In Proceedings of the 3rd IEEE Conference on Games, , 2021.

Generalization of Agent Behavior through Explicit Representation of Context

Cem Tutum
The University of Texas at Austin
cem.celal@gmail.com

Suhaib AbdulQuddos
The University of Texas at Austin
suhaib.abdulquddos@gmail.com

Risto Miikkulainen
The University of Texas at Austin and Cognizant AI Labs
risto@cs.utexas.edu

***Abstract*—In order to deploy autonomous agents in digital interactive environments, they must be able to act robustly in unseen situations. The standard machine learning approach is to include as much variation as possible into training these agents. The agents can then interpolate within their training, but they cannot extrapolate much beyond it. This paper proposes a principled approach where a context module is coevolved with a skill module in the game. The context module recognizes the temporal variation in the game and modulates the outputs of the skill module so that the action decisions can be made robustly even in previously unseen situations. The approach is evaluated in the Flappy Bird and LunarLander video games, as well as in the CARLA autonomous driving simulation. The Context+Skill approach leads to significantly more robust behavior in environments that require extrapolation beyond training. Such a principled generalization ability is essential in deploying autonomous agents in real-world tasks, and can serve as a foundation for continual adaptation as well.**

***Index Terms*—Neuroevolution, Evolution of Neural Networks, Extrapolation, Generalization, context**

I. Introduction

To be effective, autonomous agents need to be able to perform robustly in previously unseen situations. Especially in real-world decision making and control applications such as games, simulations, robotics, health care, and finance, agents routinely encounter situations beyond their training, and need to adapt safely. A common practice is to train these models, mostly deep neural networks, with data collected from a number of hand-designed scenarios. However, the tasks are often too complex to anticipate every possible scenario, and this approach is not scalable.

One popular approach to address this problem is few-shot learning, in particular metalearning, either by utilizing gradients [1]–[3] or evolutionary procedures [4], [5]. In metalearning, systems are trained by exposing them to a large number of tasks, and then tested for their ability to learn relevant but previously unseen tasks. There are also a number of approaches mostly for supervised learning setting where new labels need to be predicted based on limited number of training data. However, applications of few-shot learning to control and decision making, including reinforcement learning problems, are limited so far [6].

The approach in this paper is motivated by work on opponent modeling in poker [7], [8]. In that domain, an effective approach was to evolve one neural network to decide what move to make, and another to modulate those decisions by taking the opponents playing style into account. When trained with only a small number of very simple but different opponents, the approach was able to generalize and play well against a wide array of opponents, include some that were much better than anything seen during training.

In poker, the opponent can be seen as the context for decision making. Each decision needs to take into account how the opponent is likely to respond, and select the right action accordingly. The player can thus adapt to many different game playing situations immediately, even those that have not been encountered before. In this paper, this approach is generalized and applied to control and decision making more broadly. The main idea is that the history of the system is the context. A skill network reacts to the current situation, and a context network integrates observations over a longer time period. Together they learn to represent a wide variety of situations in a standardized manner so that a third, controller, network can make decisions robustly. Such a Context+Skill system can thus generalize to more situations than any of its components alone.

The Context+Skill approach is evaluated in this paper on several game domains: (1) Flappy Bird game extended to include more actions and physical effects (i.e. flap forward and drag in addition to flap upward and gravity); (2) Lunar Lander (from OpenAI Gym) extended with variations in main and side engine power as well as mass of the lander; and (3) CARLA autonomous driving environment where the steering curve, torque curve, and map can vary. Such extensions allow generating a range of unseen scenarios both by extending the range of effects of those actions as well as their combinations. The approach generalizes remarkably well to new situations, and does so much better than its components alone. Context+Skill approach is thus a promising approach for building robust autonomous agents in real-world domains.

II. Methodology

This section introduces the experimental domains, the Context+Skill approach, and the evolutionary training methodology.

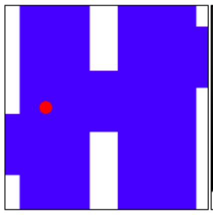
A. Experimental Domains

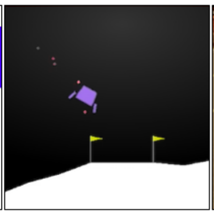
Three different environments are used for experiments. The first one, Flappy Ball (FB; Fig. 1(a)), is an extension of the popular Flappy Bird computer game [9]. The agent, controlled by a neural network, aims to navigate through the openings between pipes without hitting them for a certain length of time. The agent can flap forward and flap upward; gravity will pull it down and drag will slow it down. The agent gets a reward of +1 every time it passes a pipe successfully, and a penalty of -1 at each time step it crashes into the pipes and -5 when it crashes in the ceiling or ground. At every time step, the agent receives six-dimensional sensory information: vertical position, horizontal and vertical velocities, horizontal distance to the right edge of the closest pipe, and the height of the top and bottom pipes, normalized to [0,1]. The effects of flap upward and forward as well as gravity and drag can change between episodes; Therefore, the agent has to infer such variations from its interactions with the environment over time.

The second domain is LunarLander-v2 (LL; Fig. 1(b)) from OpenAI Gym suite, modified to allow variations in mass of the spacecraft and the effect of the main and two side engine thrusters. As in the original LL-domain, the agent receives eight-dimensional numerical sensory information as its input (i.e., position and velocity of the lander in 2D, its angle and angular velocity, and whether each leg touches the ground). The purpose is to land on the lunar surface in a designated region indicated with the flags safely and in minimal time. The episode finishes if the lander crashes or comes to rest.

The third domain is the CARLA open-source autonomous driving simulation environment (Fig. 1(c)). The agent has both lateral (steering) and longitudinal (throttle) control of the car while driving between two points in a given certain amount of time. The steering and the torque curves are modified to evaluate generalization. Furthermore the agent is placed in completely unseen tracks as it tries to complete the same task albeit under more difficult driving conditions (as imposed by the modified steering and torque curves). As in the other environments, the agent receives numerical sensory information as five numerical values: Rangefinder coordinates which describe the distance between the agent and the lane boundaries along five axes. All control actions are given as continuous values varying within [-1,1]. The objective is to minimize distance from the lane center, wobbliness, and final distance from a target zone.

These domains can be seen as proxy for control and decision making problems where the changes in the environment require immediate adaptation, such as operating a vehicle under different weather conditions, configuration changes, wear and tear, or sensor malfunctions. The challenge is to adapt the existing policies to the new conditions immediately without further training, i.e. to generalize the known behavior to unseen situations.


(a) Flappy Ball


(b) Lunar Lander

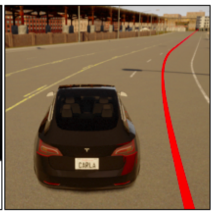

(c) CARLA

Fig. 1. Scenes from Flappy Ball (FB; (a)), Lunar Lander (LL; (b)), and CARLA (c). The red circle in the FB domain represents the agent and the white columns are pipes that move from right to left as the game progresses. The purple object in the LL domain represents the spacecraft that controls its main and side engines to land safely on the white surface designated with flags. In CARLA, the agent controls the throttle and steering of a car to reach destination staying as close as possible within the lane indicated by the red curve. The effects of the actions are varied across episodes to evaluate how well the controller adapts to previously unseen situations. For animated examples, see https://drive.google.com/drive/folders/1GBdJzD9tDHJkd59YbQUOIQua6nCiLjXa.

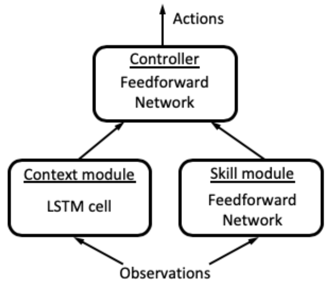
B. Context+Skill Model

The core idea evaluated in this paper is to implement the agent as a Context+Skill network that takes advantage of an explicit representation of context. The Context+Skill Network consists of three components: the Skill and the Context modules and the Controller (Figure 2(a)). The first two modules receive sensory information from the environment as numerical values, as described in previous subsection. They send their output to the Controller, a fully connected feedforward neural network that makes the decisions on which actions to take.

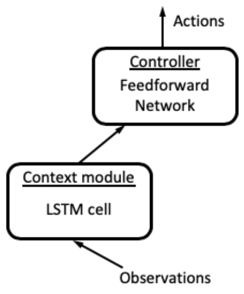
The Skill module is also a fully connected feedforward network. Together with the Controller they form the Skill-only Network S (Fig. 2(c)). The Skill module used in this study has 10 hidden and five output nodes and the Controller has 20 hidden hidden nodes. S is used as the baseline model throughout the study. In principle it has all the information for navigating through the pipes, but does not have the benefit of explicit representation of context.

The other main component in the Context+Skill framework is the Context module. It is composed of a vanilla Long Short Term Memory (LSTM) cell [10]. There are three gates in this recurrent memory cell: input, forget, and output. The gates are responsible for learning what to store, what to throw away, and what to read out from the long-term memory of the cell. Thus, the cell can learn to retain information from the past, update it, and output it at an appropriate time, thereby making it possible to learn sequential behavior [11], [12].

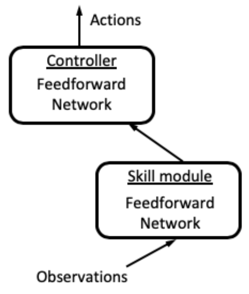
The Context module used in this study consists of an LSTM cell size of 10. The memory of the Context module (h_{t-1} and c_{t-1}) is reset at the beginning of each new task, and accumulated (transferred) across episodes within each task. It can therefore form a representation of how actions affect the environment. The output of the LSTM (h_t) is sent to Controller as the context. Together the Context module and the Controller form the Context-only network C (Fig. 2(b)). It serves as a second baseline, allowing integration of observations over



(a) Context+Skill Network, CS



(b) Context-only Network, C



(c) Skill-only Network, S

Fig. 2. The architecture of the Context+Skill network and its ablations. (a) The full network consists of three components: a Skill module that processes the current situation, a Context module that integrates observations over the entire task, and a Controller that combines the outputs of both modules, thereby using context to modulate actions. This architecture is compared to (b) context-only ablation, and (c) skill only ablation in the experiments. Each component is found to play an important role, allowing the CS network to generalize better than its ablations.

time, but without a specific Skill network to map them directly to action recommendations.

The complete Context+Skill Network, CS (Fig. 2(a)) consists of both the Context and Skill modules as well as the Controller network of the same size as in C and S. The motivation behind the CS architecture, i.e. of integrating the Context module with S, is to make it possible for the system to learn to use an explicit context representation to modulate its actions appropriately. The method for discovering these behaviors is discussed next.

C. Evolutionary Learning Process

The goal in each domain is to find a safe solution that optimizes the performance objective. Although it is possible to formulate the optimization process based on that single objective, it turns out the diversity resulting from the multiobjective search speeds up training and helps discover well-performing solutions [13]. In the experiments in this paper, the first solution that reaches a satisfactory level in the safety objective is returned as the final result of evolution, and its generalization ability evaluated.

In each domain, the first objective, f_0, measures safety, and the second, f_1, measures performance, and is in conflict with safety. Accordingly in the FB domain, the number of any type of collisions (or hits) is minimized, whereas the number of successfully passed pipes is maximized. In the LL-domain, the total rewards (indicating a successful landing) is maximized, whereas the landing time is minimized. In the CARLA domain, safety f_0 is optimized by minimizing a cumulative weighted sum of a lane-distance penalty and a wobbliness penalty:

$$f_0 = \sum_{t=1}^{t=T} (d(t) + \lambda \mathrm{abs}(s(t) - s(t-1))), \tag{1}$$

where $d(t)$ is the distance from the center of the lane at time t (shown as a red line in Fig. 1(c)), $s(t)$ is the steering output at time t, and λ is a proportionality coefficient ($= 5.5$ in the experiments below). Time is incremented in 1/30 sec intervals; each episode lasts $T = 20$ seconds, or less if the agent reaches a specified target zone (identifying the end of the track) before that. Performance f_1 is optimized by minimizing the Euclidean distance between the agent and the target zone at the end of the episode.

TABLE I
PARAMETER RANGES DURING TRAINING

FB (±20%)			LL (±10%)			CARLA (±15%)		
$\text{Flap}_{\text{base}}$	=	-12.0	$\text{Main}_{\text{base}}$	=	20.0	α_{base}	=	1.0
Fwd_{base}	=	5.0	$\text{Side}_{\text{base}}$	=	1.0	β_{base}	=	1.0
$\text{Gravity}_{\text{base}}$	=	1.0	$\text{Mass}_{\text{base}}$	=	8.0			
$\text{Drag}_{\text{base}}$	=	1.0						

In the evolutionary learning process, the weights of all three neural networks described in Section II-B are evolved while the network architecture remains fixed. The goal is to maximize the average fitness across multiple tasks, where each task is based on different physical parameters in the FB, LL and CARLA domains. The FB-domain has four parameters (Flap, Fwd, Gravity, Drag), the LL-domain has three parameters (Main, Side, Mass) and the CARLA-domain has two parameters (α and β, which control the steering angle and torque curves of the car, respectively). In each task during evolution, only one parameter is subject to change, while the rest are fixed at their base values (given in Table I). Thus, the number of tasks is equal to the number of parameters. The parameters in each domain are varied during training within ±20%, ±10% and ±15% in the FB, LL and CARLA-domains, respectively.

Each task, and therefore each parameter, is uniformly sampled n_{episodes}=5 (except in the CARLA domain where each parameter is sampled six times) times within the limits specified above. Thus, there are 20 fitness evaluations per individual in each generation in the FB domain, 15 evaluations in the LL-domain and 12 evaluations in the CARLA-domain. The fitness of every individual in the population, i.e., average score of all episodes, is evaluated in parallel on the same task distribution for a fair comparison. The memory of Context module in CS and C is reset at the beginning of each task, and transferred from episode to episode otherwise.

Non-dominated sorting genetic algorithm (NSGA-II) [14] was implemented in the DEAP evolutionary computation

Algorithm 1 Evolutionary process for training networks

```
1:  procedure EVOLVE
2:      stop := False
3:      parents := random_init_individuals($\mu$)
4:      task_params = prepare_task_params($n_{episodes}$,$n_{tasks}$)
5:      fitness = distribute(eval_fitness(), (parents, task_params))
6:      for gen from 1 to $n_{gen}$ do
7:          offspring = tournament_sel_DCD(parents, $\mu$)
8:          for i from 1 to $\lambda$ do
9:              if random() $\leq$ $p_{crossover}$ then
10:                 SBX(offspring[i], offspring[i+1])
11:             Polynomial_Mutation(offspring[i])
12:             Polynomial_Mutation(offspring[i+1])
13:         params = prepare_task_params($n_{episodes}$,$n_{tasks}$)
14:         fitness = distribute(eval_fitness(), (parents, task_params))
15:         for j from 1 to $\lambda$ do
16:             if fitness[j][0] $\geq$ $pipes_{max}$ then
17:                 if fitness[j][1] $\leq$ $hits_{max}$ then
18:                     stop := True
19:         parents := tournament_sel_DCD(parents + offspring, $\mu$)
20:         if stop == True then
21:             return parents
22:             break
```

framework [15] as the optimization method. The overall procedure is shown in Algorithm 1. It receives n_{tasks}, n_{episodes}=5, perturb=0.2 in FB, 0.1 (in LL) or n_{episodes}=12, perturb=0.15 (in CARLA), base parameter values, μ = 96 (48 in CARLA), $p_{\text{crossover}}$ = 0.9, n_{gen} = 2,500 as input. The population size (μ) is chosen as a multiple of 24 since the fitness evaluations are distributed among 24 threads on a cluster (i.e., Dell PowerEdge M710, 2× Xeon X5675, six cores @ 3.06GHz). Default NSGA-II settings, including SBX (Simulated Binary Crossover), Polynomial Mutation, tournament_sel_DCD (Tournament Selection Based on Dominance), and (μ + λ) elitist selection strategy were used [14]. The process is robust to minor variations of these choices.

III. Results

The evolutionary learning results are first described, followed by evaluation of the generalization ability of the best solutions. For animated examples of behaviors in each domain, see https://drive.google.com/drive/folders/1GBdJzD9tDHJkd59YbQUOIQua6nCiLjXa.

A. Learning

CS, C, and S architectures were evolved with different random seeds six times in FB, five times in LL, and once in CARLA (due to the computational complexity of this domain). To avoid overfitting, specific stopping criteria were selected for each domain after some experimentation. In FB, training was run until an individual in the population achieved a fitness scores of at least f_0=0.01 (hits) and f_1=22.0 (pipes). In LL, training was run until an individual reached f_0=200 (total rewards). In CARLA, evolution was run until an individual

TABLE II
Parameter ranges during generalization

FB (±75%)			LL (±50%)			CARLA (±35%)		
Flap_{min}	=	-21.0	Main_{min}	=	10.0	α_{min}	=	0.65
Flap_{max}	=	-3.0	Main_{max}	=	30.0	α_{max}	=	1.35
Fwd_{min}	=	1.25	Side_{min}	=	0.5	β_{min}	=	0.65
Fwd_{max}	=	8.75	Side_{max}	=	1.5	β_{max}	=	1.35
$\text{Gravity}_{\text{min}}$	=	0.25	Mass_{min}	=	4.0			
$\text{Gravity}_{\text{max}}$	=	1.75	Mass_{max}	=	12.0			
$\text{Drag}_{\text{base}}$	=	0.25						
$\text{Drag}_{\text{base}}$	=	1.75						

reached f_0=2043 (safety penalty) and f_1=55 meters (distance from the target). In FB and CARLA a minimum performance criteria was included to make sure the agent would not simply optimize safety by staying still. The final Pareto-optimal set in each run contained multiple individuals, of which one representative network was selected for the generalizationy evaluation. In FB and LL it was a safe network (i.e. satisfying the stopping criteria) with the highest performance, and in CARLA, it was the network with the least Euclidean distance to the origin in the Pareto front.

The evolution of S in general took the shortest amount of generations since it has the least number of parameters (e.g. 287 compared with 982 in C and 1207 in CS in the FB domain). The same architecture is used for all three domains. To make sure the number of parameters was not a factor, another S with a larger Skill module, with the same number of parameters as CS, was also evolved with the same stopping criterion. However, it performed poorly compared to the smaller S in the generalization studies, apparently because it was easier to overfit. Thus, it was excluded from the comparisons.

B. Generalization

To evaluate the generalization performance of the best performing networks, the task parameters in each domain were changed in the following ways:

- The range of variation in the task parameters was increased 35-75% beyond the training limits (Table II);
- All parameters were varied simultaneously instead of one at a time; and
- In the CARLA domain, the agents were evaluated on a new track that had never been encountered during training. This track included curves with significantly higher curvature than those encountered during training.

More specifically, the range of each game parameter was discretized into 10, 20 and 35 values in FB, LL and CARLA, respectively, and each network was tested with every possible combination of these values, i.e. with 10,000, 8000, and 1225 different settings. In FB and LL, each parameter combination was tested three times and the results averaged; in CARLA, only one sample was used. Figures 3, 4, and 5 show the difference in generalization between CS and S, CS and C and C and S. They are histograms indicating how often each difference was observed in the episodes. The first network in the subtraction performs better wrt. a minimization objective

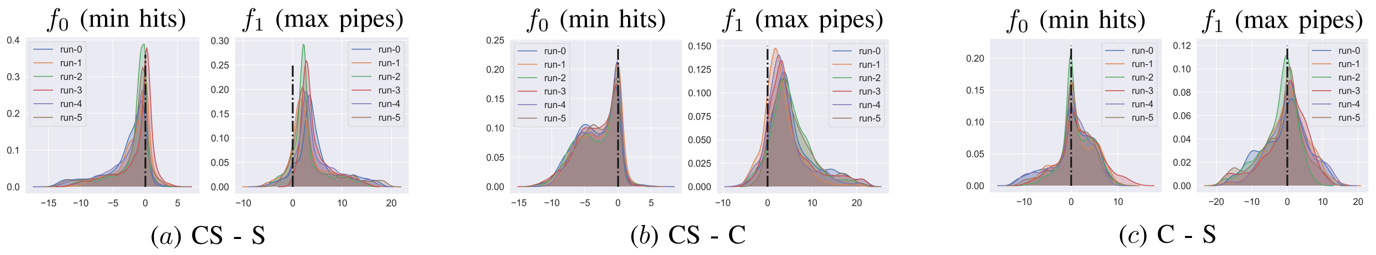


Fig. 3. Comparing generalization of CS and its ablations C and S in the Flappy Ball domain. The x-axis shows the differences in generalization performance across the 3×10^4 test episodes for the three pairs of architectures. With minimization objectives, a distribution that is skewed to the left of the 0 line (black dashed line) is better, and with maximization, a distribution that is skewed to the right is better. CS generalizes much better than S (a) and C (b), which are about equal (c).

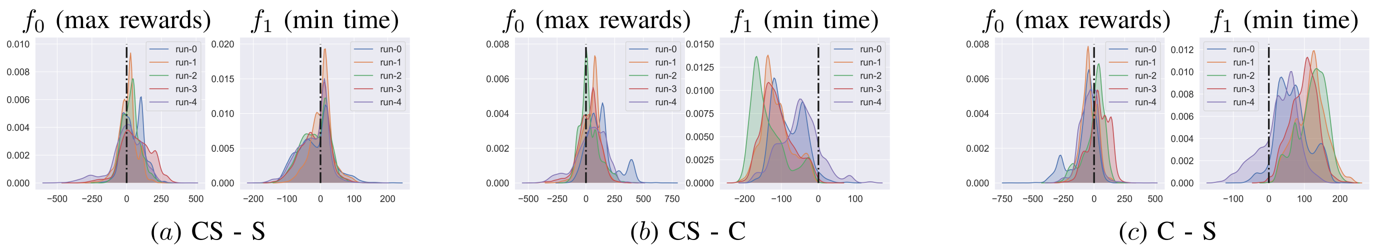


Fig. 4. Comparing generalization of CS and its ablations C and S in the Lunar Lander domain. As in Flappy Ball, CS generalizes much better than S (a) and C (b), which are about equal in rewards with reactivity providing an advantage in time (c).

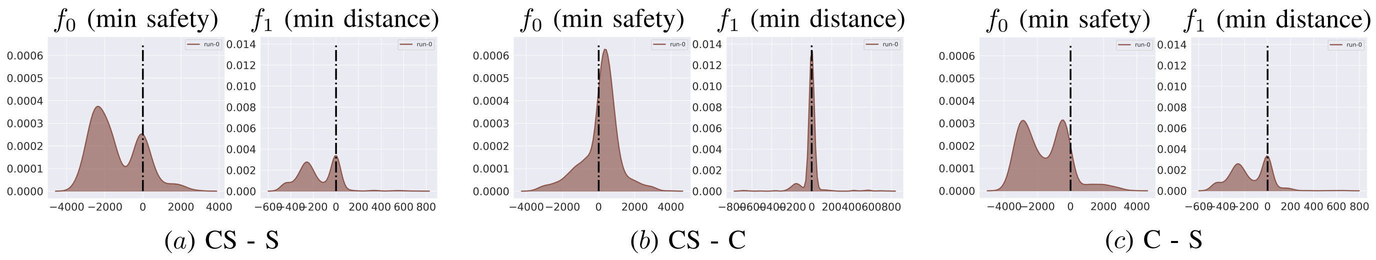


Fig. 5. Comparing generalization of CS and its ablations C and S in the CARLA domain. As in the other two domains, CS generalizes much better than S (a) and is slightly better in performance and about equal in safety than C (b). C in this domain performs better than S (c), suggesting that context dynamics are more important than reactivity in this domain.

if the histogram is skewed to the left, and wrt. a maximization objective if the histogram is skewed to the right.

Thus, the plots show that in both FB and LL, CS generalizes much better than S and C both in terms of safety and performance. In CARLA, CS generalizes better than S and C in terms of performance; in terms of safety, CS generalizes much better than S in terms of lane distance and about the same in wobbliness, and it generalizes much better than C in terms of wobbliness and about the same or slightly less in terms of lane distance. Thus, CS consistently achieves superior performance and mostly superior safety across multiple domains.

IV. Behavior Analysis

To understand how the CS architecture outperforms its individual components C and S, an FB task with parameters [Flap=-7.0, Gravity=0.58, Fwd=8.75, Drag=0.58] was evaluated further. This setting has a previously unseen exaggerated effect for forward flap, and a previously unseen diminished effects for upward flap, gravity and drag. Thus, actions tend to push up and speed up the agents more than expected, and it is difficult for it to slow down and come down.

Neither the C nor the S network performed well in this task: The C network collided with six pipes and S with five. Remarkably, CS managed to pass all 21 pipes. Both C and S used all four actions (flap upward, forward, simultaneously upward and forward, and glide, i.e. do nothing), but CS interestingly never uses flap upward alone. That action simply lifts the agent up, which is rarely optimal action in this environment where it takes such a long time to come down. If it is necessary to go up it is because the opening is high, and in that case it is more efficient to move forward at the same time.

As an illustration, second row in Fig. 6 shows a situation at the fourth and fifth pipe. Both C and S make a similar mistake by flapping up and forward. They end up too high too fast, do not have enough time to come back down, and crash into the fifth pipe. In contrast, even before the fifth pipe becomes visible, CS refrains from both actions while there is enough time for weaker gravity and drag to slow and pull down the agent, and it reaches the opening in the fifth pipe just fine.

To understand how CS manages to implement this behavior whereas C and S do not, the outputs of the Context and Skill

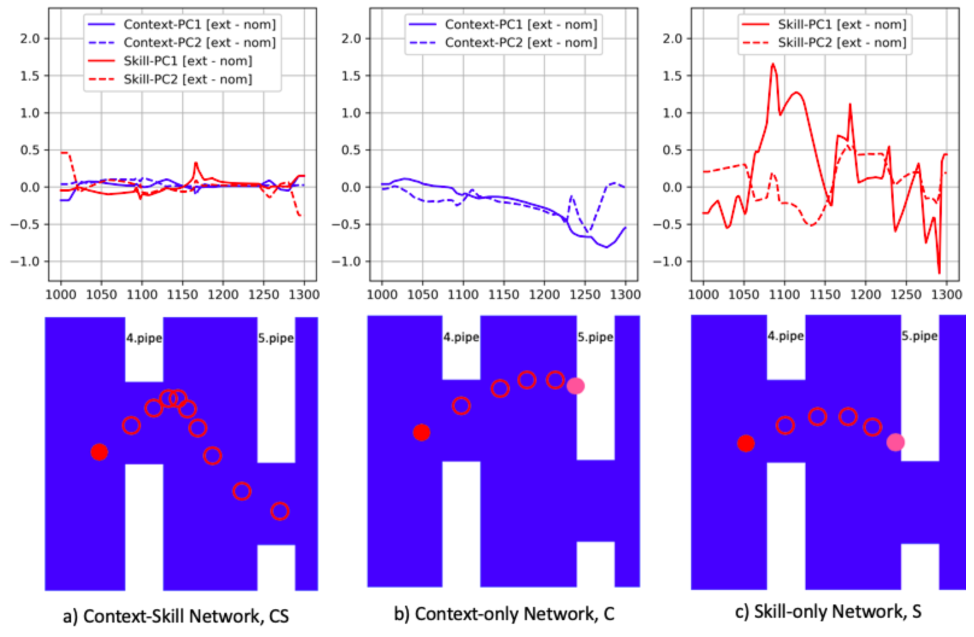


Fig. 6. Contrasting the generalization ability of (*a*) CS, (*b*) C, and (*c*) S networks. Bottom row: At the fourth pipe in the generalization task, S and C flap upward and then forward, end up too high too fast without enough time to come back down, crashing into the fifth pipe. In contrast, the CS Network avoids the collision by correctly estimating the effects of its actions, giving itself enough time to come down. Top row: Differences between the first two principal components (PC1 and PC2) of the module outputs between the nominal and generalization tasks. They differ little in CS, making it easier for the controller to output the correct actions in new contexts. For an animation of these episodes, see https://drive.google.com/drive/folders/1GBdJzD9tDHJkd59YbQUOIQua6nCiLjXa.

modules are compared between this generalization task (where C and S hit the fifth pipe) and a task where all parameters are at their base values (where C and S do not hit the fifth pipe). Their 10 and 5-dimensional outputs are first reduced to two dimensions through principal component analysis and then subtracted. The top row of Fig. 6 shows these differences for the Context and Skill modules of CS, for the Context module of C, and for the Skill module of S, at the locations in the image below.

One might intuitively expect that Context module in C and Skill module in S would not change their behavior much in the generalization task, but the Context module in CS would vary significantly to modulate the output of the Skill module. Surprisingly, the opposite is true: Both the Context and the Skill module in CS vary very little compared to those in C and S (see also the quantitative comparison in Table III). The generalization task presents novel inputs that results in novel outputs in C and S, and the controller does not know how to map them to correct actions. In contrast, the Context and Skill modules in CS have learned to standardize their output despite the change in context; their outputs are what the controller expects as its input, and is able to output the correct actions. Interestingly, this effect is similar to standardizing context in sentence processing, which makes it possible to generalize to novel sentence structures [16]. Remarkably, whereas in sentence processing the standardization was implemented by a hand-designed architecture, in the Context+Skill approach it is automatically discovered by evolution.

V. Discussion and Future Work

The Context+Skill approach represents context explicitly, and has a remarkable ability to generalize to unseen situations. In the experiments so far, the neural networks have a fixed topology; it may be possible to customize their architecture further through evolution [17], [18], and thereby delineate and optimize their roles further. Besides the architecture, the choice of training tasks plays an important role; methods that automatically design a curriculum, i.e., a sequence of new training tasks [19]–[23], could lead to further improvements. Third, instead of using handcrafted features, convolutional layers added in front of the Context and Skill modules could be used to discover features while training, extending the approach to more general visual tasks.

Lifelong machine learning tries to mimic how humans and animals learn by accumulating the knowledge gained from past experience and using it to adapt to new situations incrementally [24]. The generalization ability of the Context+Skill approach can serve as a foundation for continual learning. It provides an initial rapid adaptation to new situations upon which further learning can be based. How to convert generalization into a permanent ability in this manner is an interesting direction of future research.

VI. Conclusion

A major challenge in deploying artificial agents in the real world is that they are brittle—they can only perform well in situations for which they were trained. This paper demonstrates a potential solution based on separating contexts

TABLE III
CHANGE IN CONTEXT AND SKILL MODULES DURING GENERALIZATION.

Network	PC	MSD	STD
CS	Context-PC1	0.004	±0.058
	Context-PC2	0.003	±0.039
	Skill-PC1	0.007	±0.082
	Skill-PC2	0.018	±0.133
C	Context-PC1	0.139	±0.282
	Context-PC2	0.061	±0.147
S	Skill-PC1	0.414	±0.600
	Skill-PC2	0.088	±0.286

from the actual skills. Context can then be used to modulate the actions in a systematic manner, significantly extending the unseen situations that can be handled. This principle was successfully evaluated in three domains: challenging versions of the Flappy Bird and Lunar Lander games as well as the CARLA autonomous driving simulation. The results suggest that the Context+Skill approach should be useful in many control and decision making tasks in the real world.

VII. ACKNOWLEDGMENTS

This research was supported in part by DARPA L2M Award FA8750-18-C-0103.

REFERENCES

[1] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook," Ph.D. dissertation, Institut für Informatik, Technische Universität München, 1987.

[2] S. Thrun and L. Pratt, *Learning to Learn: Introduction and Overview*. USA: Kluwer Academic Publishers, 1998, pp. 3–17.

[3] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017.

[4] C. Fernando, J. Sygnowski, S. Osindero, J. Wang, T. Schaul, D. Teplyashin, P. Sprechmann, A. Pritzel, and A. Rusu, "Meta-learning by the Baldwin effect," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM Press, 2018.

[5] D. Grbic and S. Risi, "Towards continual reinforcement learning through evolutionary meta-learning," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '19. Association for Computing Machinery, 2019, pp. 119–120.

[6] K. Kansky, T. Silver, D. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George, "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics," 2017.

[7] X. Li and R. Miikkulainen, "Evolving adaptive poker players for effective opponent exploitation," in *AAAI-17 Workshop on Computer Poker and Imperfect Information Games*, 2017.

[8] ——, "Opponent modeling and exploitation in poker using evolved recurrent neural networks," in *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO 2018)*. Kyoto, Japan: ACM, July 2018.

[9] Wikipedia, "Flappy bird," https://en.wikipedia.org/wiki/Flappy_Bird, 2020, accessed 3-February-2020).

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[12] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. O'Reilly Media, Inc., 2017.

[13] J. D. Knowles, R. A. Watson, and D. W. Corne, "Reducing local optima in single-objective problems by multi-objectivization," in *Evolutionary Multi-Criterion Optimization*, E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, and D. Corne, Eds. Springer Berlin Heidelberg, 2001, pp. 269–283.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[15] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[16] R. Miikkulainen, "Subsymbolic case-role analysis of sentences with embedded clauses," *Cognitive Science*, vol. 20, pp. 47–73, 1996.

[17] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[18] J. Schrum and R. Miikkulainen, "Evolving multimodal behavior with modular neural networks in ms. pac-man," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '14. Association for Computing Machinery, 2014, pp. 325–332.

[19] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," 2018.

[20] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions," 2019.

[21] J. Schmidhuber, "Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem," 2011.

[22] N. Justesen and S. Risi, "Automated curriculum learning by rewarding temporally rare events," 2018.

[23] S. Risi and J. Togelius, "Procedural content generation: From automatically generating game levels to increasing generality in machine learning," 2019.

[24] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," 2018.