2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Tensor Action Spaces for Multi-agent Robot Transfer Learning

Devin Schwab^1, Yifeng Zhu^2, Manuela Veloso^1

Abstract— **We explore using reinforcement learning on single and multi-agent systems such that after learning is finished we can apply a policy zero-shot to new environment sizes, as well as different number of agents and entities. Building off previous work, we show how to map back and forth between the state and action space of a standard Markov Decision Process (MDP) and multi-dimensional tensors such that zero-shot transfer in these cases is possible. Like in previous work, we use a special network architecture designed to work well with the tensor representation, known as the Fully Convolutional Q-Network (FCQN). We show simulation results that this tensor state and action space combined with the FCQN architecture can learn faster than traditional representations in our environments. We also show that the performance of a transferred policy is comparable to the performance of policy trained from scratch in the modified environment sizes and with modified number of agents and entities. We also show that the zero-shot transfer performance across team sizes and environment sizes remains comparable to the performance of training from scratch specific policies in the transferred environments. Finally, we demonstrate that our simulation trained policies can be applied to real robots and real sensor data with comparable performance to our simulation results. Using such policies we can run variable sized teams of robots in a variable sized operating environment with no changes to the policy and no additional learning necessary.**

I. Introduction

Reinforcement learning (RL) has recently made major strides thanks to the use of deep RL methods. Deep RL has been used to learn policies for various tasks from Atari to robot control [1], [2], [3], [4], [5], [6], [7], [8]. Through these many successes, it has been shown that deep reinforcement learning (RL) can deal with high-dimensional state inputs, represented as tensors (e.g. images). However, these policies are typically fixed to the environment they were trained in and cannot adapt to situations where the number of agents changes after training. In many real world problems such as RoboCup robot soccer [9], it is necessary for agents to be adaptable to such changes. Additionally, many systems are trained in a limited laboratory environment. We would like policies and agents that can adapt to a larger or smaller environment after training is finished.

In this paper we specifically focus on the problem of zero-shot transfer. That is a policy learned with a set number of agents and entities in a set size environment can be applied without changes or additional learning when the number of agents and entities change or the environment size changes. To zero-shot transfer across changes to number of agents and entities, we use a transformation of the raw state and action space of a Markov Decision Process (MDP) into an abstracted one, such that *both* state *and* actions are multi-dimensional tensors. This new space is designed so that new policy parameters do not need to change. To facilitate zero-shot transfer across environment sizes we use a special network architecture, Fully Convolutional Q-Network (FCQN), that uses only convolutions and deconvolutions. This allows an arbitrary sized input tensor to produce a proportionally sized action tensor output.

In this paper, we specifically look at domains where multiple robots or agents are performing a task in which spatial positioning is important. Using a properly designed tensor state and action space gives us two advantages:

1) Spatial information can be preserved through the entire policy network.
2) For a fixed environment size, the dimensionality of the state and action space remains fixed even as the number of agents and other entities in the environment vary.

Rather than using a standard flat action space, where the spatial relationships between state and actions are lost, we can use a tensor for both states and actions allowing those relations to be preserved. By designing the tensors in such a way that the dimensionality stays fixed, despite changes in the number of agents, it is easier to do zero-shot transfer of single and multi-agent policies, as the policies do not need to learn additional parameters to deal with the additional inputs/outputs.

As mentioned, we utilize Fully Convolutional Q-Networks (FCQN), a network containing only convolutions and deconvolutions [10] to our tensor-based state and action encoding. Such layers preserve spatial information, and allow for arbitrary sized inputs. We show that such an architecture, combined with using tensors as the action space can allow for:

1) Faster convergence of policies tasks where spatial positioning is important.
2) Can be used as state and action space dimensionality changes due to environment size changes, thus allowing for zero-shot transfer across different environment sizes.

In the rest of the paper, we formalize the requirements for transforming back-and-forth between a standard state and action representation and a tensor representation. We present a specific pair of position-based mapping functions. We explain how the FCQN is utilized with our representation. We explain how policy learning and zero-shot transfer is

^{1}Devin Schwab and Manuela Veloso are with Carnegie Mellon University, Pittsburgh, PA, USA dschwab@anderw.cmu.edu mmv@cs.cmu.edu
^{2}Yifeng Zhu is with University of Texas, Austin, Austin, TX , USA. The work was done during the visit at CMU. yifeng.zhu@utexas.edu

done in both a single and multi-agent context. We present an empirical analysis across a pair of test environments, showing that the tensor representation and FCQN architecture allows for faster training and that policies can be transferred, zero-shot, across environment sizes and team sizes with minimal difference in performance as compared to training policies from scratch in the changed environments.

Finally, we show that these policies, despite being trained purely in simulation function with comparable performance on a set of real robots. We test on a robot system designed for the SSL RoboCup Competition [9].

II. Related Work

The DQN Atari paper kicked off a long line of work utilizing images as state representation for deep reinforcement learning (RL) [1], [2]. Images, a special case of multi-dimensional tensors, can encode large amount of information without the need for intensive feature engineering. In this work we do not limit ourselves to environments that naturally have image observations, instead we explore how different observations can be mapped to an image-like, tensor-based representations.

Tensor-representations have been used for states and actions in the game of Go [7], as the game board can be easily treated as a 2D-tensor. However, such works did not explore how other environments could be mapped to and from such a representation in order to speed up learning and facilitate transfer.

The concept of tensors for both states and actions was briefly introduced under the name "Image Action Space MDP (IAS-MDP)" [10]. We build off of this work, focusing on generalizing the functions that map back-and-forth between the original state-action space the tensor state-action space. We give a specific mapping for environments where positions play an important role in success. Unlike this previous work, we also apply the learned policies to real world robots and sensor data.

Like in the IAS-MDP work [10], we utilize the Fully Convolutional Q-Network (FCQN) architecture, which contains only convolution and deconvolution layers [11]. Removing fully connected layers allows learners to take advantage of translational invariance of the convolutional layers so that experience can be generalized to other areas of the state-tensor. Secondly, this allows a single set of policy weights to be applied to different size input tensors with no additional training. We present additional empirical evidence beyond what is shown in [10], showing that this architecture combined with the tensor representation can not only train faster than a standard representation, but also has comparable policy performance after transfer to that of a network trained from scratch in the transferred environment.

We also explore learning multi-agent policies and how the tensor-based representation can facilitate transfer across team sizes. Most prior work on multi-agent transfer are adaptations of single agent transfer techniques to multi-agent scenarios including: object oriented MDPs, task mappings, experience sharing, and supervision from more experienced agents [12], [13], [14], [15]. Here we focus on representations that allow for easy transfer.

III. Methodology

The concept of mapping functions that map back-and-forth between standard state action spaces and our tensor representation is first introduced. Then a specific transformation for MDPs with position features is described. Finally, an explanation of how single and multi-agent policies in the tensor-based state and action space can be learned and represented by the Fully Convolutional Q-Network (FCQN) architecture is given.

A. Mapping MDPs to Tensor-based MDPs

1) Markov Decision Processes: An MDP is a set of states $s \in S$ and actions $a \in A$, with an associated transition distribution $p(s_{t+1}|s_t, a_t)$ specifying the probability of transitioning from state s_t to s_{t+1} under action a_t. Given a reward function $f(s, a) \in I\!R$ and a discount factor $\gamma \in [0, 1)$, the goal of learning is to optimize for the parametric policy $\pi_\theta(s) \to a$ with parameters θ with respect to the cumulative discounted reward $E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | a_t = \pi_\theta(s_t), s_{t+1} \sim p(\cdot|s_t, a_t), s_0 \sim p(s)\right]$, where $p(s)$ is an initial state distribution. Alternatively, we can maximize an action-value function $Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_\pi\left[Q(s_{t+1}, \cdot)|s_{t+1} \sim p(\cdot|s_t, a_t)\right]$, so that the optimization goal is: $J(\theta) = \mathbb{E}_{p(s)}\left[Q^\pi(s, a)|a \in \pi_\theta(s)\right]$.

Generally, an MDP has some set of entities, X, which are either agents or objects relevant to the task. Each entity will have some set of features, $\forall x \in X, F^x$, associated with it. These generally have semantic meaning such as position, velocity, etc. The state is typically a vector concatenating these features, $S = \left[F^0; F^1, \cdots\right]$. In this standard representation, changing the number of entities or features changes the dimensionality of the state vector.

Assume a high-level action space consisting of both discrete actions, $a_d \in A_d$, and parameterized actions, $a_p(y) \in A_p$ where y is an entity in the environment [16]. The action set is the combination of both types: $A = A_d \cup A_p$. A discrete action may be something like "move left". A parameterized action may be "pass ball to y", where parameter y is another agent on the team. Parameterized actions can be treated as discrete actions by grounding the parameters, allowing any discrete action reinforcement learning algorithm to learn a policy. The number of entities also affects the action dimensionality, by changing the number of possible groundings.

2) Tensor-based States and Actions: The goal is to map from a feature based state S to a new state space S_t that has a fixed dimensionality regardless of the number of entities in the environment. Additionally, a new action space A_t must be constructed that can map back to the original action space A. If the dimensionality of either S or A change, then a policy π learned in S and A will require a reparameterization. This need for new parameters will prevent zero-shot transfer. Instead, by construction S_t and A_t using tensors that have fixed dimensionality regardless of number of entities, a policy π_t learned in S_t and A_t will require no new parameters to

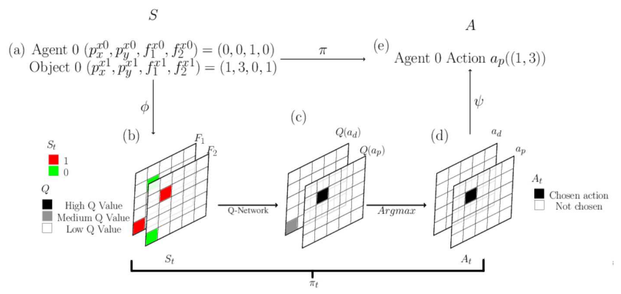


Fig. 1: Example transformation to the tensor-based state and action representation functions. For a 5x5 grid world with 2 entities: agent 0 and object 0. **(a)** The feature based state, each entity have 2 features. **(b)** The tensor based state using the position ϕ mapping. **(c)** Example Q-value output from a policy network. **(d)** Result of creating the one-hot action tensor by marking the argmax of the Q-values. **(e)** Results of mapping the action tensor back to the original parameterized action space via ψ, which results in a_p with parameter object 0 being chosen.

work, regardless of number of entities. Additionally, by using tensors, we can take advantage of network architectures that better uses spatial information in a task.

Let $\phi : S \to S_t$ be an injective mapping function, where S_t is a multi-dimensional tensor with fixed dimensionality regardless of the number of entities in the environment. Many environments, such as Atari and grid-worlds, can be naturally represented as an image, which has these properties.

Let $\psi : A_t \to A$ be a surjective mapping function, which maps from our new tensor action space back to the original MDP action space. A_t is also a multi-dimensional tensor with fixed dimensionality regardless of number of entities in the environment.

Given these mapping functions we can then choose to learn a policy $\pi_t : S_t \to A_t$ instead of a policy $\pi : S \to A$. Figure 1 shows how this tensor based representation relates to the original MDP. A traditional MDP policy maps directly from (a) to (e) in the figure. Whereas, our approach first transforms from (a) to (b), then learns a policy mapping from (b) to (d), and is then transformed back to the original action space (e).

3) Optimality of Policies in the Tensor Space: Because ϕ is injective, the new tensor state space is still fully observable. The ψ mapping is surjective, meaning that it is always possible for the policy to select every action available in the original action space. Therefore, the optimal policy in this new tensor representation will have the same value as the optimal policy in the original MDP representation. The new representation has a fixed dimensionality regardless of number of entities, yet can still represent a policy equal in value to the optimal policy in the original state and action space.

B. Position Based Mappings

In the rest of the paper we consider tasks where all entities have position features and give a specific pair of mapping functions ϕ and ψ.

1) MDP Environments: For spatially orientated tasks there is an "environment", which is implicitly encoded in the MDP transition function (e.g. no transitions to positions outside the environment, Euclidean space, etc.). An environment, E, can be defined for an MDP, where the environment has some spatial dimensions D, where $E \in I\!R^D$. All entities must exist inside the environment bounds. To be observable, entities must have positions in the environment: $p^{x_i} = (p_1^{x_i}, \cdots, p_D^{x_i}) \in E$. If we assume finite environments, then $p_i \in [e_i^{min}, e_i^{max}]$. For the rest of the formulation we will assume that positions are discrete or can be discretized while keeping the task solvable.

2) Position based ϕ and ψ Mappings: Assume each $x_i \in X$ has a position and a single feature f^{x_i} representing information such as an ID number or team affiliation. We can construct a tensor z_j of dimensionlity D, that spans the full environment E. Positions in the tensor corresponding to an entity's position will take the feature value f, and all other positions will get a default value. Mathematically:

$$z_j(k_1, \cdots, k_D) = \begin{cases} f_j^{x_i} & k_1 = p_1^{x_i}, \cdots, k_D = p_D^{x_i} \\ c_{default} & \text{otherwise} \end{cases}$$
$$s.t. \quad x_i \in X, \quad f_j^{x_i} \neq c_{default}, \quad (k_1, \cdots, k_d) \in E \tag{1}$$

Because every position in the environment has a corresponding position in the tensor, entities can be added and removed without affecting the dimensionlity, only the values in the tensor.

Most environments have multiple features for each entity. We can construct one z_j tensor for each feature type and then stack them along a new dimension to give us a full state: $S_t = z_1 \times \cdots \times z_{|F^x|}$. This includes all of the information in the original state, but with a fixed dimensionality $D+1$, where the first D has size equal to the span of the environment and the final dimension has size equal to the number of feature types.

For 2D environments, this is like a multi-channel 2D image. Figure 1 from (a) to (b) shows an example of mapping a 5×5 grid world with 2 feature types to this representation.

Actions can be represented as a one-hot, $D+1$ dimensional tensor. The first D dimensions have size equal to the environment, and the final dimension is equal to the number of action types: $|A_p|+|A_d|$. To select a discrete action, the agent marks it's own position in the action tensor. To select a parameterized action, it marks the parameter's position in the action tensor. This action space will be larger than the original action space. Actions that do not have a matching action in the original action space can be mapped to a "do nothing" action or a similar default action. Figure 1 part (d) shows an example of an action tensor where action a_p is selected with parameter corresponding to object 0's position.

3) Policy Learning with Tensor Representations: Agents can use any RL algorithm that learns an action-value function. We could directly learn an action-value function that takes the state and an arbitrary action tensor as an input: $Q^{\pi}(s_T, a_T)$. However, we are using a one-hot action tensor, so there are $|A_T|$ actions for the agent to choose from. Therefore, we can learn an action-value function which has an output equal in size and dimensions to A_T. An argmax is applied to this Q-value tensor to get the one-hot action tensor required by ψ. Figure 1 (c) shows an example Q-value tensor output. Part (d) shows how the argmax transforms this dense tensor into the 1-hot action tensor required by the ψ mapping.

4) Fully Convolutional Q-Networks (FCQN): We use a Fully Convolutional Q-Network (FCQN), which has no fully connected (FC) layers, only convolutions and deconvolutions [10]. With a tensor represention, FCQN architecture has multiple advantages compared to networks with an FC layer. 1) (de)convolutions take advantage of locality 2) (de)convolutions take advantage of translational invariance and 3) (de)convolutions can accept any size input and output a proportionally sized output. Locality and translational invariance can allow an agent to generalize experiences across positions in the state-space. By accepting any size input, we can also do zero-shot transfer across environment sizes. Figure 2 shows an outline of this architecture.

C. Multi-agent Policies

With the above framework, we can learn policies that do not require new parameters when the number of agents in an environment changes. Therefore, we can learn multi-agent policies and perform zero-shot transfer across team sizes. Each agent will evaluate its own copy of the policy network to choose its own actions based on its state, which will include information about its own position as well as masking channels like those used in [10].

To coordinate, we use an averaging layer as described in [17], as we can average across any number of agent inputs and receive a fixed size output. In theory any set of differentiable aggregation functions can be used (e.g. sum, average, products), so long as the function can output a fixed size output for any size input. To remove the need to find new weights for added agents, all agent policies use the same weights. Adding or removing an agent simply adds or removes a copy of the shared network weights.

IV. Empirical Results - Simulation

We first train and test our policies in simulation, hypothesizing that the FCQN and tensor representation can:

1) Speed-up learning compared to standard representations
2) Perform zero-shot transfer across team sizes with minimal change in performance
3) Perform zero-shot transfer across environment sizes with minimal change in performance

A. Environment Descriptions

We use two different environments for our experiments: `Passing` and `Take-the-Treasure`[1]. `Passing` and `Take-the-Treasure` are grid world environments first used in [10]. Both are grid worlds with standard moves, as well as a do nothing action and a pass ball/throw treasure to location action. `Passing` has a single controlled agent with the goal of passing to an environment controlled entity marked as a teammate as quickly as possible. The environment also contains environment controlled opponents which block passes. `Take-the-Treasure` has a team of learning agents against a team of fixed policy environment controlled opponents. The agent team starts with treasure and must keep the treasure holder from being captured by the opponents for as long as possible.

B. Hyperparameters and Network Architecture

The `Passing` FCQN network is shown in figure 2. Skip connections were used between all matching sized pairs of conv/deconv layers. All hidden layers use ReLU activations.

Below is the network structure for `Passing` using state features and fully connected layers. `c(8,4,64)` represents a convolution layer with an 8×8 filter, a stride of 4, and 64 channels. `f(512)` represents a fully connected layers whose output size is 512 hidden units. `maxpooling(3,3)` represents a max pooling layer which has `(3,3)` as the size of the pooling window. All hidden layers use ReLU activations. The network structure is: `f(256)`, `f(256)`, `f256`, and `f(`$|A$`)`.

The `Passing` network with the tensor state input and the FC layers is: `c(12,4,32)`, `c(5,1,32)`, `maxpooling (3,3)`, `c(5,1,512)`, `flatten`, then we have the dueling layer which has a value stream as `f(2048)`, `f(1)`, and an advantage stream as `f(2048)`, `f(`$|A|$`)`. All hidden layers use ReLU activations. `Take-the-Treasure` uses the same network architecture.

All networks were trained using the Double DQN algorithm [18] and the Adam optimizer with hyperparameters shown in table I. We used ϵ-greedy exploration with epsilon decayed linearly from ϵ_{start} to ϵ_{end}.

C. Learning Performance

Figures 4a, 4b shows the training curves for an FCQN. There are also baseline curves of network with fully connected layers applied to the tensor representation, and a

[1]Code and environment details https://github.com/tensor-state-action-spaces/tensor-state-action-env

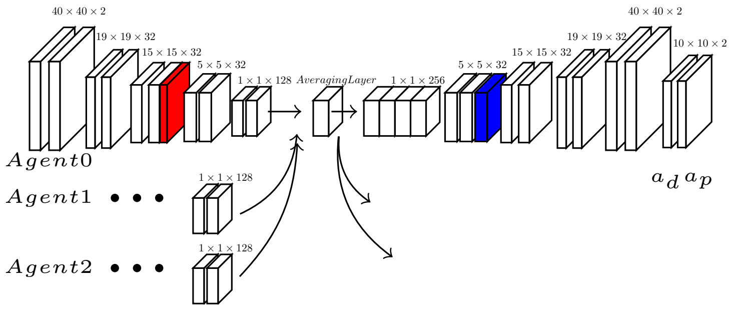


Fig. 2: FCQN network architecture with multi-agent averaging layer. Red: max-pooling. Blue: bilinear upsampling

Name	Value
`Passing` learning rate	1×10^{-5}
`Take-the-Treasure` learning rate	1×10^{-4}
ϵ_{start}	0.99
ϵ_{end}	0.1
ϵ decay steps	1,000,000
`Passing` replay mem size	1,000,000
`Take-the-Treasure` replay mem size	1,200,000

TABLE I: Hyperparameters used during training and evaluation

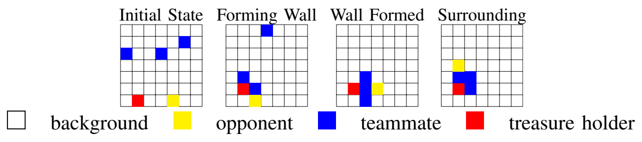
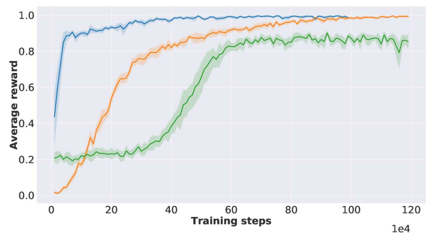


Fig. 3: Frames from the `Take-the-Treasure` domain. Teammates learn to form a walls and surround treasure holder to block the opponents from reaching the treasure holder.

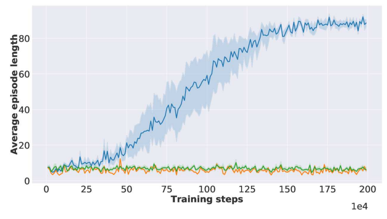
dueling network architecture using a state vector and discrete action space [19]. For each evaluation point, ϵ was set to zero and 100 episodes were run. For `Passing` we evaluate the average reward during testing because it is a single agent environment. For `Take-the-Treasure` we evaluate episode length, as we care about the whole team performance, not individual agent performance. This whole procedure was repeated from scratch 10 times in order to get the shaded error bars. Both the FCQN and the fully-connected network have approximately 3.5 million weights.

In both `Passing` and `Take-the-Treasure` the FCQN with tensor representation converges significantly faster than either the tensor representation with FC layers or the standard representation with a dueling architecture. In fact, the other representations fail to learn a meaningful policy on `Take-the-Treasure` within the allocated training steps. This demonstrates that the combination of FCQN and tensor representation can lead to significant speed-ups in some tasks. Note that for `Passing`, a reward of 1.0 is the best possible outcome, showing that we can indeed learn an optimal policy.

`Take-the-Treasure` was trained with 3 agents vs 3 opponents on a 10x10 environment. The teammates learn a coordinating policy, where the treasure holder runs from opponents, while teammates form walls or surround the holder to block opponents. Figure 3 shows a small sequence of this behavior.



(a) `Passing` training curve



(b) `Take-the-Treasure` training curve

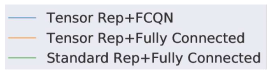


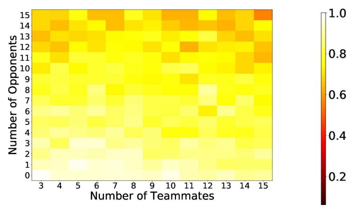
Fig. 4: Average training performance.

D. *Zero-Shot Transfer Across Team Sizes*

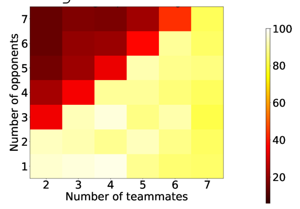
We trained one `Passing` network with a team size of 3 vs 3 opponents on a 20x20 environment and one `Take-the-Treasure` network with 3 agents and 3 opponents on a 10x10 grid. Figures 5a and 5b show the average reward achieved by the policy as it transfers across team size and opponent team size. Brighter is better.

For `Passing`, performance is fairly consistent across both the numbers of opponents and the number of teammates. Only when there are large number of opponents, such that few passing opportunities occur, do we see a major difference in performance. For `Take-the-Treasure`, we see good transfer performance, when the team sizes are equal or the agent team is larger. This is expected, as 1) the randomly chosen starting state is less likely to have the treasure holder already surrounded by opponents and 2) it is easier for more opponents to out maneuver the teammates.

1) Performance of Transfer vs Trained from Scratch: While the transferred policy has similar performance before and after transfer, it may actually perform poorly vs a policy trained directly with the new team size. We trained policies from scratch on different team sizes for 500,000 steps and



(a) `Passing` transfer as team sizes vary.



(b) `Take-the-Treasure` transfer as team sizes vary.

Fig. 5: Performance of transferred policy as team sizes vary. The brighter the cell the better the average performance.

compared this policy performance vs the zero-shot transfer performance of the policy from the previous section. We evaluate the relative performance defined as $(R_T - R_S)/R_S$ where R_T is the average reward/episode length for the transferred policy and R_S is the average reward/episode length for the trained from scratch policy.

Table II shows the statistics for relative transfer performance on `Passing` and `Take-the-Treasure`. `Passing` was evaluated for every team size combination from Figure 5a. `Take-the-Treasure` was evaluated across 7 different team size combinations.

`Passing` transferred performance is only slightly worse on average, with a small standard deviation. `Take-the-Treasure` has a larger standard deviation, but on average the transferred policies actually slightly outperform the from scratch policies. Likely, it is more difficult for the agent to learn in a larger environment. Not only is the state and action space larger, but the rewards are more sparse due to agents being more spread. Conversely, by training in a smaller environment and then transferring, the agent gets faster feedback and has less state-action space to explore while training, but can then apply what it has learned directly in the now larger environment.

	Min	Max	Mean $\pm$ Std Dev	Metric
`Passing`	-0.25	0.2	-0.03 $\pm$ 0.07	Reward
`Take-the-Treasure`	-0.11	0.26	0.14 $\pm$ 0.31	Episode Length

TABLE II: Relative performance of policies transferred to different team sizes vs policies trained from scratch with those team sizes.

E. Zero-Shot Transfer Across Environment Sizes

Table III shows the average transferred policy performance as environment size changes. Each new size evaluated the transferred policy over 100 independent episodes. `Passing` was trained on a 20x20 grid and then evaluated as height and width were varied independently from 20 to 40 with steps of 5. `Take-the-Treasure` was trained on a 10x10 grid and transferred to grids of different heights and width varied independently from 10 to 20 in steps of 2.

Just like when transferring across number of agents in the environment, we see that transferred performance is relatively consistent as environment size changes. This is despite never seeing any training data from the new environment sizes.

	Mean $\pm$ Std Dev	Metric
`Passing`	0.89 $\pm$ 0.03	Rewards
`Take-the-Treasure`	84.67 $\pm$ 6.55	Episode Length

TABLE III: Performance of transfer across environment sizes.

V. Empirical Results - Real Robot

In the previous section we used simulations of the grid world environment `Take-the-Treasure` as our test domain. In this section, we apply these grid world policies as a high-level decision making policy on a real robot system. First we explain the hardware system. Then we explain how these low-level continuous states and actions are used with our high-level `Take-the-Treasure` policy and tensor state-action spaces. Finally, we give evaluation results for running these policies on the real robot system.

A. Hardware Setup

We test our simulation trained policies using robots designed for Small Size League (SSL) RoboCup [9]. Figure 6 shows a picture of these robots. The robots are controlled via radio at 60 Hz and can move omni-directionally.

Robot positions and orientations are detected via a standard SSL overhead camera system and AprilTags [20]. Figure 7 shows an overview of the field setup. New detections are published at 60Hz.

B. State and Action Mappings

We use the policies trained in simulation with no additional learning performed on the real robots. We run on a field size of approximately 5m by 3m. Positions are discretized into a grid with cells being approximately the diameter of a robot (i.e. 18cm by 18cm), leading to a 28×17 tensor state. The continuous positions from the cameras are discretized to this grid and used to construct the tensor representation using the mapping described previously. Orientation is ignored as the robots can move omni-directionally. The environment tracks who controls the treasure and which team each robot belongs to and adds this information to the tensor state.

The move actions are translated to continuous desired positions in real-world coordinates and used as set points of a PID controller running at 60Hz. Passing the treasure updates the internal environment state of who currently has the treasure.

New decisions are made by the agents and opponents at a fixed 5Hz, regardless of whether the robot has finished executing the previous decision. Unlike in simulation, where an action is guaranteed to execute to completion before a new action is chosen, in the real world system, an agent may not end up at the requested position by the time of the next decision. However, because the policy is deterministic, it is only possible for a new controller setpoint to be chosen if at least one of the robots has moved in the tensor representation.

C. Evaluation

We ran 30 runs of 3 vs 3 robots collecting the number of steps taken in each episode. Like in simulation each episode starts with random initial positions of all the robots. Table IV shows the results of these experiments compared to simulation results.

We can see that despite the policies only training in simulation with perfect action execution and no noise on the input state, the policies have only a slight drop in performance when executed on the real robot hardware. This is despite noise in the computer vision detections, imperfect action execution and asynchronous execution of the agent actions.

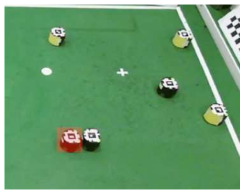


Fig. 6: `Take-the-Treasure` running on SSL RoboCup robots. Yellow robots are opponents. Highlighted robot currently has the treasure.

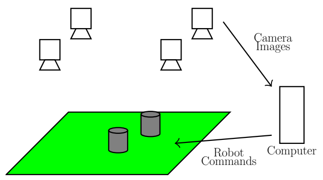


Fig. 7: System overview.

	Mean ± Std Dev
Real Robot	71.2 ± 17.7
Simulation	76.13 ± 15.6

TABLE IV: Real-robot `Take-the-Treasure` policy performance. 3 vs 3 robots on a 28×17 grid corresponding to roughly a 5m by 3m real world environment.

VI. Conclusion

We have presented a general framework for transforming standard state and action representations into a tensor-based representation. The tensor representation can be designed for easy zero-shot transfer by making the dimensions invariant to the number of agents. We have formalized a specific mapping function for spatial environments with position information. Future work will explore other mapping functions for different environment types. Using the FCQN architecture, we show empirically that we can learn single and multi-agent policies faster than standard representations while supporting zero-shot transfer across team sizes and environment sizes with minimal change in performance compared to training from scratch. Finally, we demonstrated that our abstract, high-level tensor representation is applicable to real robot execution and real sensor data.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: http://arxiv.org/abs/1312.5602v1

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *Internal Conference on Learning Representations*, 2016. [Online]. Available: http://arxiv.org/abs/1509.02971v5

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016. [Online]. Available: http://arxiv.org/abs/1602.01783v2

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[8] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[9] The RoboCup Federation, "RoboCup," 2017. [Online]. Available: http://www.robocup.org/

[10] D. Schwab, Y. Zhu, and M. Veloso, "Zero shot transfer learning for robot soccer," *AAMAS*, 2018.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[12] F. L. Da Silva and A. H. R. Costa, "Transfer learning for multiagent reinforcement learning systems." in *IJCAI*, 2016, pp. 3982–3983.

[13] G. Boutsioukis, I. Partalas, and I. Vlahavas, "Transfer learning in multi-agent reinforcement learning domains," in *European Workshop on Reinforcement Learning*. Springer, 2011, pp. 249–260.

[14] A. Taylor, I. Duparic, E. Galván-López, S. Clarke, and V. Cahill, "Transfer learning in multi-agent systems through parallel transfer," 2013.

[15] D. Garant, B. C. da Silva, V. Lesser, and C. Zhang, "Accelerating multi-agent reinforcement learning with dynamic co-learning," Technical report, Tech. Rep., 2015.

[16] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2016.

[17] S. Sukhbaatar, a. szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2244–2252. [Online]. Available: http://papers.nips.cc/paper/6398-learning-multiagent-communication-with-backpropagation.pdf

[18] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461v3

[19] Z. Wang, T. Schaul, M. Hessel, H. v. Hasselt, M. Lanctot, and N. d. Freitas, "Dueling network architectures for deep reinforcement learning," 2015. [Online]. Available: http://arxiv.org/abs/1511.06581v3

[20] M. Krogius, A. Haggenmiller, and E. Olson, "Flexible layouts for fiducial tags," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.