GROWING UP TOGETHER: STRUCTURED EXPLORATION FOR LARGE ACTION SPACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Training good policies for large combinatorial action spaces is onerous and usually tackled with imitation learning, curriculum learning, or reward shaping. Each of these methods has requirements that can hinder their general application. Here, we study how growing the action space of the policy during training can structure the exploration and lead to convergence without any external data (imitation), with less control over the environment (curriculum), and with minimal reward shaping. We evaluate this approach on a challenging end-to-end full games army control task in StarCraft: Brood War by training policies through self-play from scratch. We grow the spatial resolution and frequency of actions and achieve superior results compared to operating purely at finer resolutions.

1 INTRODUCTION

Deep reinforcement learning's success stems in part from the handling of large state spaces via powerful function approximation. But how can algorithms cope with large high-dimensional or combinatorial action spaces? Approaching domains with larger action spaces over longer horizons leads to an explosion of the trajectories space. Plenty of methods exist to structure their exploration: imitation and inverse reinforcement learning (Schaal, 1997; Ng et al., 2000), curriculum learning (Selfridge et al., 1985; Elman, 1993; Bengio et al., 2009), reward shaping (Ng et al., 1999), options (Stolle and Precup, 2002), hierarchical RL (Dietterich, 2000). Each respectively requires traces (or demonstrations), modifying the task (or environment), prior knowledge on the value structure, on the actions structure, or on the state structure. These requirements limits the applicability of the methods.

We study an approach that leverages prior knowledge on the action and policy space to greatly reduce the sample complexity of exploration: growing the action space (GAS) of a policy during its training. GAS is quite generally applicable: it does not require demonstrations, nor modifying the environment or changing the reward(s), and the induced hierarchy of actions stays actionable (not abstract). Instead, it requires (i) a form of hierarchy or abstraction over actions, (ii) a representation of different levels of details within a single model (iii) a training procedure and a growth schedule. Here, for (i), we leverage a spatial and temporal hierarchy in representing the action space. We start by learning a policy and its associated value function at the coarser levels of detail (low resolution and low frequency actions). Those form the basis of the computation of the policy at finer levels of detail, and get refined over the course of training. We represent this policy (ii) as a ConvNet (LeCun et al., 1990) encoder-decoder with LSTMs for integrating partial observations (Hochreiter and Schmidhuber, 1997). We decode directly to the finest action resolution and pool spatially and temporally to obtain coarser actions. And for (iii) we use self-play, with IMPALA (Espeholt et al., 2018), and Population Based Training (Jaderberg et al., 2017) to hyper-optimize the action space growth schedule. This training process can be seen as a curriculum self-induced by the policy's performance so far.

We propose to demonstrate the efficacy of GAS with an experimental study on the game of StarCraft: Brood War. It is a partially observable real-time strategy game where players can control hundreds of units, executing simultaneous, durative actions, for games averaging 15-20 minutes. The state space is large, but the major problem for exploration in StarCraft comes from the combinatorial action space. The game engine runs at 24 frames per second, and potentially at each frame the player could issue a different action. All actions in StarCraft are the combinations of a set of units (or single unit) executing a single command type, optionally at a target position. The lower bound of the branching factor b from (Ontanón et al., 2013) (number of actions at any point in time) is 10^{50}, and a 15 minute

game with one action per second corresponds to depth d of 900, which gives a game complexity $b^d = (10^{50})^{900}$. In this paper, we control only the military units (all the non-workers units), and restrict ourselves to one action per second, a 128×128 (or 64×64) target/position resolution, and the 3 major types of actions for military units. Already, the average number of possible actions (e.g. with 80 alive units, each selected or not) is approximately $2^{80} \times 128 \times 128 \times 3 \approx 5.9 \times 10^{28}$. Our motivation for applying GAS to StarCraft is that an average human can play strategic full games with 30 effective actions per minute, with imprecise clicks and timings, and grouping units together. We wager it is more important to master those broad strokes first, before filling in the details of optimal micromanagement. Our policy neural network is trained end-to-end from scratch through self-play. In our experiments we start by taking one action every 16 seconds, up to one per second, and start with positions at an 8×8 resolution, up to 128×128 (or 64×64 for smaller "maps"). We also group units spatially, first favoring sampling only one group down to the uniform sampling of arbitrary groups (from a single unit to all units). The number of possible actions in our example thus grows from $2^1 \times 8 \times 8 \times 3 = 384$ actions to 5.9×10^{28} over the course of training.

This approach is anything but limited to StarCraft, and is just a generalized form of refining the discretization of continuous variables. All reinforcement learning problems include a temporal dimension, some have spatial priors or graph connected objects. Most multi-agent or continuous control problems induce clear action abstractions (options). Let us now give the high-level example of controlling a robotic arm (with k motors/degrees of freedom) to reach an object: it is not a problem where we can reset or modify the environment easily, we have a notion of distance from the hand to the object. We can explore the action space by using only one (or some) motor(s) to start with, freezing the others, unblocking those additional degrees of freedom as this limited exploration leads us to some sampling of the value function. This type of policy (and its growing/GAS version) can for instance be represented with an auto-regressive model over degrees of freedom. We discuss intrinsic hierarchies and action abstraction in more detail in §3.

In summary, we explain how hierarchical modeling of the action space aligns with the goal of successively growing it. We propose a spatio-temporal action space resolution decomposition in the context of real-time strategy games. We showcase its potency as an exploration curriculum in StarCraft, where, to the best of our knowledge, we train the first end-to-end model for military units control in full games of StarCraft only with self-play, from scratch.

2 Related Work

Curriculum learning The motivation for curriculum learning is to "start small" and to present a learning algorithm with data of increasing difficulty. This concept is particularly appealing in the context of neural networks as their standard optimization algorithm, (stochastic) gradient descent, is already incremental in nature (Elman, 1993; Bengio et al., 2009). For reinforcement learning, curricula are commonly defined over tasks of increasing difficulty (Selfridge et al., 1985). They rely foremost on a suitable environment that allows the definition of such tasks, and curricula are problem-specific and require careful tuning. One approach to automate the definition of tasks is reverse curriculum learning, in which initial environment states are generated with gradually increasing distance from goal states (Florensa et al., 2017). In adversarial settings such as games, self-play (Samuel, 1959; Tesauro, 1995; Silver et al., 2017) represents a natural form of curriculum as agents are faced with opponents of the same or a similar skill-level (Bansal et al., 2017). The effectiveness of self-play has also been demonstrated in (Jaderberg et al., 2018; Silver et al., 2017; Baker et al., 2019) by solving a wide variety of complex tasks. Sukhbaatar et al. (2017) extend self-play induced curricula to not just adversarial games but also reversible or resettable games.

Hierarchical RL and options A general approach for tackling challenging tasks is to exploit inherent structure in the domain of interest. In reinforcement learning, this can be generally attempted via hierarchical reinforcement learning, with the options framework being a popular formalization (Sutton et al., 1999). Policies operating at different levels of temporal abstraction promise faster learning and better transfer between tasks. Various methods of automatic option discovery have been proposed (McGovern and Barto, 2001; Stolle and Precup, 2002). Other methods propose tailored construction methods of neural network policies leading to hierarchical behavior (Dayan and Hinton, 1993; Vezhnevets et al., 2017). Other approaches that aim to exploit hierarchical structure concern themselves with the state or observation space. With a suitable state abstraction and proper

aggregation, exploration can be greatly accelerated Nouri and Littman (2009). Finding the right abstraction is then critical, and one can either settle for a static one or begin with a coarse state space that is refined over the course of learning (Moore, 1994; Munos and Moore, 2002). Both approaches do again require sufficient domain knowledge.

Large action spaces In this work we propose to exploit structure in the action space on top of that in the state space. If the number of possible actions is very large, which is the case in combinatorial action spaces, ensuring sufficient exploration of their effects is a challenging endeavor. To this end, Farquhar et al. (2019) introduced the concept of growing action spaces (GAS) on a multi-agent micro-management scenarios in StarCraft. Here, successively increasing the action space from few (one) to several groups of units introduces a curriculum as the learning agent obtains successively finer control options. We extend this approach with both a spatial and temporal action hierarchy and apply it to unit control in full StarCraft games. A similar concept was utilized by Murali et al. (2018) with a curriculum over dimensions of control of a robotic arm. In this case, the curriculum prescribes a sampling strategy for the dimensions in order to guide exploration. Czarnecki et al. (2018) propose a curriculum over agents by mixing two policies defined on a coarse and fine action space, respectively. In contrast, our work integrates action hierarchies with multiple levels into a single policy parameterization.

3 GROWING ACTION SPACES

We present a formalisation of our approach for Markov Decision Processes (MDP), that is the tuple $\mathcal{M} \coloneqq \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions, $\mathcal{P}$ the transition probability between states conditioned on actions, $\mathcal{R}$ the reward function for reaching a state through an action, and γ is the reward discount factor. The goal is find an optimal policy for $\mathcal{M}$, noted π^* that maximizes the discounted sum of rewards (the return). For a different $\mathcal{A}' \subset \mathcal{A}$ action set (and $\mathcal{A}'' \subset \mathcal{A}'$, etc.), possibly only a subset $\mathcal{S}' \subseteq \mathcal{S}$ is reachable, and through that only a subset of the codomain of $\mathcal{R}$ (by abusing the notation, let $\mathcal{R}'$ be the induced/limited reward function). Nonetheless, it is often possible to extract a smaller (i.e. with less transitions, smaller sample complexity) MDP $\mathcal{M}'$ from a given original MDP $\mathcal{M}$. To the extend that $\mathcal{R}'$ samples salient values of $\mathcal{R}$, this smaller MDP $\mathcal{M}'$ may be helpful to structure/speed-up the exploration of $\mathcal{M}$. We note π' a policy in $\mathcal{M}'$, and because $\mathcal{A}' \subset \mathcal{A}$ and $\mathcal{S}' \subseteq \mathcal{S}$, $\pi^{*\prime} \subset \pi$. Note that nothing guarantees that $\pi^{*\prime} \subset \pi^*$. Thus, (i) using GAS is in general an assumption or prior that one can use $\mathcal{M}'$ as an exploration proxy or curriculum for $\mathcal{M}$, (ii) in order to not be in a situation where the transfer from $\pi^{*\prime}$ to π^* makes exploration harder in $\mathcal{M}$, it may be beneficial *not* to reach the optimal policy in $\mathcal{M}'$, which for instance can be achieved by mixing the levels (training π and π' simultaneously) (Czarnecki et al., 2018; Farquhar et al., 2019), or by entropy regularization. Note that nothing requires the representations (or models) for π' and π to share parameters. We now give some categories and examples of GAS, as well as strategies to represent the hierarchies and grow the action space through them.

In the continuous control setting (e.g. MuJoCo (Todorov et al., 2012)), we can often assume a C-Lipschitz value function Q where $|Q(x, u + \epsilon) - Q(x)| < C\epsilon$. In those cases, given a continuous action space $\mathcal{A}$, it is commonplace to discretize it as $\mathcal{A}'$, here the different levels of abstraction correspond to different discretization resolution. Another possibility is to use only a selected subset of the action dimensions, or, for instance, a subset of the actuators in the robotic arm from the introduction. For categorical action spaces with a natural hierarchy or taxonomy, we can leverage it: whenever we can naturally build a tree over the set of possible actions, we can grow the action space from the roots of this tree. These abstractions exist naturally in card games like Poker (Sandholm, 2015), and reinforcement learning environments such as DMLab (Czarnecki et al., 2018).

However, there is not always a trivial taxonomy structure available to restrict the action space. For instance, when it consists of selecting a subset from a set of N elements, the action space grows exponentially as 2^N. One can allow selections of at most or at least $M < N$, but this does not always make sense. These actions still sometimes have a spatial meaning, for example traffic control for deciding which traffic lights should be green, or scheduling multi-nodes jobs on a cluster where performance can be impacted by the network proximity between 2 nodes. This is also the case for the problem of StarCraft unit selection. In our experiments, we often reach states where the model controls more than 50 units, leading to an action space with $2^{50} \approx 1.12 \times 10^{15}$ actions (possible subsets). It would be unreasonable to rely on pure chance to explore it entirely, and indeed humans

don't need millions of hours of game-play to understand that the number of meaningful options is quite limited: for instance selecting all of the units or a single group of spatially close units. We leverage the spatial information on the set elements to improve exploration by introducing correlated sampling. From the beginning, the policy outputs a probability for each set element to be selected. However, sampling is changed to ensure that spatially nearby set elements are more likely to be selected together than not. More generally, correlated sampling can be applied as long as a meaningful distance measurement can be defined between actions.

In the options framework proposed by (Sutton et al., 1999; Stolle and Precup, 2002), a Markov option is executed by either choosing a new option with probability $\beta(s_t)$, or otherwise following the current option. One method of growing the resolution in this case would be to slowly warm up β during the training procedure. This results in agents at the beginning of training following options for a long time before switching. When we are not in an option framework, we may run into durative actions in complex environments. If we expect that a durative action is generally too short for its effects to be felt, we can simply run the action multiple times in a row, growing the action space by slowly lowering the number of times it is run in a row. In a robotic arm, perhaps random exploration will not be effective, but encoding the inductive bias of moving in a certain direction for a long time in the beginning of training may help training converge faster. Running a durative action for some time should make it easier for the agent to learn the reward provided.

4 GAS MODELING FOR REAL-TIME STRATEGY GAMES

4.1 TASK

In this study, we tackle the problem of controlling military units in StarCraft: Brood War. In contrast to previous works, we focus on playing full games rather than isolated micro-management scenarios, in which two sets of units are provided (Synnaeve and Bessiere, 2011; Churchill et al., 2012; Usunier et al., 2016; Farquhar et al., 2019). Besides control of military units, full game-play requires management of worker units for resource gathering as well as ensuring economic progress to enable the production of military units in the first place. In our setup, these tasks are handled by rule sets provided by the StarCraft bot platform *TorchCraftAI* (Gehring et al., 2019; Synnaeve et al., 2016). These include a planning algorithm to realize a fixed unit build order which dictates the high-level strategy over the course of the game. In our training and evaluation setup, we provide identical rule configurations for all agents.

As is common in real-time strategy games, observations in StarCraft are limited to the immediate vicinity of the player's own units. The remaining area on the map is covered by the "fog of war". The observations our agent receives are represented symbolically in a similar fashion as described in Farquhar et al. (2019). The win condition of a game is to destroy all buildings of the opponent. The final reward is 1 for a win, -1 for a loss.

4.2 ACTION SPACE HIERARCHY

We consider the following combinatorial action space, inspired by the primitive actions of the Brood War game engine and the typical human interaction with the game. At each time step, we issue a single command at a single location to a subset of all controllable units. This also corresponds to the action space implemented in *TorchCraftAI* (Gehring et al., 2019). Informally, this action is composed of a selection of a single command, the set of units to give that command to, and the target position of the command. Formally, each action is a 3-tuple $A = \langle A_U, A_P, A_C \rangle$. A_U is itself a combinatorial action with $A_U = \langle A_u \rangle_{u \in U}$ for a set of units U and binary actions $A_u \in \{0, 1\}$ such that $A_u = 1$ means we select the unit. The target position $A_P = \langle x, y \rangle$ of the command corresponds to a pixel coordinate on the map. We support three primitive commands offered by the StarCraft game engine, $A_C \in \{Attack, Move, AttackMove\}$ The actions respectively result in attacking an enemy unit, moving to a target location, and moving to a location while engaging in fights with opponent units encountered along the way. The selected action will be executed by the game engine until completion. Execution is performed on a per-unit basis, i.e. units will execute an old action until a new action is given, allowing the control of multiple groups of units separately despite selecting only one action at each time step. We note that the command A_C and position A_P are directly tied to each other and thus model $\langle \mathcal{A}_P, \mathcal{A}_C \rangle$ as a joint action space $\mathcal{A}_{P,C}$ of dimension $|\mathcal{A}_{P,C}| = |\mathcal{A}_P||\mathcal{A}_C|$.

Based on the natural spatial hierarchy of the game, we propose to grow the action space of both unit selection A_U and command and target position $A_{P,C}$. Trading off action accuracy with ease of modeling, and noting that human players seldomly rely on pixel-perfect control, we map all actions to a fixed grid over the whole playing area. The first and lowest level of actions are *build tiles*, each corresponding to a 32-by-32 pixel region. Denoting with $\mathcal{A}_P^l$ the action set over positions at level l, $\mathcal{A}_P^1$ is thus defined over build tiles. With each successive level, we halve the resolution of positions so that $4|\mathcal{A}_P^l| = |\mathcal{A}_P^{l+1}|$. Consequently, a two-by-two neighborhood of positions in level l is collapsed to a single action in level $l+1$. Selecting a position at level $l+1$ amounts to randomly selecting any of the corresponding positions at level l. This action hierarchy results in joint action spaces $\mathcal{A}_{P,C}^l$.

For unit selection, we express the per-unit actions A_u in build tiles as well. The resulting action space $\mathcal{A}_V$ consists of binary actions A_{v_p} for each build tile position p. If $A_{v_p} = 1$, all units located in the build tile corresponding to p will be included in the resulting action A. Units are often moved in groups; higher-level actions thus bias the binary, per-unit selection based on the distance between them. We realize this by sampling A_{v_p} with a spatially correlated yet uniform noise ϵ_p^l, i.e.

$$A_{v_p} = \begin{cases} 1 & \text{if } \pi_{v_p} > \epsilon_p^l \\ 0 & \text{otherwise} , \end{cases}$$

with π_{v_p} denoting the probability of selecting units at position p according to the current policy π. At the lowest level of actions, ϵ_p^l is generated uniformly $\epsilon_P^1 \sim \mathcal{U}(0,1)$. For $l > 1$, we obtain spatial correlation by smoothing. Given an arbitrary smoothing kernel K_l, we can maintain the uniform noise in the long run by transforming the noise via $\mathrm{cdf}(K_l * \mathrm{icdf}(\epsilon_P^1))$, where icdf and cdf are the (inverse-)cumulative distributions of a Gaussian, and $*$ indicates the convolution operator. This ensures that the expectation of ϵ_p^l is maintained across different levels. For our experiments, we use the $K_l\left(x, y\right) = \exp\left(-\left(x^2 + y^2\right) / \left(2 \cdot \sigma_l^2\right)\right)$ as the smoothing kernel, where σ_l is a characteristic distance expressed in build tiles.

In Figure 1, we visualize our proposed hierarchy on a small sub-section of a 64-by-64 build tile map. The unit selection noise ϵ_u^l, corresponds to random noise at the lowest level and will gradually increase in spatial correlation in higher levels. The distribution over positions is shown for two of the three commands. The decrease in resolution for higher action levels directly results in fewer possible actions at the expense of accuracy. For example, the actions over which $\pi_{P,C}^4(C = Attack)$ is defined make it impossible to select an individual target among the red units on the bridge. The agent can only chose whether to attack the units on the bridge not. On the other hand, it is still possible to decide whether to enter the battle or not.

The temporal dimension of the action space is structured by modifying the time steps between taking actions, denoted with $\delta_t^l = l$. At a coarse level, only few actions can be performed during a game. This provides the agent with an opportunity to experience the effects of long actions such as moving units across long distances. The lowest level, $l = 1$, corresponds to taking an action at every time step.

4.3 Multi-Scale Policy Model

We enable joint training of all action space levels by employing a single neural-network parameterization for all resulting policies π^l. For unit selection, the same policy is used for different action levels, as only the correlation of the noise ϵ_p^l is adjusted. The action space hierarchy over $\mathcal{A}_{P,C}$ is realized by parameterizing $\pi_{P,C}^1$ with neural network weights θ. For additional levels l, we do not introduce extra parameters but instead obtain $\pi_{P,C}^{l+1}$ by average-pooling $\pi_{P,C}^l$. Concretely, for a maximum map size (W, H), $\pi_{P,C}^1$ corresponds to a three-dimensional tensor $|C| \times H \times W$. The remaining distributions for $l > 1$ are thus computed as

$$\pi_{P,C}^{l+1}(C, x^{l+1}, y^{l+1}|s) = \frac{1}{4} \sum_{m=0}^{1} \sum_{n=0}^{1} \pi_{P,C}^l(C, 2x^l + m, 2y^l + n|s) \; .$$

For our parameterization, we opt for an encoder-LSTM-decoder model, where a ResNet (He et al., 2015) trunk is used for encoding as in Farquhar et al. (2019). Symbolic unit observations are first

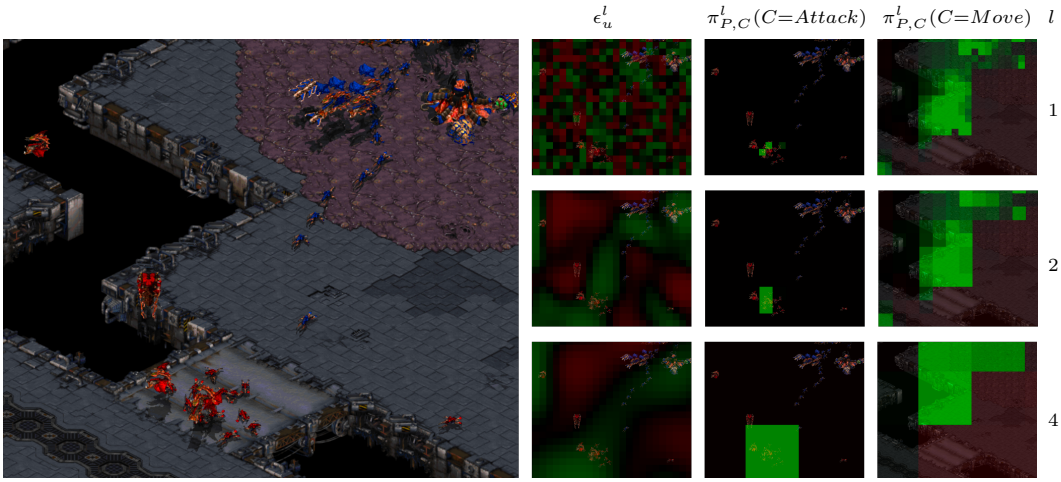


Figure 1: Visualization of spatially correlated noise and action space resolution. **Left** Close-up of a StarCraft: Brood War game with two players (blue and red), showing $\approx 12\%$ of the whole map area). The "fog of war" limiting the blue player's observations is not shown here. **Right** For the agent playing blue, we show (from left to right) the sampling noise for unit selection ϵ_u^l, example distributions over positions for $Attack$ and $Move$ commands at different spatial levels l (top to bottom). Green represents high values, red stands for low values; actual values are scaled for visualization purposes.

encoded individually, with the resulting embeddings then placed at their corresponding locations for concatenation with spatial features. To ease memorization of past observations, a spatially replicated LSTM performs temporal integration of the resulting features (Xingjian et al., 2015; Synnaeve et al., 2018) before the ResNet trunk is applied.

The encoder output is provided to a LSTM (Hochreiter and Schmidhuber, 1997) and decoded using upsample-convolution blocks to its original resolution. Each upsample-convolution block consists of a 2-by-2 upsampling operation and concatenation with the intermediate output of matching resolution of the ResNet trunk, followed by a residual convolution block. We employ two separate decoders for $\pi_{P,C}$ and π_V, such that we can condition the unit decoder on the sampled position and command. We only duplicate the upsample-convolution blocks of the model, and keep the LSTM trunk the same same.

We note that for a given game state, only a subset of all actions are permitted. Attack actions can only be issued to currently visible opponent units, and unit selection can be constrained to locations at which any of the player's units are present. We mask out invalid actions by constraining the respective probabilities in $\pi_{P,C}(C = Attack)$ and π_{v_p} to 0. Similarly, we implement play at coarser time resolutions by masking out all possible actions. For a time step t and temporal action level l, the agent will obtain policy gradients if $t \bmod \delta_t^l = 0$ only. Otherwise, gradients are solely provided by back-propagation through time used for the LSTM cells.

Model training is performed in an actor-critic fashion Sutton and Barto (1998). While the policy is computed on partial observations only, we grant the critic access to full observations. The critic function is parameterized separately from the policy network using the same architecture. The availability of full observations removes most of the need for temporal integration so that we do not include recurrent layers in the critic network. The encoder is followed by a linear layer, predicting the state-value function V^π.

4.4 Growing Schedules

Over the course of training, we aim to successively decrease the action level l to obtain finer-grained policies in a self-play setting of several agents. With the premise that finer actions enable better game-play, we expect that an appropriately timed action space refinement for a single agent results in increased performance over other agents. The common approach of hand-crafting a growing schedule comes with the pitfall that optimal training might require non-monotonic schedules. This is amplified

by the fact that we jointly grow three dimensions of the action space (positions, correlation of unit selection, and the time span between actions). For example, deciding whether to flee a battle requires frequent actions but only a coarse target position. In contrast, tactical positioning is possible on larger time spans, but can require very specific selection of position and units. We thus avoid the manual definition of schedules and instead introduce a refinement of the action space as a possible mutation operation in population based training Jaderberg et al. (2017).

The single parameterization of policies detailed in §4.3 enables the use of *smooth* growing schedules, in which multiple action levels can be utilized. To this end, we specify a distribution over action levels, with growing corresponding to a shift of the probability mass towards lower levels. The final resolution then corresponds to a distribution with $Pr(l = 1) = 1$ and 0 for all $l > 1$. At each time step, a level is first sampled according to the current distribution, and the action will then be sampled from the respective policy. We apply a smooth growing schedule to the position action only and specify single levels for unit selection and the interval between taking actions.

5 EXPERIMENTS

Scenarios We selected three different scenarios to run experiments and ablations and investigate the effect of our proposed action space curricula on training speed and final performance. A *toy micro* experiment allows us explore the effect of action space growing in an isolated setting. The second is a *mid-game* scenario, in which players start out with 2 bases as well as military units and defensive buildings. This scenario trains much faster and allows us to study our training dynamics in more detail. Finally, we play *full* games where both players start out as in a standard game of StarCraft.

Fixed Baselines In order to measure progress, we ground the evaluation of all trained models by competing against handicapped versions of the rule-based StarCraft bot *CherryPi*, that is not learning anything. *CherryPi* is a top StarCraft bot: it won the SSCAIT 2017/2018 tournament and came second at the AIIDE 2018 competition. We note *CPI* for the full bot with no handicaps; *CPIIdle* is handicapped by not taking any actions for the first 40 seconds of game time. *CPIRestrict50* and *CPIRestrict25* are versions of the bot that randomly drop 50% and 25% of their actions, respectively. Our trainable policy is identical to those baselines for the economy and worker management.

Training Setup We train a population of several agents in parallel Jaderberg et al. (2017) where each agent is trained with IMPALA Espeholt et al. (2018). After a random pairing for matchups, a game trace is collected by playing another agent in the population; multiple games are played in parallel in a pool of rollout workers shared among the whole population. For each agent, traces are then aggregated in a FIFO queue of maximum size 2500 – a replay buffer – where each entry is a BPTT-sized (back-propagation through time) chunk, including LSTM state burn-in (Kapturowski et al., 2019). Chunks are sampled with prioritized experience replay (Schaul et al., 2015), and gradients are then computed with the V-Trace off-policy correction (Espeholt et al., 2018).

After a fixed number of model updates (between 500 and 1000 in our experiments), agents produce individual checkpoints of their parameters. Similar to the framework described in Li et al. (2019), the resulting policy enters an evaluation round with past population members. We compute a ranking using TrueSkill (Herbrich et al., 2007), and the rank of the checkpointed policy determines if the agent gets culled. When a training agent is interrupted, a new agent is initialized from a previous population member, with hyper-parameters subject to mutation (all our experiments use a mutation rate of 50%). Parent selection is done proportionally to the rankings in the most recent tournament. We limit the number of archived agents – non-training population members – and remove the ones with the worst rank. We detail the PBT hyper-parameters in Table 1, the learning rate and the free GAS hyperparameters are candidates for mutation in all our experiments.

Metrics While our PBT setup optimizes for the TrueSkill of a population, we can measure and compare the success of training runs with additional metrics. First, we compare against fixed baselines, which do not take part in training games, both in terms of win rates and TrueSkill rating. We further let members of different populations directly compete against themselves and analyze the resulting win rates.

Parameter	Toy micro	Late Game	Full Game
Active population size	2	8	8
Maximum archived agents	2	16	64
Batch size	120	56	120
BPTT steps	16	48	24
Checkpoint interval (updates)	500	1000	800
Entropy factor (fixed/mutated)	10^{-4}	Mutated	10^{-4}
Ranking against *CPI* versions	No	Yes	No
Reward	Win/lose	Win/lose + Δsupply	Win/lose

Table 1: Hyper-parameters used during training for each of the three scenarios discussed in §5.

5.1 TOY TASK

In this task, the agent starts like in a normal game, in a map with 4 starting locations, and the game ends when one player discovers the other player's base. Players can split up their initial set of units to try to find which of the remaining 3 starting locations contains the enemy. Intuitively, randomly exploring at a high spatiotemporal frequency is harder than exploring at a lower spatiotemporal frequency, and splitting units should be beneficial. Therefore, growing all three of U, P, and T resolutions should allow the model to explore much faster.

For both tasks, after we finish training, we pit the agents against each other and show the winrates in Figure 2. The best agent in the GAS run was able to discover that quickly moving the initial single unit ("Overlord") towards the closest enemy base candidate location was a good strategy. Because this unit is very slow, and rarely reaches the base before other units spawn, it shows that the model is able to pick up a very long-horizon reward signal. Additionally, the model will move the faster units produced to a different base than the initial overlord. Figure 2 indicates that growing all three is most effective on this task.

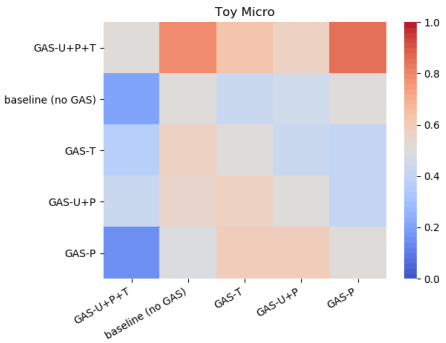


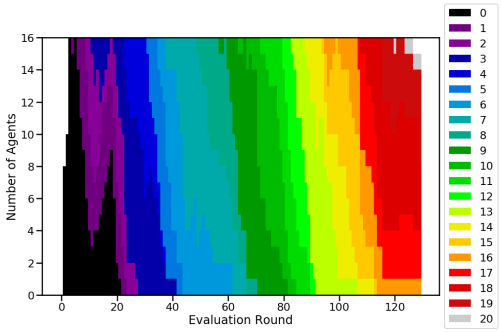
Figure 2: The *toy micro* task, where the goal is to find the opponent base. We play 30 games between each agent after training for one day, and display the winrate in a head-to head matchup, e.g. first to find enemy base. The plot shows that we need all three growing schemes to succeed on this task.

5.2 MID-GAME SCENARIO

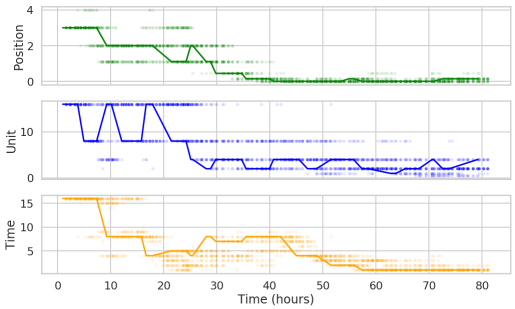
We propose a mid-game scenario to ablate our experiments. We use a small map with 4 bases, and begin with each player having a few military units and in control of two of the bases. In order to compare the effects of our GAS scheme and population based-training we run: (1) a baseline with no GAS, (2) Growing only temporally, (3) Growing all three axes. To further speed up training, we rank against *CPI*, as well as the restricted versions, during PBT. Even though we do not train against them, PBT will select agents that do well, giving a smoother learning curve for our experiments.

Over the course of the training, agents trained with GAS learn different strategies as the overall level of the population increases. First, agents learn to gather their military units in one base, and wait for the enemy to attack. Then, they value the center of the map and fight over it to prevent the enemy from gathering all its units together. Later on, they learn that they should retreat their units to defend when their base is under attack, or sometimes choose to trade bases if they are already attacking an enemy base. Finally, they learn to refine their unit selection to handle different unit types differently.

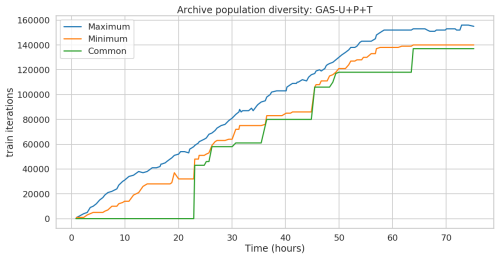
After training for 80 hours, we compare the results of the ablations in 3: trainings without GAS for U, P, and T plateau after about 40 hours of training to a local optimum. The policies trained on those runs don't use the "Move" command and almost exclusively rely on "Attack-Move" commands. While "Move" orders are essential to retreat or flee, they are almost impossible to learn without a



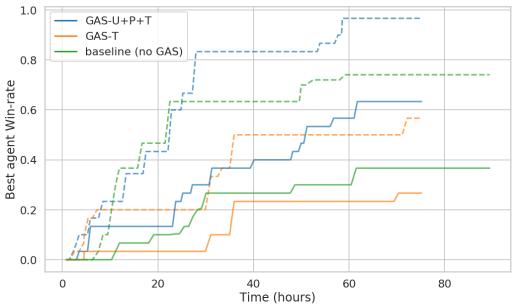
(a) Distribution of the different agent generations in the archived population (training with GAS).



(b) P (green), U (blue) and T (yellow) resolutions schedules found by PBT: dots represent agents in the population, and the line is the GAS schedule for the lineage of the best agent in the final evaluation round



(c) Population diversity for GAS training: minimum and maximum number of training iterations for agents the population, as well as training iterations of their common ancestry.



(d) Best win-rate obtained up to different wall clock times against *CPI* (and *CPIRestrict25* in dashed lines)

Figure 3: Mid-game scenario

proper exploration scheme. Indeed, for a retreat action to be successful, the policy has to select an entire group of units (retreating half of the army causes the loss of the other half), and commit to this action for a few seconds until the group reaches a safe place.

On the population trained with GAS, we don't observe any such plateau. At a given time, different agents are training at different levels of detail and exploring a diverse set of strategies (3b). For Time resolution for instance, progress is not monotonic: even after reaching the finest resolution, it still makes sense to explore at a coarser resolution to discover new strategies, before refining the strategy at the finest level, as shown by the level of actions schedule of the best final agent. Periods of exploration and exploitation alternate at the scale of the population as well. We report in 3c the minimum and maximum number of training iterations for archived models, and compare it to the number of training iterations for the most recent common ancestor of the archived population. When the gap with the common ancestor increases, so does the overall diversity of the policies. When a new and significantly stronger policy is discovered, it gradually fills the archive slots, trading-off diversity for performance.

5.3 Full Games

Finally, we run GAS vs. baseline (no GAS) experiments on the full game setting. Games are played on a single two-player map, and we observe that we are able to learn unit control from scratch. We observe the same strategies as in the mid-game scenario, of controlling the center of the map and engaging in base trades. In this experiment we do not use any reward shaping nor do we include the fixed baselines in our evaluation to rank the population (see Table 1), we still manage to obtain an above 50% winrate vs. *CPI* in about 50 hours of training (see Figure 4). Though sometimes there are plateaus, the winrate consistently improves.

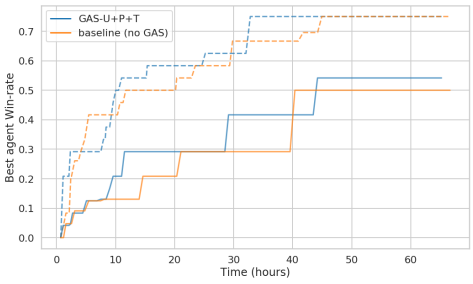


Figure 4: Win rate of the best so far agent vs. *CPIRestrict25* (dashed lines) and *CPI* for different ablations in full games.

GAS Agent	Win Rate against Agent without GAS (%) 1h	5h	10h	20h	40h	60h
1h	63.9	42.0	36.7	33.3	31.7	26.5
5h	73.3	60.0	23.3	40.0	23.3	57.1
10h	76.7	76.7	66.7	63.3	46.7	38.5
20h	70.0	56.7	73.3	63.3	50.0	30.0
40h	90.2	96.7	40.0	83.3	70.0	56.2
60h	69.6	86.7	78.9	80.0	72.2	66.7

Table 2: Win rates obtained in a tournament of the best agents obtained from two separate training runs after a specific amount wall clock time, in the full games setting.

In Table 2, we extract the best GAS and baseline agent during different times in the training, and play them against each other in a tournament with 30 games per matchup. The difference is even bigger in this case, it took the baseline 40 hours of training before beating the best GAS agent at 10 hours of training. We suspect GAS allows agents to learn more robust and more diverse policies, instead of exploiting quirks of the environment, since the action space changes during the training.

6 Conclusion

In this work, we proposed methods for successively growing the action space of reinforcement learning agents, and demonstrate it in the context of a challenging full game unit control task. Our policy architecture and sampling scheme enable a hierarchy over actions without introducing additional parameters. Consequently, experience obtained with coarse high-level actions directly helps training low-level actions. Experiments on StarCraft: Brood War demonstrate that agents trained while growing the action space make faster progress and enjoy superior exploration. This is supported quantitatively by comparing win rates and rankings against fixed baseline agents and between populations; and qualitatively by analyzing the exhibited strategies. We also find that population based training is an efficient way to discover a suitable growing schedule. Combined with self-play, GAS enables us to beat a strong rule-based baseline in mid-game scenarios.

References

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

David Churchill, Abdallah Saffidine, and Michael Buro. Fast heuristic search for rts game combat scenarios. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'12, pages 112–117. AAAI Press, 2012. URL `http://dl.acm.org/citation.cfm?id=3014629.3014650`.

Wojciech Marian Czarnecki, Siddhant M Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Simon Osindero, Nicolas Heess, and Razvan Pascanu. Mix&match-agent curricula for reinforcement learning. *arXiv preprint arXiv:1806.01780*, 2018.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Gregory Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. Growing action spaces. *arXiv preprint arXiv:1906.12266*, 2019.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

Jonas Gehring, Zeming Lin, Daniel Haziza, Vegard Mella, Daniel Gant, Nicolas Carion, Dexter Ju, Danielle Rothermel, Laura Gustafson, Eugene Kharitonov, Nicolas Usunier, and Gabriel Synnaeve. TorchCraftAI v1.1, July 2019. URL `https://doi.org/10.5281/zenodo.3341787`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill^{TM}: a bayesian skill rating system. In *Advances in neural information processing systems*, pages 569–576, 2007.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281*, 2018.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Y. LeCun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. Advances in neural information processing systems 2. chapter Handwritten Digit Recognition with a Back-propagation Network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-100-7. URL `http://dl.acm.org/citation.cfm?id=109230.109279`.

Ang Li, Ola Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, 2019.

Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

Andrew W Moore. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In *Advances in neural information processing systems*, pages 711–718, 1994.

Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine learning*, 49 (2-3):291–323, 2002.

Adithyavairavan Murali, Lerrel Pinto, Dhiraj Gandhi, and Abhinav Gupta. Cassl: Curriculum accelerated self-supervised learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6453–6460. IEEE, 2018.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Ali Nouri and Michael L Littman. Multi-resolution exploration in continuous spaces. In *Advances in neural information processing systems*, pages 1209–1216, 2009.

Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959.

Tuomas Sandholm. Abstraction for solving large incomplete-information games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 4127–4131. AAAI Press, 2015. ISBN 0-262-51129-0. URL http://dl.acm.org/citation.cfm?id=2888116.2888295.

Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Oliver G Selfridge, Richard S Sutton, and Andrew G Barto. Training and tracking in robotics. In *IJCAI*, pages 670–672, 1985.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Gabriel Synnaeve and Pierre Bessiere. A bayesian model for rts units control applied to starcraft. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 190–196. IEEE, 2011.

Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.

Gabriel Synnaeve, Zeming Lin, Jonas Gehring, Dan Gant, Vegard Mella, Vasil Khalidov, Nicolas Carion, and Nicolas Usunier. Forward modeling for partial observation strategy games-a starcraft defogger. In *Advances in Neural Information Processing Systems*, pages 10738–10748, 2018.

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct 2012. doi: 10.1109/IROS.2012.6386109.

Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.