Improved Learning in Evolution Strategies via Sparser Inter-Agent Network Topologies

Dhaval Adjodah*, Dan Calacci, Yan Leng, Peter Krafft, Esteban Moro, Alex Pentland
MIT Media Lab, *Foundation Bruno Kessler, Italy
`{dval, dcalacci, yleng, pkrafft, emoro, pentland} @mit.edu`

Abstract

We draw upon a previously largely untapped literature on human collective intelligence as a source of inspiration for improving deep learning. Implicit in many algorithms that attempt to solve Deep Reinforcement Learning (DRL) tasks is the network of processors along which parameter values are shared. So far, existing approaches have implicitly utilized fully-connected networks, in which all processors are connected. However, the scientific literature on human collective intelligence suggests that complete networks may not always be the most effective information network structures for distributed search through complex spaces. Here we show that alternative topologies can improve deep neural network training: we find that sparser networks learn higher rewards faster, and at lower communication costs.

1 Introduction

We draw upon a previously largely untapped literature on human collective intelligence as a source of inspiration for improving deep learning via evolutionary algorithms. Distributed evolutionary algorithms have proven to be capable of state-of-the-art training for deep neural networks on reinforcement learning tasks [12]. These algorithms share and remix the parameter values discovered as a population of processors (which we refer to as ’agents’ here). Implicit in these algorithms is the network structure along which processors share parameter values are shared. So far, existing work applying evolutionary algorithms to deep learning has implicitly utilized fully-connected networks in which all processors are connected.

The scientific literature on human collective intelligence suggests that fully-connected networks may not always be the most effective for distributed search through complex spaces [7, 8]. Simulations and human experiments have shown that sparser network structures can improve collective learning in a variety of group problem-solving scenarios [7, 8, 1]. Sparser networks are topologies where agents are not sharing learning from every other agent (fully-connected), and are instead organized in less-connected structures.

Here, we show that alternative network structures can improve deep neural network training using Evolution Strategies (ES) [12] running OpenAI’s Roboschool 3D Humanoid Walker as our learning task. We find that sparser network topologies of agents (i.e. processors) perform better, and with a significantly lower communication cost. Although the highest-performing topologies we study require that each agent communicate with only 4–10% of all other agents on average, they can learn faster and produce higher rewards than the fully-connected baseline.

Our key findings are as follows:

1. We find that explicitly designed networks that incur a lower communication cost yield faster and higher learning than fully-connected networks in an evolutionary algorithm for deep reinforcement learning.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2. These networks result in a multiplicative effect in total reward: networks with only 1,000 agents produce results competitive to fully-connected networks with 4,000 agents.
3. We find that sparser graphs can achieve up to 33.5% higher reward than a corresponding fully-connected network, and that they can reach the fully-connected maximum up to 32% earlier.

2 Related Work

Distributed DRL approaches attempt to solve the fundamental predicted instability [15] of using non-linear approximators such as neural networks to represent the action-value function. If several agents pool their very varied experiences together, then the model can be learned with non-correlated data. The Gorila framework [11] collects many experiences in parallel from many agents and pools them into a global memory store on a distributed database. A3C [9] instead runs many agents asynchronously on several environment instances while varying their exploration policies. This effectively increases exploration diversity in parameter space and de-correlates agent data while reducing resource usage significantly, but can also cause heavy communication bottlenecks.

Recently, black box optimization methods such as evolution strategies have been shown to overcome such communication bottlenecks. ES runs many agents that need only to share their scalar reward values each iteration to learn efficiently. This data efficiency allows ES to solve benchmark DRL problems in record time by utilizing a very large number of CPU cores distributed over a network.

All the approaches described above organize agents in a fully-connected network topology: the algorithm updates a global-level parameter set using information available from all agents at every step. As described in the next section, the scientific literature on collective intelligence in animal and human groups suggest that other network topologies could be more effective for such distributed search problems in complex task spaces.

2.1 Human Collective Intelligence

Studies of human and animal groups have revealed that groups of problem-solvers often exhibit capabilities well beyond the skill of any of their individual members. This emergent problem-solving ability of groups is called their collective intelligence (CI), and it has been found to be affected by an array of factors such as the learning strategies of group members and their communication network structure [16, 7]. Recent studies of collective intelligence have modeled groups of problem-solvers as distributed information processors (i.e. agents), and we take this philosophical approach here [6].

The network structure that agents use to share information significantly influences the performance of groups. For example, in studies of simulated groups that attempt to search an NK task space (a parameterized space of arbitrarily high ruggedness), sparser networks have been shown to result in increased exploration, higher overall maximum reward and higher diversity of solutions [4, 7].

In human experiments, where agents attempt to solve a different task than the class of NK problems, denser communication networks have instead resulted in higher group performance [8]. Recent work has shown that these opposite effects can be explained by the different learning strategies agents employ and the complexity of the target task [1].

3 Algorithm: Networked Inter-Agent Learning

We introduce the notion of network topology and independent agent updates to the ES paradigm. Instead of updating a global policy using all agents at each iteration, each agent in the network performs an update using only its neighboring nodes. In implementation, our approach maintains the massive scalability of the original ES algorithm, allows for greater exploration of parameter space, and does not require a centralized master in implementation for learning: learning can be done at a node-centric level.

3.1 Evolution Strategies

One of the central limitations of modern distributed DRL is the lack of scalability due to the high communication costs of sharing parameters between agents. ES attempts to solve this problem by using a derivative-free approach, which we loosely outline here. ES chooses a fixed deep architecture, and initializes a single set of network weights θ. ES then creates a population of N parameters, each perturbed from θ by adding randomly sampled Gaussian noise $\epsilon_i \sim \mathcal{N}(0, I)$ to θ directly in parameter space. The rewards from running the target task using these perturbed weights are collected, and a gradient is constructed by calculating a weighted average of perturbations via the rewards F_i, $\alpha \frac{1}{N\sigma} \sum_i^N F_i \epsilon_i$. In implementation, this scheme requires that agents only share their scalar rewards, allowing ES to scale massively to thousands of CPUs. This scalability has allowed ES to attain state of the art performance on some of the hardest DRL benchmarks in record time, for example by solving the Mujoco Humanoid Walker task in 10 minutes.

3.2 Networked Evolution Strategies

The main differences between Networked Evolution Strategies (NES) and the original ES are summarized in Table 1. In NES, we introduce the notion of independent, networked "agents", each with their own individual parameter set θ_i, that perform updates separately. Previously, Evolution Strategies would run a number of episodes, each with a noised version of the parameter. In NES, we deploy agents that each run an episode with a different parameter. The agents are arranged in an undirected, unweighted network structure $G = (V, E)$, with each agent i corresponding to a node $v_i \in V$ in the network. On each iteration t, we perturb the parameter set of agent i, θ_i^t, by a Gaussian noise vector ϵ_i sampled in the same way as in the evolution strategies algorithm.

	Original ES	Networked ES
No. of parameters being explored	1	n (no. of agents)
Percent. agents needed for update	100	4–10
Use of broadcast	no	yes
Possible Topologies Allowed	fully-connected	all

Table 1: Main differences between ES and NES

In the optimization step, each agent performs its own independent update. Each agent i uses the same rank-centered weighting function as ES, but only uses the closed set of their neighborhood, $N[i]$, to perform the update. This set of nodes includes node i itself. Because different agents have different parameters, we calculate the difference in parameters between θ_i^t and each perturbed parameter set of other agents in $N[i]$, $(\theta_i^t - (\theta_k^t + \sigma\epsilon_k))$. We then weight each difference with its reported reward F_k^t, instead of calculating a gradient by computing a weighted average over the perturbations applied to each neighbor's parameter set (as in the Evolution Strategies algorithm). Each agent's parameter update at time $t+1$ is then $\theta_i^{t+1} \leftarrow \theta_i^t + \alpha \frac{1}{n\sigma} \sum_{k=1}^{N[i]} F_k^t(\theta_i^t - (\theta_k^t + \sigma\epsilon_k))$, as shown in Algorithm 1.

This change in each agent's parameter update means that the parameter sets of different nodes diverge after the first update. The update step in the networked algorithm presented here has each node effectively solving for its neighborhood's average objective, rather than the global average objective as in ES. In the case of a fully-connected network, each agent's neighborhood $N[i]$ is equal to the full set of vertices V, and the update is equal to the case of the original ES algorithm. We hypothesize that the divergent objective functions in the case of networked ES results in a greater diversity of policies being explored. Although the neighborhood-only constraint on node parameter updating does not add any penalty term to the update step (line 13-14 in Algorithm 1), updating using only one's neighbors can be very roughly interpreted as a type of regularization in the same way that Dropout [13] is not strictly regularization but is often seen as acting like regularization by preventing over-fitting.

A final difference in our algorithm is the implementation of stochastic global broadcast. This was implemented to counteract the problem that, as nodes are now searching for better parameters in their local neighborhood only, the effective combination of possible parameters around any parameter decreases significantly, scaling with the size of a node's neighborhood. We take inspiration from random restarts in simulated annealing and implement a stochastic global broadcast: with probability β each iteration, we force all agents to adopt the highest-performing parameter set from the previous iteration, centering the network on a current local maximum. We find that past a certain minimum ($\beta \approx 0.5$), broadcast has minimal effect on both the reward and learning rate of the network topologies we test.

Algorithm 1 Networked Evolution Strategies

```
Input: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0^i$ where $i$ = 1, 2, . . ., n (for n workers), communication Graph $G$, global broadcast probability $B$
Initialize: $n$ workers with known random seeds, initial parameters $\theta_0^i$
for $t$ = 0, 1, 2,. . . do
    for each worker $i$ = 1, 2, . . ., n do
        Sample $\epsilon_i \sim \mathcal{N}(0, I)$
        Compute returns $F_i$ = $F(\theta_i^t + \sigma\epsilon_i)$
    Send scalar returns $F_i$ to worker $i$'s neighbors $j$ = 1, 2,. . ., m as defined by graph $G$
    Sample $\beta \sim \mathcal{U}(0, 1)$
    if $\beta < B$ then
        Set $\theta_i^{t+1} \leftarrow \arg\max_{\theta_i^t} F(\theta_i^t)$
    else
        for each worker $i$ = 1, 2, . . ., n do
            Reconstruct all perturbations $\epsilon_k$ for neighbors k on $G$, k = 1,. . ., m
            Set $\theta_i^{t+1} \leftarrow \theta_i^t$ + $\alpha \frac{1}{n\sigma} \sum_{k=1}^{m} F_k^t(\theta_i^t - (\theta_k^t + \sigma\epsilon_k))$
```

4 Experiment Setup

Here, we describe the setup of experiments designed to test how different network metrics affect our algorithms' learning. We choose to focus our experiments on OpenAI's Roboschool 3D Humanoid Walker (specifically, RoboschoolHumanoid-v1, shown in Figure 1), an open-source implementation of MuJoCo [14]. The 3D Humanoid Walker is considered to be a difficult benchmark task, and therefore serves as a good point of comparison between learning algorithms. Many other learning tasks exists, such as the Atari game environments [10].

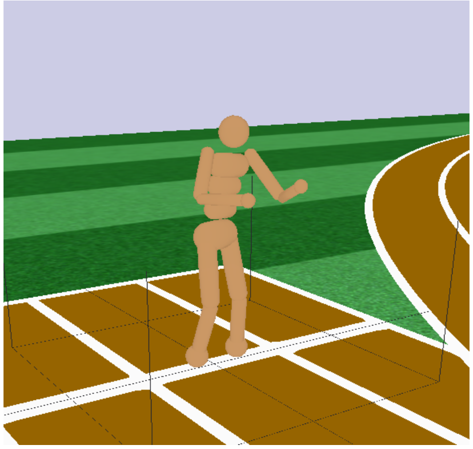

Figure 1: A screenshot of Roboschool's 3D Humanoid Walker, RoboschoolHumanoid-v1.

4.1 Graph Generation

To run our experiments, we generate a large set of canonical network topologies, as well as a set of engineered topologies that were designed to isolate various network statistics. In all cases, we fix the number of nodes (agents) to be 1000. We generate a population of Erdos-Renyi random graphs by varying the routine's main parameter, p. Erdos-Renyi random graphs are constructed by connecting each pair of possible nodes at random with probability p. We ensure that the network consists of only one component (i.e. that there are no disconnected nodes or components in the network). An example of an Erdos-Renyi Graph of average density p = 0.4 can be seen in Figure 2, compared to a fully-connected network with the same number of nodes (only 40 nodes are used in this example for illustration, although we use 1000 nodes in our experiments).

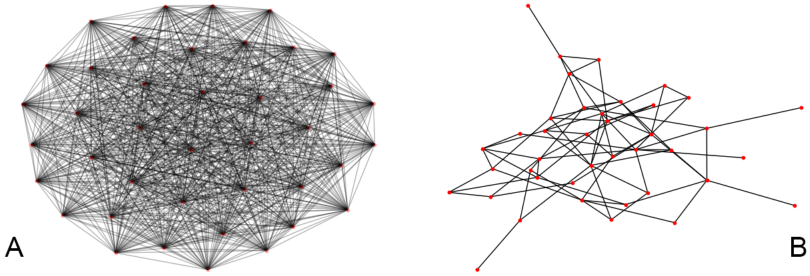

Figure 2: Comparison of a fully-connected network topology (A) and an Erdos-Renyi graph (B) of average degree 0.4, both with 40 nodes.

To control for and explore network characteristics more independently, we use random partition graph generation, a generalization of the planted-l-partition scheme introduced in [2], which allows us to vary statistics such as clustering and centrality, while keeping modularity constant. We first split the graph into k sub-communities, and assign each node to a sub-community with uniform probability, similar to an Erdos-Renyi graph. We then run the following routine for a set number of iterations: first, sample a source node n_s from the network, then, with probability p_{in}, sample a second target node n_t from the same cluster n_k that both n_s and n_t belong to. Otherwise, with probability p_{out}, sample the node n_t from all nodes not in the same cluster n_k, and construct an edge between n_s and n_t (in between clusters). All sampling is done with replacement, resulting in graphs with differing numbers of edges. In effect, our engineered graphs are actually a number of smaller Erdos-Renyi clusters connected to each other, making them sufficiently similar to be easily compared to the results of the Erdos-Renyi graphs.

4.2 Experiments

We first create a baseline by running fully-connected networks of 1000 agents using OpenAI's original ES code 10 times - they repeat each experiment 7 times. We then fit each run using a logistic growth function similar to [3]. We use the higher asymptote as a measure of maximum reward for each run, and then use the average of these maximum asymptotic rewards as a measure of performance, henceforth referred to as the baseline. Although we fit the learning trajectory using growth functions because there are random jumps to very high reward values, we find that our results do not vary significantly if we use other measures such as the mean or median of the top 5% of rewards over time.

We then run all our network variants (both in terms of topology and attributes) and similarly obtain a measure of the mean asymptotic reward. We take care to make sure we compute these asymptotes over the same number of iterations to maintain comparability of results, and we also ensure that rewards stabilize over time to an asymptote in order to get an accurate observation of maximum achieved reward.

5 Results

5.1 Higher and Faster Learning

As can be seen in Figure 3, our best network (an engineered network with 1000 agents) not only beats fully-connected networks with a similar number of agents (processors), but can beat up to 4000 agents arranged in a fully-connected network. This increase in efficiency could be due to the vastly larger parameter space being explored by each local neighborhood.

Regarding Erdos-Renyi networks, they achieve up to a 26% increase from the baseline reward, as shown in Figure 4(a). We see that as the networks become denser, the average improvement compared to baseline decreases, approaching zero as networks become close to fully-connected: a random graph with an average density of 0.9 still does 5% better than a baseline network (which has a density of 1.0).

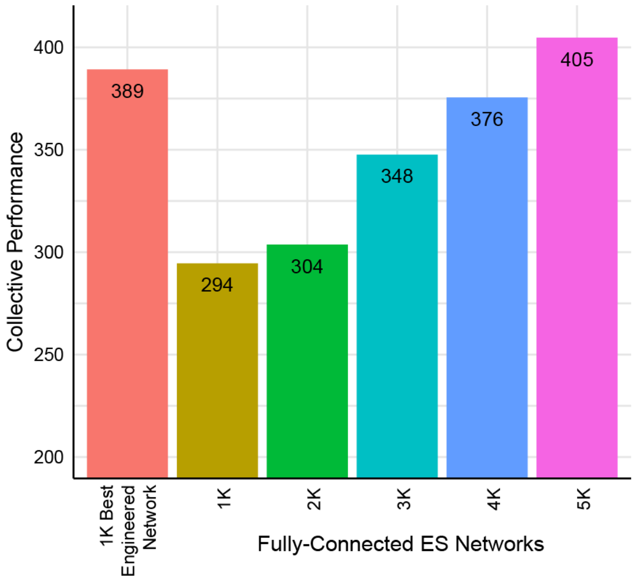

Figure 3: 1000 agents arranged in our best engineered network can beat up to 4000 agents arranged in a conventional fully-connected network.

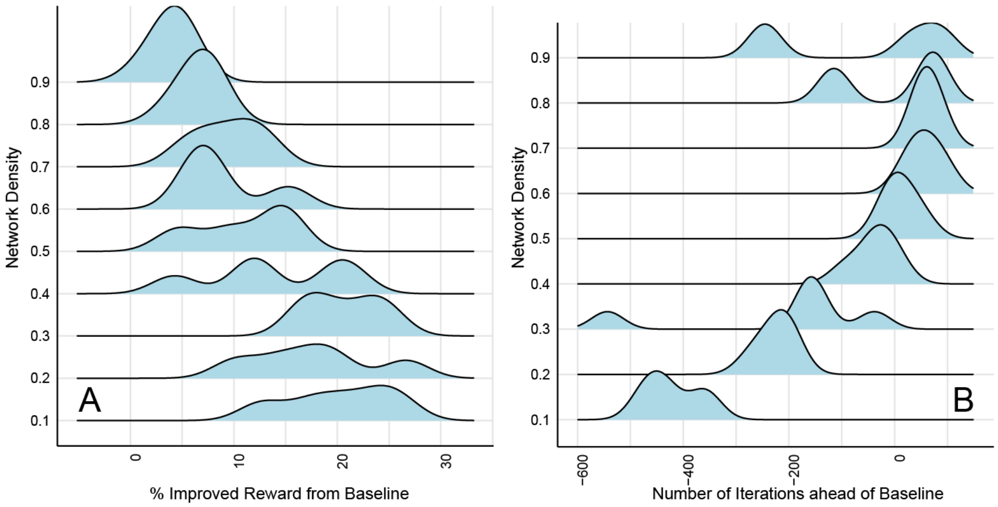

Figure 4: The distribution of reward (a) and learning rate (b) over several repeated runs of our algorithm varies strongly with the density of Erdos-Renyi networks (reward is calculated as the improvement from baseline; learning rate is defined as the number of iterations ahead of the fully-connected network to reach baseline reward)

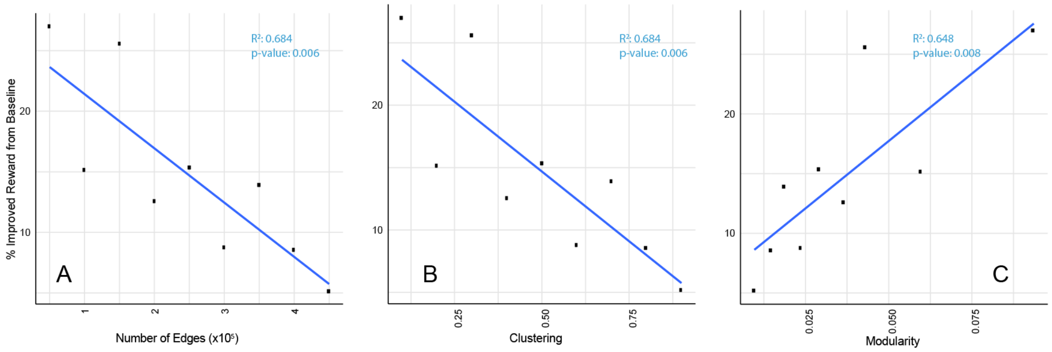

Figure 5: Erdos-Renyi networks can learn higher rewards with less communication costs (a); sparsity both at the local neighborhood level (b) and at the global inter-cluster level (c) leads to higher rewards.

Additionally, while the fully-connected networks take about 320 iterations to reach their asymptotic maximum result, our fastest network reaches that value in only 220 iterations (and keeps learning), an improvement of 32% (Figure 4(b)). Denser networks tend to learn faster, but the relationship is not monotonic: as the network approaches being fully connected, the distribution flattens and the average learning rate decreases.

This increase in speed could be due to the fact that the separate network neighborhoods of agents are able to visit a larger number of parameters in parallel, and hence can find higher maxima faster. Because we also implement a probabilistic broadcast, which sets the parameters of all agents to those of the highest-performing agent with probability β at the end of each iteration, we ensure that the network tends to converge to better-performing parameters. In short, our networked decentralization strikes a balance between increased parameter exploration diversity and global communication, similarly to Simulated Annealing [5]. As control, we tested a degenerate network where agents do not communicate with any other agents, except for broadcast, and find that no learning occurs. Additionally, we find that there is little variation in reward and speed beyond $\beta = 0.5$: the improvement is instead driven by the network structure.

To understand what causes certain specific network topologies to perform better, we calculated network metrics across all 1000 nodes in each Erdos-Renyi network. We find strong correlations between these network metrics and reward, as shown in Figure 5. Specifically, we find that as the number of edges (communication between agents) increases, the reward decreases (Figure 5(a)). This decline may be because, as communication increases, the local neighborhoods become less isolated from one another and the diversity of parameters being explored decreases. This, in turn, leads to lower rewards (closer to baseline). Clustering is a measure of how many of the neighbors of each node form a closed triangle, and is therefore a super-local measure of connectedness. We again find that as clustering increases, rewards decrease (Figure 5(b)). Modularity, a measure of inter-neighborhood global connectedness, also correlates with higher rewards (Figure 5(c)). Overall, we interpret these results to mean that sparser networks - at both the local and global level - can learn faster and achieve higher rewards than baseline, and with less communication cost (less dense networks have less edges, and hence lower communication between nodes).

Based on these observations, we then design new networks that push these network metrics to even higher (or lower) values to engineer for high-reward topologies. We focus on optimizing for higher rewards here and leave optimizing for faster learning to future work.

5.2 Improvements from Engineered Topologies

As seen in Figure 6(b), Erdos-Renyi graphs suggest that decreasing the number of edges would increase performance. Consequently, we engineered networks with an even smaller number of edges: the largest number of edges for engineered networks was smaller than Erdos-Renyi graphs (Figure 6(a)). As predicted, our engineered networks show increased rewards: 26% for the highest Erdos-Renyi compared to 33.5% for the best engineered graph. Interestingly, the relationship is non-monotonic: the trends in rewards with respect to the number of edges in Erdos-Renyi and engineered networks are opposite to one another. Perhaps under a certain threshold number of edges,

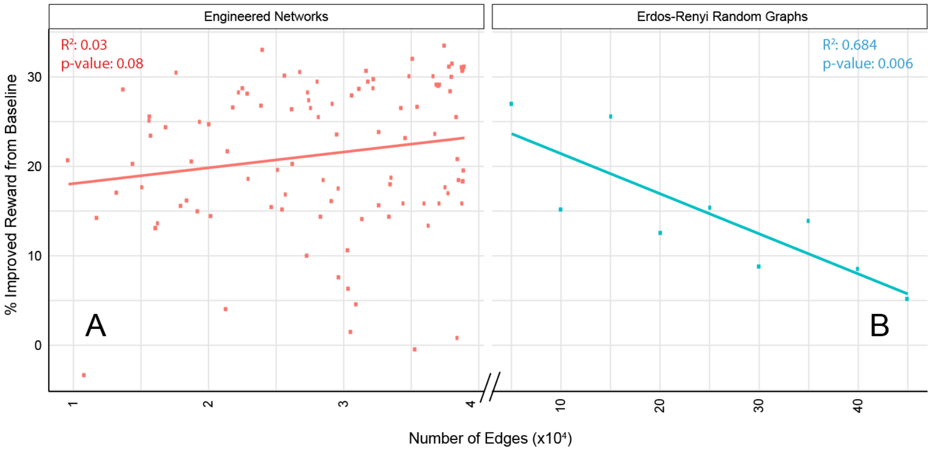

Figure 6: Erdos-Renyi graphs suggest that lower communication (edge counts) lead to higher reward (b) : we thus engineer graphs with even less communication (a) and find improvements, but with a non-monotonic behavior.

agents are no longer able to communicate efficiently within their thinned neighborhood and good gradients are not being communicated to neighbors who end up relying more on their very few neighbors' rewards, which in turn leads to ineffective search.

We find the same non-monotonic behavior for average path length, clustering (local connectedness), and modularity (global sparsity). Although our best engineered networks still do better than Erdos-Renyi graphs, rewards decrease if network connectedness decreases too much. In such cases, even extremely high broadcast probabilities do not allow such overly-thinned networks to learn. Note that the larger scattering variance in the generated network rewards is because we run each engineered network only once (to allow for our greater exploration of engineered topologies), instead of running repeated experiments for each topology, which we do for Erdos-Renyi and fully-connected networks.

From these explorations, we conclude that the best network structure is one that is globally and locally sparse: the network should consist of random graph clusters, each sparsely connected internally, with few connections between the clusters. Care should be taken not to create networks that are too sparse or else learning performance will suffer. Overall, it is clear that fully-connected networks are inefficient, learn more slowly and attain lower rewards than sparser networks.

6 Conclusion

In this work, we design a new networked evolutionary algorithm, informed by the literature on human collective intelligence, and report experimental results in using this algorithm to solve a benchmark deep reinforcement learning problem. We find the counter-intuitive result that sparser communication between learning agents can lead to faster and higher learning. Conventional wisdom would have suggested that the optimal network topologies would be close to fully-connected, and that diminishing returns would be found on either side of that optimum. We show that this optimum connectedness is actually very sparse, and that it is non-monotonic in reward. Future work could explore how other insights from the literature on human collective intelligence could further improve distributed learning algorithms, such as by experimenting with dynamic networks where nodes can be rewired at each iteration. The application of alternative network structures to other distributed deep learning algorithms, such as gradient-based algorithms, is another promising avenue for future work.

7 Acknowledgements

The authors would like to thank Zheyuan Shi, Abhimanyu Dubey, Otkrist Gupta, Stanislav Nikolov, Jordan Wick, Alia Braley and Ray Reich for their help and advice.

References

[1] Daniel Barkoczi and Mirta Galesic. Social learning strategies modify the effect of network structure on group performance. *Nature communications*, 7, 2016.

[2] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[3] Matthias Kahm, Guido Hasenbrink, Hella Lichtenberg-Fraté, Jost Ludwig, Maik Kschischo, et al. grofit: fitting biological growth curves with r. *Journal of Statistical Software*, 33(7):1–21, 2010.

[4] Stuart Kauffman and Simon Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology*, 128(1):11–45, 1987.

[5] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[6] Peter M Krafft, Julia Zheng, Wei Pan, Nicolás Della Penna, Yaniv Altshuler, Erez Shmueli, Joshua B Tenenbaum, and Alex Pentland. Human collective intelligence as distributed bayesian inference. *arXiv preprint arXiv:1608.01987*, 2016.

[7] David Lazer and Allan Friedman. The network structure of exploration and exploitation. *Administrative Science Quarterly*, 52(4):667–694, 2007.

[8] Winter Mason and Duncan J Watts. Collaborative learning in networks. *Proceedings of the National Academy of Sciences*, 109(3):764–769, 2012.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[12] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[13] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[14] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[15] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[16] Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. Evidence for a collective intelligence factor in the performance of human groups. *science*, 330(6004):686–688, 2010.