# Playing StarCraft II with Curriculum Transfer Learning

**Jahnvi Patel** CS15B046

**Narayan S** EE15B108

## Abstract

Deep reinforcement learning and especially the Asynchronous Advantage Actor-Critic algorithm, has been successfully used to achieve super-human performance in a variety of games. StarCraft II is a challenging domain for Reinforcement Learning making it very difficult to even reach human level performance. In this entire project, we tried to understand the complexities of this environment and discuss the results from our experiments on the environment. We have compared two architectures and have shown that transfer learning can be an effective paradigm in this complex scenarios requiring skill transfer. Next, we also analyze some issues with the conventional transfer approach and present a few solutions to handle them.

## 1 Background

### 1.1 The SC2LE Environment

The StarCraft II API allows programmatic control of StarCraft II. It can be used to start a game, get observations, take actions, and review replays. PySc2 is a Python wrapper over the API that defines an action and observation specification and allows easy visualization of what the agent is doing over a set of feature maps. SC2LE models actions similar to how a user would play, for instance, instead of sending individual commands to units directly, the agent has to select the unit first by drawing a rectangle around it, and then issue commands to the selected unit. This makes the action space more complicated, but more akin to how a human interacts with the game.

The environment provides a run-loop in which the agent can perform an action in lockstep with the game simulation, so that the environment can be configured for the agent to act every n (=16 in our case) game steps. This can be done by setting `steps` parameter while running the code. The action space is modeled to be similar to a human interface. The format of the action is a compound action consisting of a base action $a_0$ and a variable number of arguments $a_1, ...., a_l$ each of which have variable dimensions. An example action that moves a selected unit is `Move_screen` which takes a boolean argument "queued [2]" which determines if this command should be enqueued instead of performed instantly, and "screen [84,84]" which takes two numbers that represent a point on the screen. The agent is provided the valid actions at any time step.

One more addition of this API is that rather than taking raw RGB pixels as input to the agent, it provides a set of "feature layers" each of which corresponds to something specific in the game, for example: unit type, hit points, etc. The non-spatial obervations correspond to those which are not specific to any region of state space, for instance the amount of gas and minerals collected, the set of actions currently available which depends on game context, etc.

The spatial features consist of two sets of feature layers, minimap and screen. Minimap corresponds to a downsized view of the entire map and screen is the primary view of some subsection of the map, depending on where the camera is. The screen feature is a [n,n,13] tensor. It can be set using `sz` parameter in the code. Similarly, the minimap resolution can also be set.

The SC2LE feature layers on visualization look like :


### 1.2 Mini-games

Mini-games provide a way to isolate and learn small essential components of the game by designing a clear reward structure and restricting the state space and possible actions. We consider learning on a set of seven mini-games provided by DeepMing and consider additional mini-games available online like TheRightPath.

1. **MoveToBeacon** : Move a marine to the beacon.
2. **CollectMineralShards** : 2 Marines whose task is to collect the Mineral Shards with optimal collection requiring both Marine units to be split up and moved independently.
3. **TheRightPath**: Move to beacon finding the optimal route collecting minerals and avoiding mines.
4. **FindAndDefeatZerglings** : 3 marines are used to defeat endless supply of Zerglings, requires efficient exploration and combat.
5. **DefeatRoaches** : To learn combat strategies for focus fire on Roaches.
6. **DefeatZerglingsAndBanelings** : 9 Marines fight against a group of 6 Zerglings and 4 Banelings.
7. **CollectMineralsAndGas** : Learn to expand capacity to gather Minerals and Vespene Gas.
8. **BuildMarines** : SCVs collect minerals to build supply depots which will be used to build marines.

Several other mini-games of varying difficulties are also available. Note that each of the maps operate on a single screen and have a resolution of $64 \times 64$. The best current scores for each of these mini-games can be obtained from the leaderboard on `starcraftgym.com`.

## 2 Problem Statement

**Defining mini-games and using them as curricular tasks to learn StarCraft Micro-Management**

The broad goals of our project are as follows :

- Implement an agent that uses a generalized action/feature space across tasks on PySc2 and attempt to replicate DeepMind baselines.

- Determine what network architectures will allow the agent to converge quickly and consistently. Additionally, we considered the trade-off between convergence speed and whether the agent has converged on a sub-optimal policy.
- Train on simpler mini-games like FindAndDefeatZerglings or MoveToBeacon and use the trained model to reduce sample efficiency/improve performance on harder tasks like DefeatZerglingsAndBanelings or TheRightPath.
- To understand improvements in learning can be achieved by training an agent in one scenario which was pre-trained on a different scenario
- To reduce negative transfer across mini-games by evaluating different ways to perform transfer.
- Simplify future development so that a new model requires just a minor change to be made.

# 3 Solution Approach / Results

## 3.1 Choosing the right mini-games

We group the available mini-games provided along with pysc2 as well as those available online according to the domain knowledge available and choose the best possible way to transfer across these tasks. We can also define new mini-games using PySc2 map maker but we have deferred this part as we do not have a reliable way to obtain human baselines for new maps.

## 3.2 Implementing a generalized agent

### 3.2.1 A2C Agent

We use an Advantage Actor Critic (A2C), a policy gradient approach with the gradient definition as:

$$\underbrace{(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{policy gradient}} + \beta \underbrace{(G_t - v_\theta(s_t))\nabla_\theta v_\theta(s_t)}_{\text{value estimation gradient}} + \eta \underbrace{\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)}_{\text{entropy regularization}},$$

where $v_\theta(s)$ is the value function estimate and $G_t$ is the n-step return. The last term acts as entropy regularizer and promotes exploration.

In order to avoid the complex joint distribution over a for defining $\pi(a|s)$ [Note that actions as described earlier include other parameters like coordinates on screen], we transform the problem of choosing an action $a$ to a problem of making a set of decisions as follows:

$$\pi(a|s) = \prod_{l=0}^{L} \pi(a^l|a^{<l}, s).$$

The ordering can be choosing the function identifier first, followed by categorical variables and pixel coordinates.

There are several challenges here. Most A3C agents in Atari games use either spatial features like images or human crafted features. However, the architectures here need to handle both. Spatial actions require a pixel on the game map where the action should be taken for which there are again over 7,000 underlying actions. We next describe the two architectures that we tested:

### 3.2.2 AtariNet

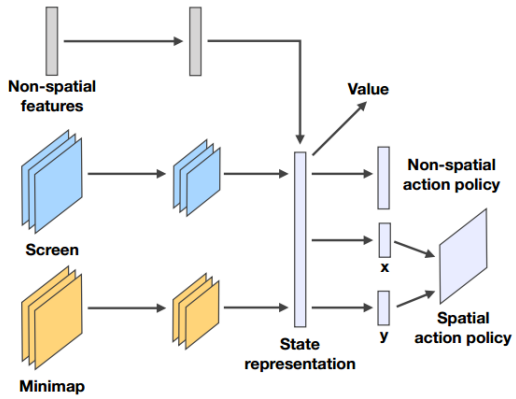
The non-spatial features vector is processed by a tanh activation layer. The screen and minimap feature vectors are passed throught a CNN and the results are concatenated through a ReLU layer. Finally,

| Parameter | Value |
|---|---|
| Learning Rate | 7E-4 |
| Discount Factor | 0.99 |
| Screen resolution | 32 x 32 |
| Minimap resolution | 32 x 32 |
| Frequency of agent's action | 1 per 16 steps of sc2bot |
| Exploration strategy | Epsilon greedy |

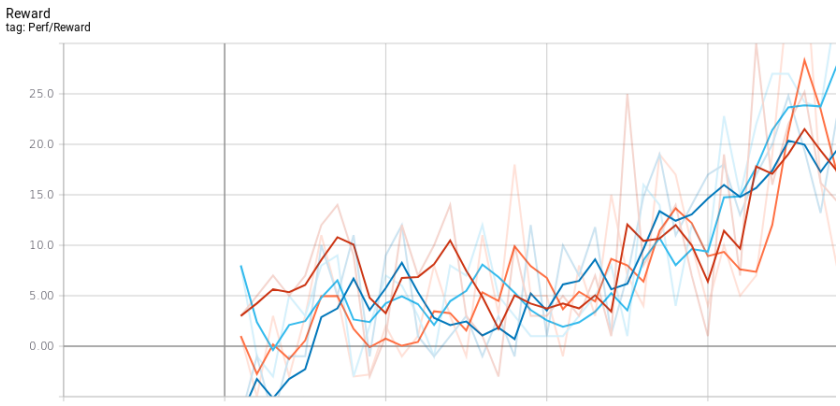
Table 1: Hyper-parameters used

the resulting vector is then used as input to linear layers that output policies over function identifier and arguments independently. The policies here are modeled separately for x and y coordinates.

The hyper-parameters used for these are as follows:




AtariNet reached the expected reward of 26 as expected for simpler games like MoveToBeacon as shown below. However, it could not perform well for the harder tasks.

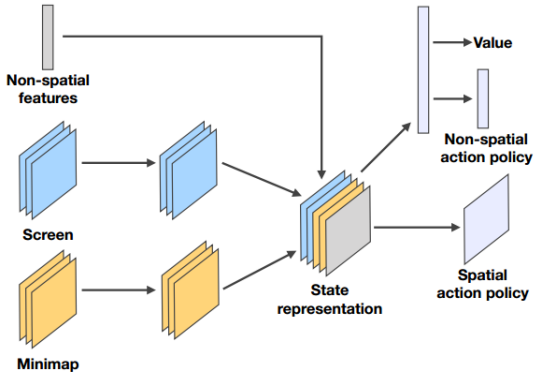



### 3.2.3 FullyConv Network

CNNs usually reduce spatial resolution of the input with each layer and ultimately finish with a fully connected layer that discards it completely. However, in StarCraft, the spatial actions act within the same space as inputs so we want the convolutions to be resolution-preserving. This is done by using padding and zero stride at each layer. The state representation is formed by the concatenation of the screen and minimap network outputs along with non-spatial features. We use $1 \times 1$ convolution to obtain spatial policies whereas non-spatial policies are obtained after passing through FC layer with ReLU activation. This gave significantly better performance than AtariNet.

| Mini Game | A2C (ours) | Human Player | DeepMind |
|---|---|---|---|
| MoveToBeacon | 25 | 26 | 26 |
| FindAndDefeatZerglings | 33 | 46 | 45 |
| DefeatZerglingsAndBanelings | 90 | 729 | 62 |

Table 2: Comparison of scores between a human player, DeepMind baselines and our baseline




(We are yet to figure out how we got better performance than DeepMind!)

### 3.3 Curriculum Transfer Learning

Now to improve the performance further, we tried to transfer from simpler tasks like FindAndDefeatZerglings which involves stationary Zerglings as enemies to DefeatZerglingsAndBanelings involving a set of 6 Zerglings and 4 Banelings that can attack our marines directly.

We do so by using a pretrained model and use it to learn the harder mini-game. However, one essential problem that we encountered was negative transfer, for instance, in DefeatZerglingsAndBanelings, we started with a score of 95 which kept deteriorating with time as the model started overfitting.

One way that we tried is to prune the models by returning to a good checkpoint and retraining which however, does not always help. Another way that we re-evaluated the tasks from which we were transferring. We did not get any significant improvements in ti

## 4 Future Directions

We are yet to finish complete implementation of A2T for the given architectures. A2T would help solve the issue of negative transfer while learning on a different task. However, this does not solve the micromanagement scenario completely. Forgetting in neural networks is one issue that we would need to target if we want a single model to solve micromanagement.

## References

[1] StarCraft II: A New Challenge for Reinforcement Learning

[2] StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning

[3] Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain

[4] Automated Curriculum Learning by Rewarding Temporally Rare Events