Fractal Neural Networks as Ensembles of Continuous-Valued Cellular Automata: Pathfinding and Related Problems of Variable Scale

Sam Earle

December 2, 2019

Abstract

We suggest that any agent playing a game with variable map-size should learn a map-size-invariant gameplay policy. Inspired by recent Reinforcement Learning environments with scalable maps—and scalable 2D observation and action spaces along with them—we consider a simple, scalable, pathfinding problem. Curious as to how trainable Neural Networks might come to solve such problems in a truly map-size-invariant manner, we outline an implementation of a path-finding Cellular Automoton as a recursive, convolutional NN, albeit with an atypical, periodic activation function. We note the structural equivalence between this manual implementation of a continuous-valued CA, and a single column of a Fractal Neural Network with weight-sharing. We consider whether FNNs might be able learn policies similar to the one encoded in our Neural CA. To guarantee agents access to coarse map-features, we propose an alternate, explicitly dimensionality-reducing FNN model.

1 Introduction

We propose a new desideratum for strategy-game-playing and related Reinforcement Learning agents. We expect any agent whose observation and action-spaces can be represented as $n-$dimensional grids to be able to transfer its skills to larger $n-$dimensional grids, insofar as they represent the same game at a different scale. Human Go players, for example, sometimes practice on smaller boards to hone elements of strategy that are applicable to games on larger boards. In popular strategy games like Civilization, various map-sizes are offered (and the player usually advised to start small). If an agent can be said to have developed a behavioural policy that properly represents the game-elements and the relations between them, then this policy should be effective at multiple scales.

Consider a simple example, in which an agent receives as its input a full view of a maze, represented as a connected subset of a 2D grid, and must colour the

shortest path between source and target nodes in the maze, edge by edge. A standard path-finding algorithm could solve this task regardless of maze size, but the neural network models used to encode the behavioural policy of the RL agents commonly applied to arcade and strategy games are not capable of scaling up their policy to act on input sizes larger than were trained upon without training of additional parameters.

We show, with a fixed and hand-weighted neural network which implements a continuous-valued Cellular Automoton—that sufficiently deep, recursive networks can solve a time-sensitive minimal spanning tree task at multiple scales. We are not necessarily interested in path-finding specifically, but take it as a good minimal example of a task that requires an agent to leverage spatial relationships between features of arbitrarily large maps. By implementing a CA that solves this task deterministically as a recursive NN, we intend to show that recursive NN architectures are a natural fit for agents trained to play strategy games at various board sizes.

In concurrent work (**??**), we introduce 3 RL environments that support scaling of the gameboard: a tile-by-tile board-filling task in Conway's Game of Life—with a randomly initialized array of cells—(**??**); a time-sensitive, randomized minimal-spanning-tree problem in SimCity1—framed in terms of power deliver—(**??**); and a general population-maximization problem in SC—in which the agent can build any zone/structure/land-feature—(**??**). In each, the agent has the full multi-channel 2D gameboard as its observation space, and the same 2D board with build-specific channels as its action space. In SimCity, it is natural to expect optimal "city planners" to be near-optimal "Regional Planners;" they both deal with the same problems of road- and power-network layout, and the spatial distribution of different zone types—relations that play out according to the same local rules at every map size (though they may contribute to different global dynamics at different scales, as is the case for extra-municipal demand for zone-by-type, where demand can fluctuate more intensely on larger maps with larger populations).

We use recursive neural networks to do 0-shot transfer to larger gameboards in our RL environments, with varying success (Figures **??** and **??**). In particular, scaling up the spanning-tree problem—by design—demands that the agent's network increase in depth, because the problem cannot be solved optimally without sufficiently large receptive fields. This motivates the use of NNs generated by a repeated 'fractal' expansion rule, and sharing a single set of weights; because these networks—by re-application of the expansion rule—can increase in depth and receptive-field size, in a natural and structurally coherent manner.

In pursuit of similarly transferable skills, Compositional Pattern Producing Networks have been used to generate indirect encodings of NNs—which scale with observation and action-space size—to do 0-shot transfer to octopus arms with more segments, in an octopus-arm control task [1]. Similarly, NerveNet [2] uses Graph NNs to to 0-shot transfer to 3D agents with more complex bodies (where limb connectivity is represented by a graph). It would be interesting to consider whether or how a CPPN could produce recursive, convolutional networks like those presented here.

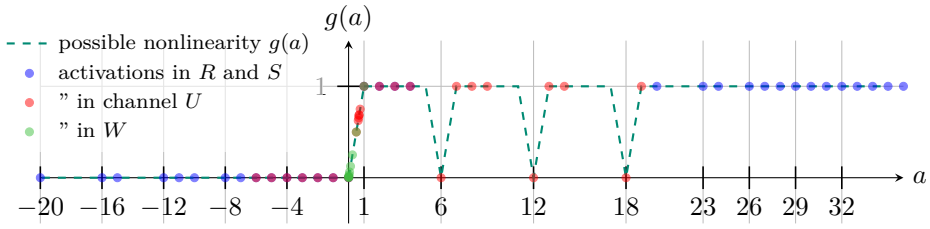

Figure 1: A sufficient activation function for our Neural CA, which performs computations only at the specified points.

2 Scalable, parallelized pathfinding

Cellular Automata have been used extensively for pathfinding [3, 4]. Slime mould has been used to solve mazes [5] and plan road networks between sources and sinks [6], the starved mould spreading out from its starting position toward nutrient sources. Its behaviour was subsequently modeled using CA [7].

In [8], CA have likewise been used for pathfinding, guiding a token through a 2D grid littered with obstacles toward targets, by moving in one of the four cardinal directions at each step. They use a CA to calculate the Manhattan distance, by 'flooding' the map with an activation that spreads out from the goal to path-adjacent tiles. With each tick, newly-activated tiles are given incrementally higher values than at the step previous, in order to track the time in which they were reached by the flood.

Our implementation was initially designed in order to generate a minimal-spanning tree, in order to solve an electricity-delivery problem over a sparse urban grid, and can thus have multiple sources (nodes in the spanning tree) and targets (nodes not in the spanning tree) at any particular step. In order to facilitate its implementation as an FNN, our CA simultaneously floods outward from sources and targets, and finds an ordering by manhattan distance of all tiles that are in the middle of some path from a source to a target.

2.1 Spanning Trees with Recursive Neural Networks

In appendix A, we define a CA capable of solving minimal-spanning-tree problems at multiple scales. We intentionally define the CA using more channels than necessary, to make it functionally identical to its neural implementation.

Our neural network is convolutional, recursive and and as deep as necessary for the CA it implements to complete its computation. It consists of one set of convolutional weights applied repeatedly to an activation state. This recursive body is book-ended by 1×1-kernel convolutional layers to map the agent's hidden activation to and from the output and input spaces, respectively. On the way in, existing channels of the board are copied to a few channels of activation space, while the rest is left empty.

We use the same nonlinear activation function ($g(a)$ in Figure 1) at each layer.

$g(a)$ can be seen as the combination of a hard tanh function with a periodic one. In general, some weights b_A feed activation from a 3×3 spatial patch, to its central node—and from channel A to channel B—of subsequent activation-maps. The values of particular weights will be given by:

$$b_A = \begin{bmatrix} b_{-1,1} & b_{0,1} & b_{1,1} \\ b_{-1,0} & b_{0,0} & b_{1,0} \\ b_{-1,-1} & b_{0,-1} & b_{1,-1} \end{bmatrix}$$

, where $m_{j,k}$ is the value of the weight leading from the node that is j and k cells over from the central node. The bias at neurons in B is denoted by b_{BIAS} and left equal to 0 unless otherwise specified.

First, let us define the following constant convolutional weights:

$$m_{\text{vn}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad m_{\text{c}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

. Then we characterize the channels R and S by their convolutional weights:

$$r_R = s_S = m_{\text{vn}} + 31(m_{\text{c}}) \quad r_S = s_R = -3(m_{\text{vn}}) \quad r_O = s_O = -4(m_{\text{c}})$$

, where the first rules allow the permanent spread of activation floods on their respective channels, the second ensures that these floods come to a halt upon meeting in space, and the third prevents floods from entering tiles occupied by an obstacle (where O is the fixed, binary obstacle channel). Both rules are affected by the hardtanh structure of the nonlinearity, using appropriately sized weights to 'skip' the periodic stretch as necessary.

The gate channel U is given by

$$u_R = 6(m_{\text{vn}}) \qquad u_{\text{BIAS}} = -6 \qquad u_U = -1/2(m_{\text{c}})$$
$$u_S = m_{\text{vn}}$$

. Using the periodic structure of the nonlinearity, when both floods enter the VN patch, an activation of 1 will arise (after the bias). After this cell is activated, it oscillates between 1/2 and 3/4, converging to 2/3, thus remaining semi-active to prevent a repeat activation of 1.

For the binary out-gate:

$$v_U = 4 \qquad v_{\text{BIAS}} = -3$$

, taking advantage of the fact that U has all $u_i \notin (\frac{3}{4}, 1]$

Our timekeeper W has only to start 'counting' when the gate fires:

$$w_V = m_{\text{c}} \qquad w_W = 1/2(m_{\text{c}})$$

. Finally, an output channel stores the earliest collision-site as a maximum value:

$$o_W = -1(m_c) \qquad\qquad o_{\text{BIAS}} = 1$$

. After a 1×1 embedding copying only this layer, its highest-valued tile is selected as the site of an edge-build.

3 Considerations for Learning Agents

The kind of activation function required by the above neural implementation is more complex than the simple ReLU activation used in our RL experiments and common in convolutional NN's. In particular, the periodic stretch's non-monotonicity could make it impossible to train weights via backpropogation [9]. We might still, however, entertain the possibility that a neuro-evolutionary algorithm could be able to replace the need for backpropogation—either by evolving weight-values (as does NEAT [10]) or ignoring them entirely [11].

In a similar vein, [12] extends NEAT with a mutation that switches one for another of a predetermined list of activation functions. But NEAT could also be applied to evolve a Compositional Pattern-Producing Network [13] (with 1 input and output dimension)—which, notably, are well-suited to generating periodic functions—to encode the activation function.

Or, more simply, we could hope that—by taking advantage of more channels to the hidden activation-state, and/or more densely populating weight-matrices than in our sparse manual implementation—the trained network could manage to approximate the behaviour of our CA by other means, in spite of a simpler non-linearity.

It is interesting to note that the activation function is only specified for a particular set of points on an interval of the real number line. Our implementation computes certain discrete values at a fixed number of points, along with certain convergent sequences. We could continue to implement additional CA rules in new channels, by following the existing strategy, and scaling and transforming input to the nonlinearity as necessary, so that it could co-occur with the existing rules, with computations taking place on a different subset of the activation function.

We should note that the use of convergent series may pose practical difficulties as the floating-point accuracy needed to differentiate certain activations grows exponentially with the number of ticks. Scaling activations and adding appropriate curvature to the activation function might help in combatting this issue.

Future work should examine if it is possible to create a mapping from any one of some subset of CAs, to a single set of convolutional weights (interpreted as the shared weights of a single, sufficiently-deep, recursive neural "column") and an accompanying activation function. If it is, we could gain a new level of abstraction in discussing the behaviour of trained or evolved recursive neural networks, by exploiting the inverse mapping to extract CAs from the network; or,

we could use the language of CAs as a new and automatic means for providing strong built-in priors to trainable NNs.

Even our manually-implemented CA could be run in parallel with a trainable model—a nontrainable extension of it, over which backpropogation does not occur—sharing its activations with the agent, and vice verse, with each concurrent tick. Such an approach would be similar in spirit to RL approaches that train agents to leverage auxiliary language models relating to the environment [14], though CA 'languages' are conceptually much lower-level than conventional ones.

4 Fractal Neural Networks

Fractal Neural Networks [15] were introduced to simplify ResNet's [16] residual structure. With respect to the overall structure of standard ResNet, residual blocks are replaced by fractal blocks, though dimension-reducing blocks remain unchanged. Fractal blocks result from a repeated expansion rule, which takes an existing network, copies it, and joins the copies to create a larger (twice as deep) network. The nature of the expansion rule ensures that shallower paths run through the network at various scales, providing the same benefits as residuals, but with a more succinct structure. In our concurrent work, we experiment with weight-sharing in FNNs, noting that with weight-sharing, the networks can be conceived of as ensembles of continuous-valued CA (**??**).

In a step toward likewise replacing the usual, max-pooling, down-/up-scaling blocks in Deep CNNs, we consider a new rule for fractal expansion that incorporates strided (downscaling) and transposed (upscaling) convolutions (Figure 2). This allows for the agent to naturally adjust the amount of dimensionality-reduction performed dependent upon map size in an organic way. In this rule, instead of using just a single convolution as a new skip connection, a single convolution is sandwhiched between pairs of down- and up-scaling convolutions—as many as the columns's index, and no more than possible for a given map-size. FNNs generated by this new expansion rule have shown comparable performance to standard FNNs in preliminary experiments in our scalable RL environments.

By considering these dimensionality-reducing FNNs as ensembles of subnetworks with student-teacher dynamics—as in the original paper—we see that networks with lower-dimensional state-spaces can learn from networks with higher-dimensional states, allowing us to scale to compute demands during inference by extracting subnetworks of varying complexity.

In neuroscience, Recurrent Fractal Neural Networks have been used to model the exchange of local and global information in visual input [17]. The interplay between coarse and fine-grained visual features is facilitated by a columnar fractal network structure which ensures the presence of short paths between neurons at various depths in the visual cortex. Our dimensionality-reducing columns make this interplay explicit, necessarily producing coarse representations—accounting for the possibility that deep columns of FNNs with fixed-dimension activation spaces learn fine-grained/localized features, their depth allowing global considerations without necessarily blurring local information; as is the case when deep,

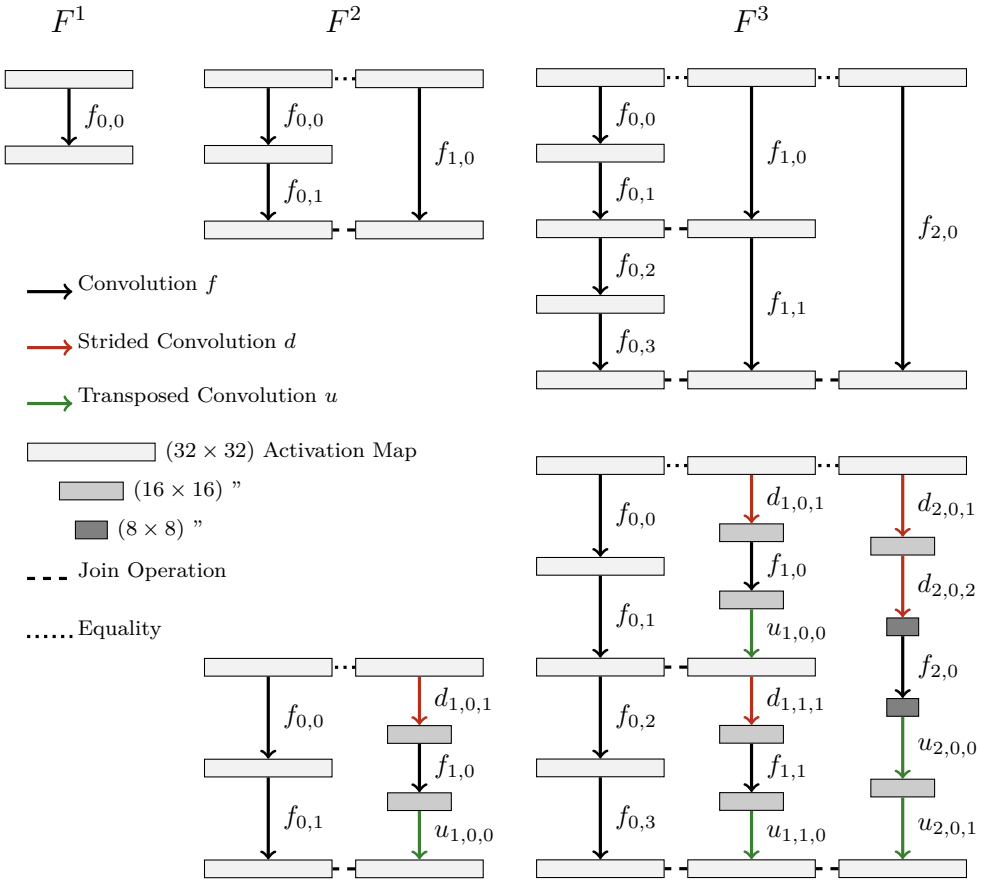

Figure 2: Fractal Neural Networks after the first three applications of the expansion rule for a regular FNN (above) and for a dimensionality-reducing FNN (below). With intra-column weight-sharing, then fixing only j (the column index), all $f_{j,k,l}$, $d_{j,k,l}$ and $u_{j,k,l}$ share weights amongst themselves. With inter-column weight-sharing, there is one set of weights f, d, u, for each.

recursive neural networks [18] with residuals [19] are trained to perform image super-resolution.

5 Future Work and Conclusion

The 'Squeeze' FNN has columns whose structure evoke the notion of 'strided' and 'transposed' CA. In such a CA, rules could be applied to patches of the board in stride, in order to compress it; or map smaller patches to larger ones in order to expand it. If such a CA is randomly initialized on an infinite plane, it can tick forever while scaling dimension in either direction.

A countercurrent of research has suggested that increasing the width of deep NNs is a viable alternative to increasing their depth [20]. A third expansion rule can be conceived, which copies the network and runs the copies in parallel; it sandwiches the original between a new pair of down-/up-scaling layers, with the copy acting as a skip connection between activations on either side of this new pair. After repetition, it results in a structure where skip connections traverse chains of downscaling-followed-by upscaling at various depths—between various pairs of activations of equal resolution (rather than just at the extremities, between full-size activations)—allowing the network to overlay coarse with fine-grained features at multiple scales.

We expect that requiring agents to perform at arbitrary map-sizes will narrow the task of learning generalizable representations of game elements. It is easy to imagine how most canonical RL strategy or arcade environments might scale to larger map sizes, in theory if not already in practice. With slight alteration of certain existing environments (strategy and board games in particular, in addition to the newly-introduced simulation-games discussed here), we could measure the map-size-invariance of policies learned by RL agents across a wide variety of tasks.

But to train agents capable of such 0-shot scaling, we need a scale-invariant, indirect encoding of the network. We thus propose the use of FNNs with weight sharing to encode sets of weights that can be combined (via fractal expansion) to form NN architectures of variable complexity. This agent-side complexity can then be scaled in tandem with map-size complexity, providing the agent with enough 'compute time'—or large enough receptive fields—to properly consider the entire map at each step.

Continuous-valued Cellular Automata are presented as a possible explanatory framework for scalable FNNs of this sort, while out our Neural CA could provide a baseline in the minimal-spanning-tree task for NN-based RL agents.

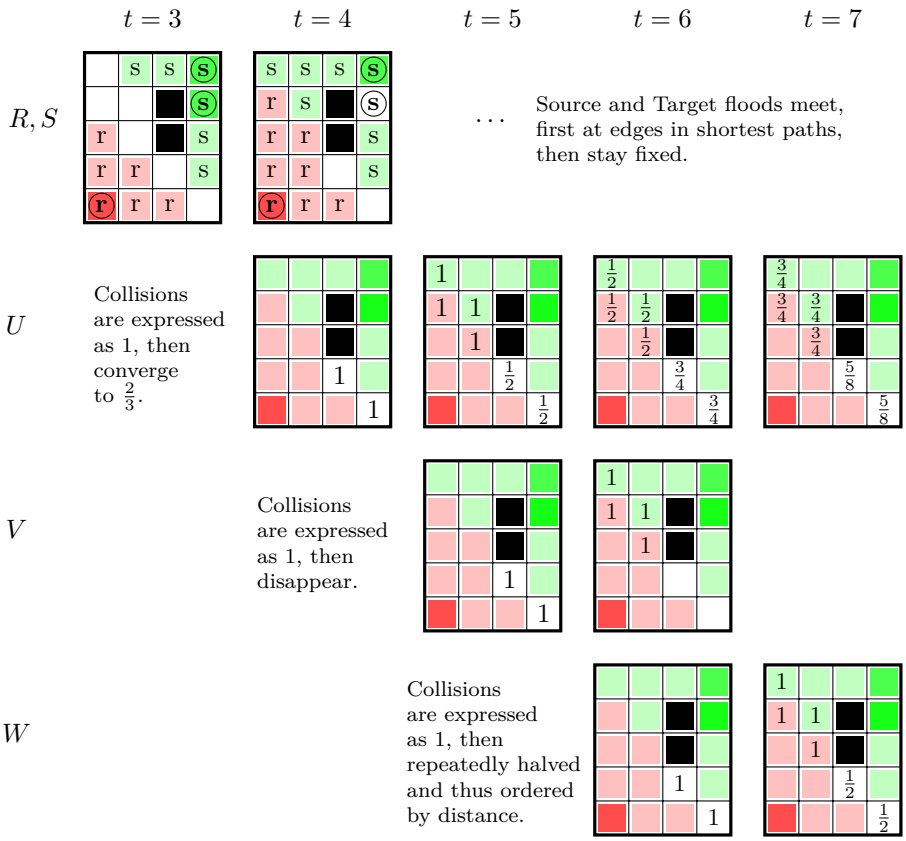

Figure 3: Various channels of our Neural CA unrolled across time. Unless indicated otherwise, all cells have activations of 0. Empty and unchanging channel-maps are omitted.

A Spanning Trees with Cellular Automata

We take our notation from [8]. Our automaton, before applying its ruleset, copies the board over to an array of the same spatial dimensions, with additional (all-0) channels. This representation will remain unchanged by the CA, and includes binary channels R', S' and O, encoding the presence of sources, targets, and obstacles, respectively. Assume as well that channels R' and S' are copied to new channels R and S during this embedding. The later two channels will produce floods of activation emanating from source and target cells, respectively. The (symmetric) transition rules for these channels are as follows:

$$r_i^{t+1} = \begin{cases} 1 & \text{if } r_i^t = 1 \vee (\exists x \in \eta_i | r_x = 1 \wedge \neg \exists x \in \eta_i | s_x = 1 \wedge o_i \neq 1) \\ 0 & \text{otherwise} \end{cases}$$

$$s_i^{t+1} = \begin{cases} 1 & \text{if } s_i^t = 1 \vee (\exists x \in \eta_i | s_x = 1 \wedge \neg \exists x \in \eta_i | r_x = 1 \wedge o_i \neq 1) \\ 0 & \text{otherwise} \end{cases}$$

where r_i^t, s_i^t are the states of the the cell i at timestep t in channels R and S, respectively, and η_i is the Von-Neumann neighbourhood of the cell i.

These floods of unit-activation on either channel will not overlap spatially; rather meeting to form a boundary, along which cells are halfway between each of some source-target pair.

Thus we define channels U and V, which together act as a gate, passing an activation at these boundary cells—as soon as their VN neighbourhood is occupied by activations from both floods—to W, which will proceed by halving the received activation repeatedly, in order to track the temporal order in which border-cells were discovered (and by extension, the lengths of the paths to which they belong).

The transition rule for the in-gate U is:

$$u_i^{t+1} = \begin{cases} 1 & \text{if } u_i^t = 0 \wedge \exists x, y \in \eta_i | s_x = 1 \wedge s_y = 1 \\ 1 - u_i/2 & \text{if } u_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

For any u_i, when floods collide in the patch η_i, then $u_i = 1$. At all following ticks, $u_i \in [1/2, 3/4]$, oscillating, to eventually converge to $2/3$.

The out-gate V has binary activation, with cells active only after being equal to 1 in U:

$$v_i^{t+1} = \begin{cases} 1 & \text{if } u_i^t = 1 \\ 0 & \text{otherwise} \end{cases}$$

Activations in V will thus appear at a node i two ticks after the floods have entered the neighbourhood η_i.

The timekeeper-channel W is characterized by the following rule:

$$w_i^t t + 1 = \begin{cases} 1 & \text{if } \exists x \in \eta_i | u_i = 1 \\ w_i^t/2 & \text{if } w_i^t > 0 \\ 0 & \text{otherwise} \end{cases}$$

. Once W is non-empty, the location of its least non-zero activation can be selected as the site for an edge-build—this edge being guaranteed to belong to the shortest path from the existing spanning tree to an unconnected node (or previous agent build).

References

[1] Brian G Woolley and Kenneth O Stanley. Evolving a single scalable controller for an octopus arm with a variable number of segments. In *International Conference on Parallel Problem Solving from Nature*, pages 270–279. Springer, 2010.

[2] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. 2018.

[3] Priyata Singhal and Harish Kundra. A review paper of navigation and pathfinding using mobile cellular automata. *Int. J. Adv. Comput. Sci. Commun. Eng.(IJACSCE)*, 2:43–50, 2014.

[4] P. Rosenstiehl, J.R. Fiksel, and A. Holliger. Intelligent graphs: Networks of finite automata capable of solving graph problems. In RONALD C. READ, editor, *Graph Theory and Computing*, pages 219 – 265. Academic Press, 1972.

[5] Andrew Adamatzky. Slime mold solves maze in one pass, assisted by gradient of chemo-attractants. *IEEE transactions on nanobioscience*, 11(2):131–134, 2012.

[6] Andrew Adamatzky and Jeff Jones. Road planning with slime mould: if physarum built motorways it would route m6/m74 through newcastle. *International Journal of Bifurcation and Chaos*, 20(10):3065–3084, 2010.

[7] Michail-Antisthenis I Tsompanas, Georgios Ch Sirakoulis, and Andrew Adamatzky. Cellular automata models simulating slime mould computing. In *Advances in Physarum Machines*, pages 563–594. Springer, 2016.

[8] C Behring, M Bracho, M Castro, and JA Moreno. An algorithm for robot path planning with cellular automata. In *Theory and practical issues on cellular automata*, pages 11–19. Springer, 2001.

[9] Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Taming the waves: Sine as activation function in deep neural networks. page 12, 2017.

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

[11] Adam Gaier and David Ha. Weight agnostic neural networks. *arXiv preprint arXiv:1906.04358*, 2019.

[12] Alexander Hagg, Maximilian Mensing, and Alexander Asteroth. Evolving parsimonious networks by mixing activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 425–432. ACM, 2017.

[13] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[14] Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions. *arXiv preprint arXiv:1906.00744*, 2019.

[15] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Erhard Bieberich. Recurrent fractal neural networks: a strategy for the exchange of local and global information processing in the brain. *Biosystems*, 66(3):145–164, 2002.

[18] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1637–1645, 2016.

[19] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3147–3155, 2017.

[20] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.