processing Theory: Foundation and 1-D Time. *IEEE Trans. Signal Process.* 56, 8-1 (2008), 3572–3585. https://doi.org/10.1109/TSP.2008.925261

[22] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[24] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-To-End Memory Networks. In *NIPS*. 2440–2448.

[25] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[26] T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

[27] Elise van der Pol, Daniel E. Worrall, Herke van Hoof, Frans A. Oliehoek, and Max Welling. 2020. MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/2be5f9c2e3620eb73c2972d7552b6cb5-Abstract.html

[28] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. 2018. NerveNet: Learning Structured Policy with Graph Neural Networks. In *ICLR (Poster)*. OpenReview.net.

[29] Kenny Young and Tian Tian. 2019. MinAtar: An Atari-inspired Testbed for More Efficient Reinforcement Learning Experiments. *arXiv preprint arXiv:1903.03176* (2019).

[30] Kenny Young and Tian Tian. 2019. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176* (2019).

[31] Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. (2016).

[32] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. In *NIPS*. 3391–3401.

[33] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. 2018. Relational Deep Reinforcement Learning. *arXiv preprint* abs/1806.01830 (June 2018).

[34] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter W. Battaglia. 2019. Deep reinforcement learning with relational inductive biases. In *ICLR (Poster)*. OpenReview.net.

[35] Chiyuan Zhang, Oriol Vinyals, Rémi Munos, and Samy Bengio. 2018. A Study on Overfitting in Deep Reinforcement Learning. *CoRR* abs/1804.06893 (2018). arXiv:1804.06893 http://arxiv.org/abs/1804.06893

[36] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. 2020. RTFM: Generalising to New Environment Dynamics via Reading. (2020).

[37] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

A APPENDIX

A.1 Hyperparameters

All models used in our experiments, including relational models and the baseline CNN, have 4 hidden layers. We train all models using RMSprop [26] with a learning rate of 0.001.

In the RL setting, an agent can, in principle, produce an infinite amount of training data given enough computational power and time, implying that increasing the model size can almost always be helpful for in-distribution performance. For fair comparison, we aimed to ensure similar parameter counts for all relational models tested. Taking LavaCrossing as an example, the CNN model contains 149k total parameters; R-GCN, 149k; and NLM, 139k.

We also evaluated even larger CNN models but did not observe significant improvements to in-distribution performance. We only tested NLM with maximum arity of 2, because of the large computational cost of higher arity. Each layer and each arity had an output dimension of 64, resulting in 192 intensional predicates per layer. R-GCN models used 2 relational convolution layers, each with an outputting 64-dimensional feature vectors. After max-pooling, we then apply 2 dense layers, each with 128 hidden units. CNN models used 2 convolution layers, each with a 3×3 kernel and 12 features, and 2 dense layers, each with 128 hidden units.

A.2 Relational Inductive Bias and Generalization

Prior work [1] defines relational inductive bias as constraints on relationships and interactions among entities in a learning process. Usually, the relational inductive bias is implemented by a specific pattern of parameter sharing and neural connectivity in the neural architecture. For example, CNNs exploit local connections constraining information processing to a limited, local receptive field, sharing parameters between different local kernels to introduce spatial translation invariance. Such relational inductive biases have been argued to be critical for promoting combinatorial generalization [1].

Combinatorial generalization is the process of exploiting compositional structure underlying a problem to successfully perform inference, prediction, or useful behaviours on previously unseen examples or scenarios [1]. This concept is closely related to systematic generalization, a defining capability human intelligence, which modern deep learning methods have yet to attain. Combinatorial generalization can help the model improve sample efficiency and generalize to new tasks. In practice, it is important to disentangle combinatorial generalization from memorization. One way to explicitly test for combinatorial generalization is to test the model on out-of-distribution held-out data (sampled from a different distribution than the training data). This data should be generated by a process mirroring the compositional rules governing the generation of the training data.

Unlike in-distribution generalization, which has been well-studied by learning theory and statistics, out-of-distribution generalization cannot, in general, be achieved by simply increasing the amount of data. However, successful combinatorial generalization would allow out-of-distribution generalization on such held-out data that follows similar composition rules as the training data.

In the RL setting, given a fixed policy π, the sampled data consists of trajectories of states, actions and rewards. We call an RL environment *out-of-distribution* if, at test time, either initial state, dynamics, or the task distribution in multi-task learning, is varied such that the distribution of trajectories differs from that at training. We say an environment or collection of environments (with some appropriate sampling distribution over these environments) is *in-distribution* if it generates trajectories with the same likelihood as under the training distribution. For example, consider the case in which the initial state of an environment is procedurally generated and, during testing, we keep all the level generation logic the same, only using different random seeds. Though the agent may see initial states it never saw in training, we will would not call these test-time configurations out-of-distribution, as the probability that the agent meets these initial states remains the same at test time as at training time.

A.3 Matrix Representation of Message Passing

Here we provide a block matrix formalization of a class of neural network layers, shedding light how different neural architectures can be represented by their respective relational inductive biases. Suppose we have n m-dimensional input feature vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^m$, and want to map them to n output feature vectors $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^m$ of the same dimensionality.

If we consider the case where this mapping is linear, we can concatenate all the input feature vectors $\mathbf{x}_i$ and express the transformation as a block matrix product:

$$\begin{bmatrix} \mathbf{A}_{11} & \cdots & \mathbf{A}_{1b} & \cdots & \mathbf{A}_{1n} \\ \vdots & \ddots & & & \vdots \\ \mathbf{A}_{a1} & & \mathbf{A}_{ab} & & \mathbf{A}_{an} \\ \vdots & & & \ddots & \vdots \\ \mathbf{A}_{n1} & \cdots & \mathbf{A}_{n} & \cdots & \mathbf{A}_{nn} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_b \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_a \\ \vdots \\ \mathbf{y}_n \end{bmatrix}$$

We see the update rule of $\mathbf{y}_a$ is:

$$\mathbf{y}_a = \sum_{b=1}^{n} \mathbf{A}_{ab} \mathbf{x}_b \tag{5}$$

The submatrix $\mathbf{A}_{ab} \in \mathbb{R}^{m \times m}$, which we refer to as the *message passing matrix*, dictates the message passing from entity a to entity b. When no constraints are applied to $\mathbf{A}$, the overall mapping expressed in Appendix A.3 corresponds to a standard dense linear layer. In the following sections, we refer to the constraints over the message passing matrices $\mathbf{A}$ as the relational inductive bias. This formalization provides a more concrete definition of relational inductive bias than the conceptual one proposed by Battaglia et al. [1], i.e., constraints on relationships and interactions among entities in a learning process.

Neural architectures typically encode two types of constraints: sparse connectivity and sharing parameters. Sparse connectivity can be achieved by setting some of the $\mathbf{A}$ components to zero. For example, $\mathbf{A}_{ab} = \mathbf{0}$ means no message can be passed from entity b to entity a. Parameter sharing simply sets some of the message passing matrices to correspond to the same matrix. Specific patterns of connectivity and parameter sharing encode different relational

inductive biases, ensuring specific, desirable properties during computation, represented by message passing operations. For instance, the locality of a convolutional layer can be accomplished by keeping connectivity from a to b only if entity a is in the receptive field of entity b. Leveraging the [32, Lemma 3], we can also implement the permutation equivariance (the inductive bias of Deep Set [32]) by sharing parameters across diagonal submatrices and sharing parameters among the off-diagonal submatrices.

The relational graph of R-GCN provides a natural way to describe arbitrary connectivity and parameter sharing constraints. By rearranging and applying the distributive law, we can express the message passing matrix as:

$$\mathbf{A}_{ab} = \sum_{r \in \mathcal{R}_{ba}} \frac{1}{c_{a,r}} \mathbf{W}_r, \tag{6}$$

where $\mathcal{R}_{ba}$ is the set of all relation labels for relations from b to a, and $c_{a,r}$, $\mathbf{W}_r$ are the normalization constant and weight matrix of R-GCN in Eq. (2).

We now provide an example demonstrating the equivalence of R-GCN with local directional relations and 2D convolution: The update rule for a single feature vector of a linear convolution layer can be written as $\mathbf{y}_a = \sum_{b \in K_a} \mathbf{A}_{ab} \mathbf{x}_b$, where K_a is the set of entities in the receptive field around a and $\mathbf{A}_{ab}$ is the same value for all a and b with the same relative position to each other. Given , the equivalence between R-GCN with local directional encoding and image convolution becomes clear. The only difference is that R-GCN introduces an extra normalization factor, which is a constant in this case, as most of the nodes have the same number of incoming edges.

A.4 Relational Inductive Biases of Convolutional Layers

We now consider the case where entities are associated with a 2D feature map and analyze the relational inductive biases of convolutional layers. For all tasks we consider, each environment observation is (or contains) a *feature map*, defined as a mapping $m : W \times H \to P$ with $W = \{0, ..., w\}$, $H = \{0, ..., h\}$ and $P = \mathbb{R}^m$, where w and h is the width and height of the input, and n is the dimensionality of the feature vector. Each feature vector corresponds to an entity and therefore, a node in the relation graph. When dealing with a feature map, a single 2D convolution layer has the following relational inductive biases: *i) Locality*: Information is only transmitted from nodes in a kernel-sized receptive to the central node. Formally, A_{ab} is not a zero matrix only if a is in the reception field around b. *ii) Anisotropy*: The the propagation of information in different directions follows different rules. Using our block matrix formalization: in a reception field around entity a, the value of A_{ab} for each distinct b is distinct. *iii) Spatial translation equivariance*: Node pairs with the same relative position share the same message passing rules. Namely, A_{ab} and A_{cd} should be the same if the relative position between a, b and that between c, d are the same.

Locality is a useful relational inductive bias for most environments where feature maps abstract (or are sampled from) the physical world because the objects that are close to each other are more likely to inform or interact with each other. For example, in the grid world tasks we considered, most interactions happen locally and even remote interactions typically rely on local, intermediary objects (e.g., an intermediary object thrown by entity a to a remote entity b). Note that spatially adjacent objects may be further abstracted as a single larger entity.

Anisotropy can be used to judge directionality, useful in the RL environments considered in this work, as actions are bound to directions e.g.,"*move left*" and "*move right*".. We emphasize this inductive bias to highlight why relations should be directional in GTG. Note that MLPs are by default anisotropic

Translation equivariance is an important inductive bias which captures the intuition that interactions between two entities likely depended on the relative positions between these two entities rather than their absolute position. Translation equivariance relates to the assumption commonly made by feature engineering in image processing, where and the rules for extracting features should be independent of the feature position. Here, each filter corresponds to a high-level feature vector.

A.5 Computational Costs

The computational cost of our model is proportional to the number of ground atoms (i.e.,edges in the KB and thus in the computational graph) induced by the underlying relations. Let N be the number of entities (e.g.,pixels). Our model's time complexity ranges from $O(N)$ for convolutional layers to $O(N^2)$ for fully-connected graphs, e.g.,Transformer layers. For GTG's relation determination rules, the time complexity of local directional relations is $O(N)$, that of alignment relations (defined in Section 4.2.3) is $O(N^{3/2})$, and that of remote relations is $O(N^2)$.