Using NEAT to Learn Operators for Flexible Boolean Composition within Reinforcement Learning

Amir Esterhuysen
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
amiresterhuysen@gmail.com

Steven James
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
steven.james@wits.ac.za

Geraud Nangue Tasse
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
geraud.nanguetasse1@wits.ac.za

Benjamin Rosman
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
benjamin.rosman1@wits.ac.za

Jonathan Shock
Department of Mathematics
University of Cape Town
Cape Town, South Africa
jonathan.shock@uct.ac.za

Abstract

Skill composition is a growing area of interest within Reinforcement Learning (RL) research. For example, if designing a robot for household assistance, we can not feasibly train it for every task it might face during a period of months or years. Instead, it could benefit from having a set of broadly useful skills and the ability to adapt or combine these skills to deal with specific situations. This approach mimics the way in which humans actually learn—by gradually expanding their knowledge base of basic skills which can be combined in endless ways, instead of independently learning new abilities from scratch each time a novel problem arises. Existing work has demonstrated how simple skills can be composed using Boolean operators to solve new, unseen tasks without further learning. However, this approach assumes that the learned value functions for each atomic skill are optimal, an assumption which is violated in most practical cases. We propose a method that instead *learns* operators for composition using evolutionary strategies. We empirically verify our approach in tabular and high-dimensional environments. Results demonstrate that our approach outperforms existing composition methods when faced with learned, suboptimal behaviours, while also promoting robust agents and allowing for transfer between domains.

Keywords: Neuroevolution, NEAT, Reinforcement Learning, Composition

1 Introduction

The development of versatile agents capable of adapting to new contexts and challenges is a critical goal across artificial intelligence and particularly in RL. One approach is to compose learned tasks via the fundamental Boolean operations—disjunction (OR), conjunction (AND), negation (NOT)—with a fixed, provably optimal set of composition operators [2].

Tasks are modelled as a Markov Decision Process (MDP), given by a tuple of the form $(\mathcal{S}, \mathcal{A}, \rho, r, \gamma)$, where (i) $\mathcal{S}$ is the state space, (ii) $\mathcal{A}$ is the action space, (iii) $\rho(s'|s, a)$ governs transitions between states in $\mathcal{S}$ after taking some action in $\mathcal{A}$, (iv) $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and (v) $\gamma \in [0, 1]$ is a discount factor.

Consider two tasks with optimal *world value functions* (an extended value function that carries knowledge of all goals in an environment [3]) given by $\hat{Q}_1^*$ and $\hat{Q}_2^*$. Let $\hat{Q}_{min}^*, \hat{Q}_{max}^*$ be respectively the optimal world value functions in the case where all goals in the environment are marked as undesirable and desirable. Then for $s \in \mathcal{S}, g \in \mathcal{G} \subseteq \mathcal{S}$, and $a \in \mathcal{A}$, tasks are composed as follows:

- *disjunction*: $(\hat{Q}_1^* \vee \hat{Q}_2^*)(s, g, a) = \max\{\hat{Q}_1^*(s, g, a), \hat{Q}_2^*(s, g, a)\}$
- *conjunction*: $(\hat{Q}_1^* \wedge \hat{Q}_2^*)(s, g, a) = \min\{\hat{Q}_1^*(s, g, a), \hat{Q}_2^*(s, g, a)\}$
- *negation*: $(\neg\hat{Q}_1^*)(s, g, a) = (\hat{Q}_{min}^*(s, g, a) + \hat{Q}_{max}^*(s, g, a)) - \hat{Q}_1^*(s, g, a)$

This approach, however, only guarantees success when an agent has optimal knowledge of the behaviours being composed—a requirement that restricts the viability of Boolean composition methods in real-world problems. We thus propose that the agent learns how to compose suboptimal behaviours. We adapt the NEAT algorithm (Neuroevolution of Augmenting Topologies) [4] to learn an alternative set of robust composition operators which outperform the fixed Boolean algebra approach [2] when faced with challenges that often manifest in practical RL environments. Our approach is shown to be viable first in a tabular setting. We are then able to learn operators in the tabular setting which can be applied with positive results in the function approximation setting—despite these environments differing in scale and state-space. This is particularly encouraging for the goal of producing robust RL agents.

2 Learning Composition Operators

NEAT uses a genetic algorithm to evolve neural network solutions [4]. This simultaneously optimises network weights and the overall network topology, exploring a much larger space of candidate networks compared to standard neural network training algorithms like backpropagation. Our goal is to produce alternative operators $f_\vee, f_\wedge, f_\neg$—each a neural network—which outperform their Boolean algebra equivalents when given suboptimal input. Algorithm 1 describes how we use the underlying principles of NEAT to learn neural networks which act as robust composition operators for building RL skills. The evolutionary process is guided by a fitness function—by default this is the total reward achieved over a number of evaluation episodes. Code for the base NEAT implementation is found in the NEAT-Python library [1].

2.1 Differentiable Approximations

We briefly consider a middle ground between these learned functions and the Boolean set of fixed operators. The operators for optimal disjunction and conjunction are respectively the *max* and *min* functions. It is reasonable to hypothesise that even when these operators are insufficient, something similar might perform well. We examine three differentiable functions which approximate the *min* and *max* (note there is no analogue for the negation operator). Let $\hat{Q}_1$ and $\hat{Q}_2$ be world value functions in this environment. For a given triplet $(s, g, a) \in \mathcal{S} \times \mathcal{G} \times \mathcal{A}$, let $q_1 = \hat{Q}_1(s, g, a)$ and $q_2 = \hat{Q}_2(s, g, a)$.

Then,

- $\mathit{Boltzmann}(q_1, q_2) = \dfrac{q_1 \cdot e^{\alpha \cdot q_1} + q_2 \cdot e^{\alpha \cdot q_2}}{e^{\alpha \cdot q_1} + e^{\alpha \cdot q_2}}$
- $\mathit{LogSumExp}(q_1, q_2) = \dfrac{1}{\alpha} \log\left(e^{\alpha \cdot q_1} + e^{\alpha \cdot q_2}\right)$
- $\mathit{MellowMax}(q_1, q_2) = \dfrac{1}{\alpha} \log\left(\dfrac{e^{\alpha \cdot q_1}}{2} + \dfrac{e^{\alpha \cdot q_2}}{2}\right)$

If we let *approx* signify any one of these functions, we have that $\lim_{\alpha \to -\infty} \mathit{approx}(q_1, q_2) = \mathit{min}(q_1, q_2)$ and $\lim_{\alpha \to \infty} \mathit{approx}(q_1, q_2) = \mathit{max}(q_1, q_2)$. The parameter α is optimised by iterating through potential values.

Algorithm 1 Learning Composition Operators

Inputs:
Desired Composition type (OR/AND/NOT), set of base task world value functions
$\hat{\mathbf{Q}} = \{\hat{Q}_1 \dots \hat{Q}_n\}$, where $\hat{Q}_i : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ for all $i = 1 \dots n$.
Initial Population Size *Num_Pop*
Number of generations *Num_Gen*
Number of evaluation episodes *Num_Eval*
Initialize:
population of *Num_Pop* networks
for *generation* = 1 to *Num_Gen* **do**
  for $f \in$ *population* **do**
    $fitness \leftarrow 0$
    for *evaluation episode* = 1 to *Num_Eval* **do**
      Randomly generate specific desired composition within type (eg `RED OR BLUE`)
      Fetch relevant value functions $\hat{Q}_{base} \subset \hat{\mathbf{Q}}$
      while *episode not terminated* **do**
        $\hat{Q}_{composed} \leftarrow f(\hat{Q}_{base})$
        Randomly generate starting state s for agent
        Select action $a = \arg\max_{a \in \mathcal{A}} \hat{Q}_{composed}(s, g, a)$
        Execute action a to collect reward r and reach state s'
        $fitness \leftarrow fitness + r$
      end while
    end for
    Assign final $fitness$ to f
  end for
  Arrange all $f \in$ *population* by $fitness$
  Discard networks $f \in$ *population* with lowest $fitness$
  Choose networks $f \in$ *population* with highest $fitness$ to act as *parents*
  Mate *parents* together to produce *children* which are added to *population*
  Randomly mutate *children* to add variety to *population*
  Update *population* going into next *generation*
end for
  return $f_{best} = \{f_1 \dots f_n\}$, the n highest-fitness networks produced across entire run

3 Tabular Experiments

The simplest setting we consider is a 4 goal 9×9 Four Rooms domain with deterministic transitions. Base tasks are pairwise disjunctions between goals—for example, the base tasks `(`G_1 `OR` G_2`)` and `(`G_2 `OR` G_3`)` are used to define the composed task `(`G_1 `OR` G_2`) AND (`G_2 `OR` G_3`)` = G_2.

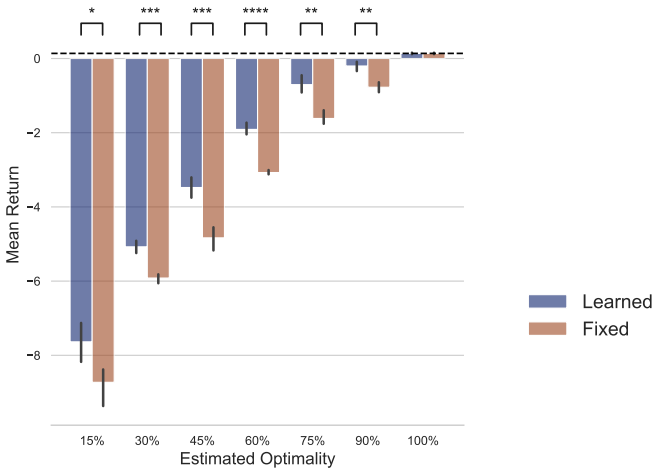


Figure 1: NOT performance for value functions of increasing optimality. Error bars represent 95% confidence interval over 5 runs. The dotted horizontal line represents theoretically optimal performance. A significance marker of * indicates a p-value ≤ 0.05, ** is a p-value ≤ 0.01, *** is a p-value ≤ 0.001, and **** is a p-value ≤ 0.0001

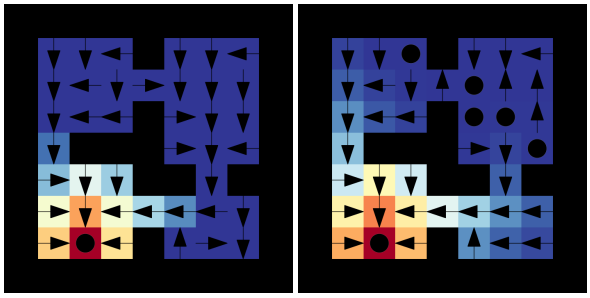
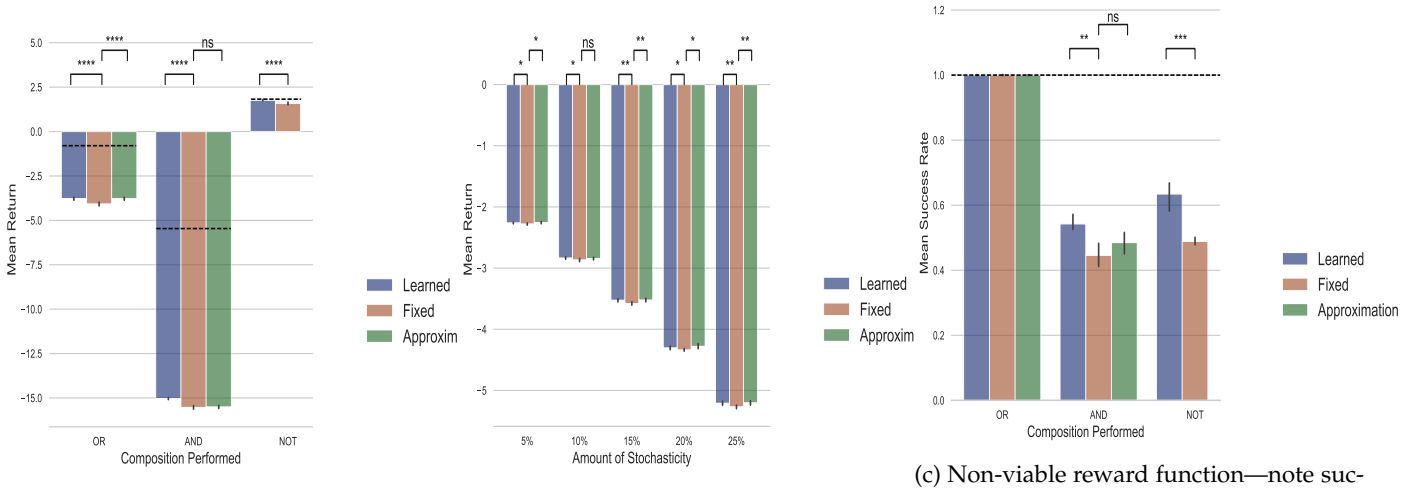


Figure 2: Policies for a specific conjunction task in the case of learned (left) and fixed (right) operators. Circles indicate the agent choosing to stay in its current state. The goal is shown by the red cell in the bottom left of the grid.

Figure 1 shows returns across value functions of increasing optimality (estimated through convergence patterns during training) when performing negation. Learned operators exhibit superior performance to their fixed Boolean equivalents for all suboptimal value functions, and in the case of optimal value functions we match fixed operator performance. Figure 2 depicts how a resultant policy may differ based on the composition operator. Using a learned conjunction operator, the agent gets stuck at 3 states in the bottom right of the grid, while being able to achieve the desired goal from all other states. With the fixed Boolean *min* operator, the agent gets stuck at 12 different states—clearly learned operators can achieve tangible improvements under suboptimal conditions.

We next consider modifications beyond standard suboptimality, depicted in Figure 3. First, the case of a larger 13×13 Four Rooms with 12 goals. This change leads to the number of base tasks growing from 6 to 66. Value functions are estimated to be 50% optimal. Learned operators achieve a significant advantage across all 3 operations. Second, when stochastic transition dynamics are incorporated into the original Four Rooms setting. A stochasticity level of $p \in (0, 1)$ means an agent will move in its desired direction with probability $1 - p$, and has equal probability $p/3$ of moving in each of the remaining directions. Value functions are trained to optimality in this case, with the presence of randomness disrupting the Boolean algebra framework. Here we see significant advantages for learned operators across the spectrum of stochasticity when performing conjunction, with differentiable approximations remaining competitive. Finally, we consider a non-viable reward function that allows us to learn the base tasks but which leads to undesired behaviour when performing conjunction and negation on these base tasks. To circumvent this issue, we optimise success rate of achieving the correct goal—not reward. Again, optimal value functions are used and we see learned operators with an advantage. For disjunction, optimal behaviour is not disrupted by the limitations of the reward function.



(a) Larger environment

(b) Stochastic transitions

(c) Non-viable reward function—note success rate on the y-axis.

Figure 3: Error bars represent 95% confidence interval taken over 10 runs. A significance marker of * indicates a p-value ≤ 0.05, ** is a p-value ≤ 0.01, *** is a p-value ≤ 0.001, and **** is a p-value ≤ 0.0001. ns indicates the discrepancy is not significant.

4 Transfer Between Tabular and Function Approximation Settings

Fundamentally, we wish to promote behaviour that is robust and adaptable. It is thus desirable that we can take operators learned in a source tabular setting and apply them in a target higher-dimensional setting—with distinct state-space representation. Harnessing existing knowledge in new and more complex domains is an important step for the generalising skills of an RL agent, while also allowing for more efficient resource usage—especially as running NEAT in this higher-dimensional setting is significantly more compute-intensive. We consider two cases: i) learning operators in the source domain before zero-shot transferring to the target domain, and ii) learning operators in the source domain before fine-tuning them in the target domain. In this study, Four Rooms acts as our smaller source domain and an environment called Boxman is the more complex target domain—here the state-space is described a higher-dimensional vector of 84×84 RGB images and agents traverse around obstacles to collect desired objects, identified by 6 colour-shape combinations (e.g. "blue circle").

Figure 4 compares four approaches when testing conjunction (AND) in the Boxman domain:

1. Learn AND operator fully in Four Rooms.
2. Learn AND operator fully in Boxman.

3. Learn AND operator in Four Rooms, then fine-tune it in Boxman with a scaled-down NEAT run (fewer generations/population members/evaluation episodes).
4. Use the *min* function as AND operator, as defined in the optimal Boolean algebra framework.

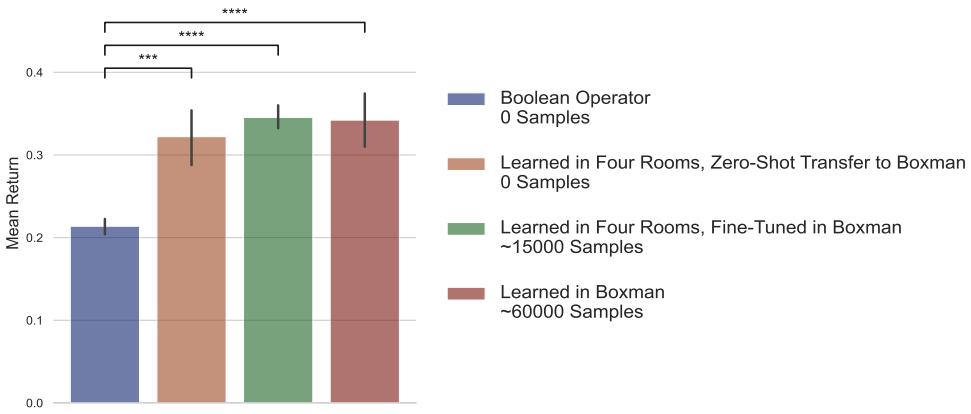


Figure 4: Mean Return for operators of different sources when performing conjunction. Number of Boxman environment samples needed to produce each operator are indicated. Results taken over 10 runs in each setup. Error bars represent 95% confidence interval. A significance marker of *** indicates a p-value ≤ 0.001, while **** indicates a p-value ≤ 0.0001

The number of Boxman environment samples observed during each case acts as an efficiency measure. The Boolean algebra and zero-shot transfer of a tabular operator require 0 Boxman samples. All uses of NEAT outperform the *min* conjunction operator. There are two important conclusions that one can take from this. Firstly, the zero-shot transfer of operators learned in a tabular domain with differing state space leads to significantly greater returns than the Boolean operator framework. Secondly, minimal fine-tuning of these operators produces roughly equal return to a NEAT run undertaken solely in Boxman, while requiring around $\frac{1}{4}$ as many environment samples.

5 Conclusion

Composition of learned behaviours is a growing area in reinforcement learning. Leveraging knowledge already possessed by an agent into new forms aids efficiency and brings us closer to realistic human-like learning. Optimal Boolean composition operators have been determined given that we have optimal knowledge of the tasks being composed. In practice, however, this is not always the case. In this work, we have examined how using NEAT to learn alternative composition operators stands up to multiple sources of unpredictability within RL environments for which the Boolean algebra framework does not account and where it fails to achieve desired performance—non-converged value functions, stochastic transitions, reward functions not fit for achieving composed tasks, and a higher-dimensional function approximation environment.

From this foundation, we emphasise the ability to transfer learned composition operators between domains with different state spaces. Operators learned in a tabular setting achieve competitive performance when applied in a function approximation setting, either through a zero-shot transfer or after being fine-tuned in this new environment. This transfer process projects strong generalisation and life-long learning abilities for RL agents.

Through this, we keep in mind certain limitations: including sample-inefficiency and an element of unpredictability to operator performance. Future work should focus on alleviating these issues, while building further on the encouraging robustness that arises from learning composition operators.

References

[1] Alan McIntyre, Matt Kallada, Cesar G. Miguel, Carolina Feher de Silva, and Marcio Lobo Netto. neat-python, 2017.

[2] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A boolean task algebra for reinforcement learning. *Advances in Neural Information Processing Systems*, 34:9497–9507, 2020.

[3] Geraud Nangue Tasse, Benjamin Rosman, and Steven James. World value functions: Knowledge representation for learning and planning. *Bridging the Gap Between AI Planning and Reinforcement Learning Workshop at ICAPS*, 2022.

[4] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.