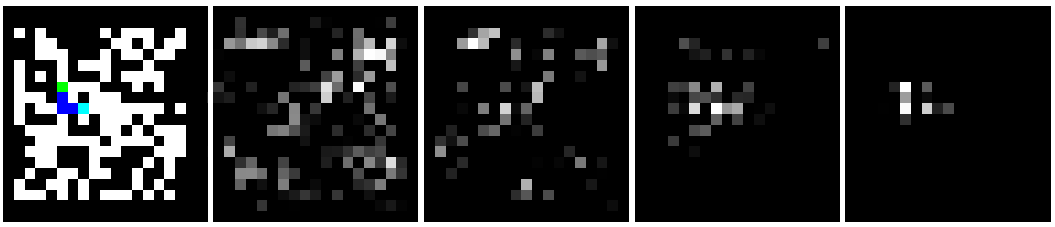
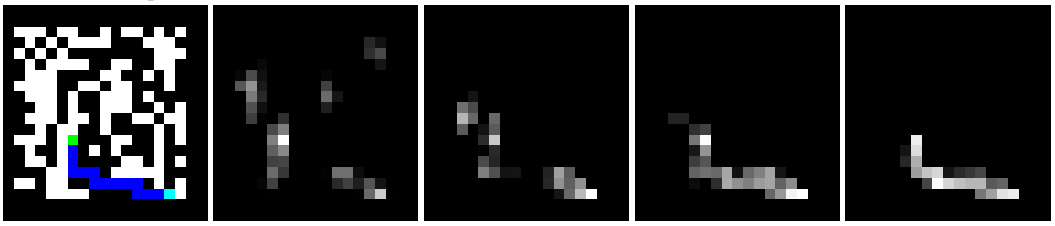
A Appendix

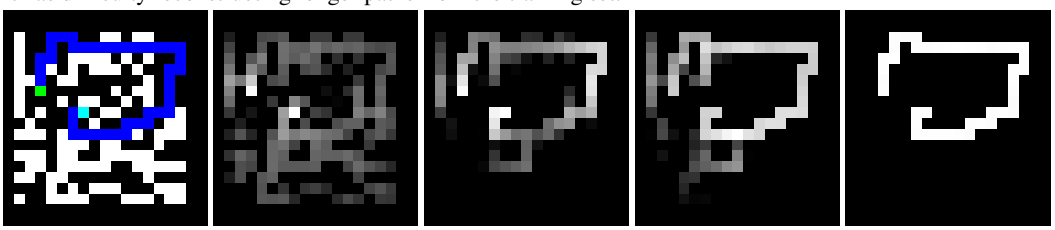
A.1 Qualitative Analysis



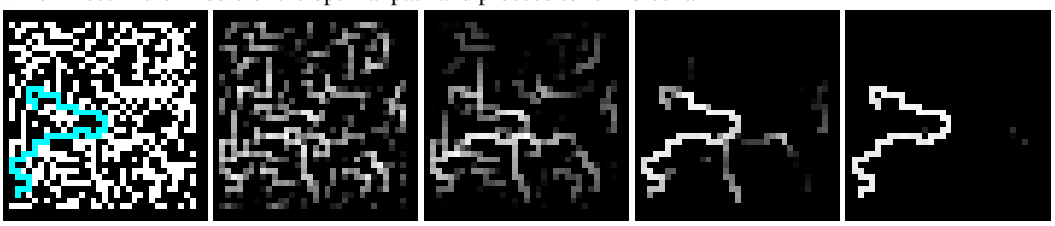
(a) MLP shortest-path behavior. The model reconstructs a small patch approximating the target path, but misses and misplaces a few tiles, and fails to maintain local coherence.



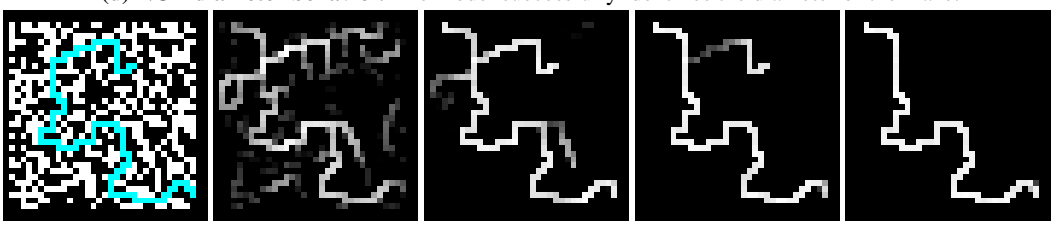
(b) GCN shortest-path behavior. The model produces a (somewhat blurry) reconstruction of the optimal path. It has difficulty reconstructing longer paths from the training set.



(c) NCA shortest-path behavior. The model appears to propagate activation out from both source and target, which meet in the middle of the optimal path and proceed to reinforce it.



(d) NCA diameter behavior. The model successfully identifies the diameter of the maze.



(e) NCA diameter behavior. The model selects the incorrect (shorter) upper fork.

Figure 2: Behavior of various models on shortest-path/diameter path-finding problems. GCN and MLP models are seen to fail frequently on simpler mazes from the training set for the shortest-path problem. NCAs can master the most complex mazes from the training set while sometimes generalizing to larger test mazes, on both the shortest-path and (more challenging) diameter problem.

A.2 HAND-CODED NCA

In discussion of the hand-coded implementations, we denote $W_k^{(m,n)}$ as the $(m,n)^{\text{th}}$ element of the weight matrix $\mathbf{W}_k$. We will define a number of elementary convolutional weight matrices, then specify how they are combined in a single convolutional layer to propagate activation over various channels. Finally, we will specify activation functions and skip connections applied to each forward pass through this convolutional layer. Initially, the hidden activation is all-zeros. We assume that, prior to each pass through the NCA, the hidden activation is concatenated channel-wise with a one-hot encoding of the input maze.

A.2.1 DIJKSTRA MAP GENERATION

In Dijkstra map generation, we define three channels: flood_s, flood_t and age.

Convolutional weights. We define an identity-preserving convolutional weight matrix $\mathbf{W}_1 \in \mathbb{R}^{3\times3}$ with $\mathbf{W}_1^{(1,1)} = 1$; otherwise, $\mathbf{W}_1^{(k,l)} = 0$ so only center tile of the weight matrix is active. This matrix is used to keep a channel active once it is activated (e.g., once a tile is flooded, it needs to stay flooded). It is also used to produce the initial activation in the flood channels from source and target channels, and prevent tiles with active wall channels (i.e. walls in the maze) from being flooded.

Then, we define $\mathbf{W}_T \in \mathbb{R}^{3\times3}$, which is the Von Neumann neighborhood, with $\mathbf{W}_T^{(k,l)} = 1$ if $(k,l) \in \{(0,1),(1,0),(1,1),(1,2),(2,1)\}$; otherwise, $\mathbf{W}_T^{(k,l)} = 0$ and it's used to flood the tiles next to flood since one can only move in this four directions.

$$\mathbf{W}_1 \coloneqq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{W}_T \coloneqq \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

In Table 7, we use the elementary weight matrices $\mathbf{W}_1$ and $\mathbf{W}_T$ to define the weight matrices between channels in the convolutional layer of which our hand-coded Dijkstra activation map-generating NCA consists.

Convolutional slice	Channel relationship	Weight matrix
$\mathbf{W}_{\text{BFS}}[\text{flood}, \text{source}]$	source $\to \text{flood}_s$	$\mathbf{W}_1$
.	$\text{flood}_s \to \text{flood}_s$	$\mathbf{W}_T$
.	wall $\to \text{flood}_s$	$-6\mathbf{W}_1$
.	target $\to \text{flood}_t$	$\mathbf{W}_1$
	$\text{flood}_t \to \text{flood}_t$	$\mathbf{W}_T$
	wall $\to \text{flood}_t$	$-6\mathbf{W}_1$
	$\text{flood}_s \to$ age	$\mathbf{W}_1$
	$\text{flood}_t \to$ age	$\mathbf{W}_1$
	age $\to$ age	$\mathbf{W}_1$

Table 7: hand-coded weights between channels of NCA for **Dijkstra map generation**.

The overall convolutional layer $\mathbf{W}_{\text{BFS}} \in \mathbb{R}^{3\times7\times3\times3}$ consists of the channel-to-channel kernels given in Table 7. For each 3×3 patch of the (padded) input activation, it maps to the next value at the center cell; it maps from the 3 hidden activation channels ($\{\text{age}, \text{flood}_s, \text{flood}_t\}$) plus the 4 channels required for one-hot encoding the maze $\{\text{empty}, \text{wall}, \text{source}, \text{target}\}$—which is concatenated with the input activation at each step—to the 3 hidden channels at the next step.

The floods flood_s and flood_t flow from tiles with active source and target tiles, respectively (i.e. sources/targets in the input maze), to empty tiles, stopping at walls. While the age channel will first activate with the appearance of a flood channel, then increment at each following timestep. Once the separate floods meet, the activation map $\mathbf{X}^{\text{age}}$ can then be used to reconstruct optimal paths (which process is detailed in the path extraction NCA in the following section).

Forward pass. At each step through the hand-coded BFS-NCA, we apply the convolutional weights to the input activation, then apply a step function to the flood channel, so that its output lies between 0 and 1. (Since the flood activation is always integer-valued before being input to the activation function, this could also be achieved with duplicate channels, using ReLU's, with biases to off-set them, effectively resulting in a step function that is linear for $x \in [0, 1]$. For simplicity, we simply apply a step function, but note that a ReLU network could represent the same algorithm.)

$$\mathrm{step} = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Let $\mathbf{X}_t$ denote the hidden activation at time-step t, with $\mathbf{X}_t \in \mathbb{R}^{3 \times w \times h}$ (where w and h are the width and height of the input maze), and $\mathbf{X}_t = \mathbf{0}$ when $t = 0$; and let X_{maze} denote the one-hot encoding of the input maze, with $\mathbf{X}_{\mathrm{maze}} \in \mathbb{R}^{4 \times w \times h}$. We denote by $\mathbf{X}^{i,j}$ the slice of the tensor $\mathbf{X}$ taken at spatial co-ordinates i, j (where these spatial coordinates correspond to the last two dimensions of the tensor), and by $\mathbf{X}^{\mathrm{channel}}$ the slice of the tensor $\mathbf{X}$ corresponding to the channel $\mathrm{channel}$ (where channels lie along the first dimension of the tensor). The $\mathrm{concatenate}$ operation acts along the channel dimension. To denote setting the value of an activation $\mathbf{X}$ at channel chan, we write $\mathbf{X}^{\mathrm{chan}} \leftarrow \cdots$; generally, this operation is individually applied to each (x, y) tile of $\mathbf{X}$. At time-step t the forward pass operates as follows:

$$\begin{aligned} \mathbf{X}_t &\leftarrow \mathrm{concatenate}\left(\mathbf{X}_t, \mathbf{X}_{\mathrm{maze}}\right) \\ \forall x, y,\ \mathbf{X}_{t+1}^{x,y} &\leftarrow \sum_{i=0}^{2} \sum_{j=0}^{2} \mathbf{W}_{\mathrm{BFS}}^{i,j} \odot \mathbf{X}_t^{x+i-1, y+j-1} \\ \mathbf{X}_{t+1}^{\mathrm{flood}} &\leftarrow \mathrm{step}\left(\mathbf{X}_{t+1}^{\mathrm{flood}}, 0, 1\right) \end{aligned}$$

The BFS-NCA will terminate when the source and target floods overlap on some tile.

A.2.2 Path Extraction

We now construct an NCA for path extraction (PE-NCA) which will take the output of the BFS-NCA and additionally perform path extraction. Path extraction starts once source and target floods, which are defined in $\mathrm{Dijkstra}$ collides. We use the age activations from $\mathrm{Dijkstra}$ output to reconstruct the optimal path(s) between the corresponding source and target.

Convolutional weights. For path extraction, we add 5 new channels: path channel, which will ultimately output the binary map of the optimal path(s), and 4 directional path-activation channels (for detecting the presence of a path coming from the right, left, top, and bottom neighbors) denoted as $\mathrm{path}_{i,j}$ for $(i, j) \in \{(0, 1), (1, 0), (1, 2), (2, 1)\}$. Then, we define weight matrices for detecting adjacent activations: $\mathbf{W}_{(i,j)} \in \mathbb{R}^{3 \times 3}$, with $W_{(i,j)}^{(k,l)} \coloneqq 1$ if $(k, l) = (i, j)$, and $W_{(i,j)}^{(k,l)} \coloneqq 0$ otherwise.

$$\mathbf{W}_{1,0} \coloneqq \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{0,1} \coloneqq \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{1,2} \coloneqq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{2,1} \coloneqq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

We can then construct the overall convolutional path extraction matrix $\mathbf{W}_{\mathrm{PE}}$, with $\mathbf{W}_{\mathrm{PE}} \in \mathbb{R}^{5 \times 8 \times 3 \times 3}$ using the channel-to-channel weight matrices detailed in Table 8 (and setting all other weights to 0). For each 3×3 patch of the (padded) input activation, it maps to the next value at the center cell; it maps from its own 5 hidden activation channels (the 4 $\mathrm{path}_{i,j}$ activations, and the overall path activation) plus the 3 hidden channels resulting from the BFS-NCA—whose output is concatenated with the input activation at each step—to the 5 hidden channels at the next step.

The identity weights from flood_s and flood_t to path, combined with the bias of -1 applied to path, ensure that path activations will first appear on any tiles where these floods have overlapped (seeing as these tiles will correspond to mid-points in the optimal path(s)). The weights from age and path

Channel relationship	Weight matrix	Bias
$\text{flood}_s \rightarrow \text{path}$	$\mathbf{W}_1$	
$\text{flood}_t \rightarrow \text{path}$	$\mathbf{W}_1$	
path		-1
$\text{age} \rightarrow \text{path}_{i,j}$	$2(\mathbf{W}_{(i,j)} - \mathbf{W}_1)$	
$\text{path} \rightarrow \text{path}_{i,j}$	$\mathbf{W}_{(i,j)}$	

Table 8: hand-coded weights between channels of NCA for **path extraction**.

to each $\text{path}_{\text{i,j}}$ are chosen so that when a path activation should "flow" to an adjacent tile in a given direction—which is the case exactly when this neighbor's age is greater than the current cell's by 1—the corresponding $\text{path}_{i,j}$ activation will be exactly -1.

Forward pass. At each step through the PE-NCA, we apply the convolutional weights and biases corresponding to path extraction—$\mathbf{W}_{\text{PE}}$ and $\mathbf{b}_{\text{PE}}$, respectively. Then, we apply a saw-tooth activation function to the directional path-activation channels. (Similar to the step function in the forward pass of the BFS-NCA, this sawtooth activation could be replicated with 3 ReLU's, and corresponding additional channels.)

$$\text{sawtooth}_a(x) = \begin{cases} 0 & \text{for } x < -2 \\ x + a - 1 & \text{for } x \in [a-1, a] \\ -x + a + 1 & \text{for } x \in [a, a+1] \\ 1 & \text{for } x > a + 1 \end{cases}$$

Since the directional path-activations are always integer-values, we can apply sawtooth_{-1} to them in order to obtain activations of 1 wherever they have value -1 (i.e., they should accept an adjacent path activation and are part of the optimal path by virtue of their age-difference with their path-activated neighbor), and 0 everywhere else.

Then, we set the path channel to be 1 if any of the directional path activations at the corresponding tile are equal to 1. This can be achieved by taking the sum of the directional path activations (which are either 0 or 1 after the sawtooth activation), and applying a step function.

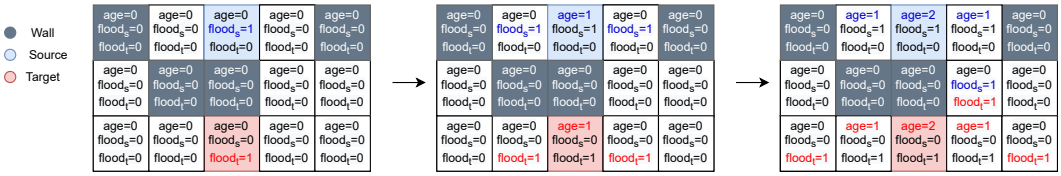
$$\begin{aligned} \mathbf{X}_t &\leftarrow \text{concatenate}\left(\mathbf{X}_t, \mathbf{X}_{\text{BFS}}\right) \\ \forall x, y,\ \mathbf{X}_{t+1}^{x,y} &\leftarrow \sum_{i=0}^{2}\sum_{j=0}^{2} \mathbf{W}_{\text{PE}}^{i,j} \odot \mathbf{X}_t^{x+i-1,y+j-1} + \mathbf{b}_{\text{PE}} \\ \forall i, j,\ \mathbf{X}_{t+1}^{\text{path}_{i,j}} &\leftarrow \text{sawtooth}_{-1}\left(\mathbf{X}_{t+1}^{\text{path}_{i,j}}\right) \\ \mathbf{X}_{t+1}^{\text{path}} &\leftarrow \text{step}\left(\sum_{(i,j)} \mathbf{X}^{\text{path}_{i,j}}\right) \end{aligned}$$

We illustrate the step-by-step operation of the path extraction NCA in Figure 3b.

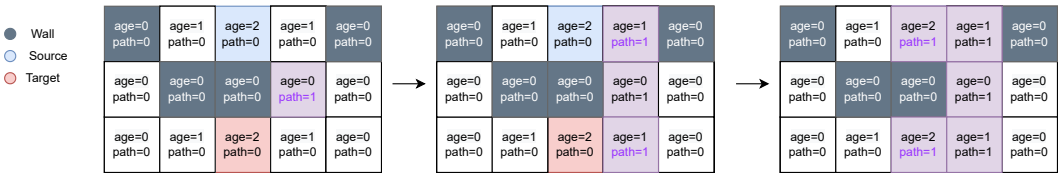
A.2.3 DFS

Convolutional weights. Let us again denote an identity-preserving convolutional weight matrix $\mathbf{W}_2 \in \mathbb{R}^{5\times5}$ with $W_2^{(2,2)} = 1$ (at the center of the weight matrix) and $W_2^{(k,l)} = 0$ everywhere else. $\mathbf{W}_2$ is similar to $\mathbf{W}_1$, in that only the center tile is active, allowing for the transfer of activations between channels, the inhibition of one activation by another (i.e. having wall activations prevent the activation of route or stack channels), and the holding-constant of activations across time-steps.

Then, we define the adjacent-activation weight matrices $\mathbf{W}_{5,(i,j)} \in \mathbb{R}^{5\times5}$ for $(i,j) \in \{(1,2),(2,1),(3,2),(2,3)\}$, which are used to check whether neighbors have any adjacent route



(a) Demonstration of hand-coded Dijkstra map generation by flooding from both source and target. The blue color age and floor activation are due to source and red one is due to target. The map generation is completed when both flood channels are active.



(b) Demonstration of hand-coded path-extraction by using Dijkstra map. The first activation of the path is where source and target flood meets. The purple color demonstrates the current path activation tile. The path extraction is completed when path channel of source and target is activated.

Figure 3: Demonstration of hand-coded NCA for a) Dijkstra map generation, b) path extraction

activations, with $W^{(k,l)}_{5,(i,j)} = 1$ for $(k,l) = (i,j)$, and $W^{(k,l)}_{5,(i,j)} = 0$ otherwise. Just as the identity-preserving matrix $\mathbf{W}_2$ is equivalent to $\mathbf{W}_1$, plus a border of 0-padding, so is each matrix $\mathbf{W}_{5,(i,j)}$ to the corresponding 3×3 adjacent-activation matrix $\mathbf{W}_{(i,j)}$.

In DFS, we propagate the route activation (equivalent to flood in BFS) sequentially. To this end, we combine the adjacent-activation matrices above with priority weight-matrices. These priority weights are the only matrices to make use of our 5×5 kernel; their job is to detect whether—given that some neighbor of a given cell has an active route which should be propagated to said cell—there is not some neighbor's neighbor to whom this activation should be propagated first.

For moving down, the priority weight matrix is defined as $\mathbf{W}_{p,(1,2)} \in \mathbb{R}^{5\times5}$ and it is equal to $\mathbf{0}$ since moving down is the highest-priority move. Then, in order to check the priority of moving right, we define $\mathbf{W}_{p,(2,1)} \in \mathbb{R}^{5\times5}$, with $W^{3,1}_{p,(2,1)} = 1$; and 0 everywhere else. This matrix effectively checks whether the neighbor to our left has some neighbor below it to whom it could pass its 'route' activation, in which case this cell should take priority over us. Similarly, we define $\mathbf{W}_{p,(3,2)} \in \mathbb{R}^{5\times5}$ for the upward move priority, with $W^{m,n}_u := 1$ if $(m,n) \in \{(4,2),(3,3)\}$, and 0 otherwise. This matrix is used to check whether the neighbor above us as a neighbor either below or to its right to whom it should pass activation first. Finally, moving left is the least prioritized action, and is represented with $\mathbf{W}_{p,(2,3)} \in \mathbb{R}^{5\times5}$, where $W^{(m,n)}_{p,(2,3)} = 1$ if $(m,n) \in \{(1,3),(2,4),(3,3)\}$, and 0 otherwise. This matrix only sends activation from the neighbor to a cell's right if this neighbor has no cells below, to the right, or above it, which would then take priority over us.

$$\mathbf{W}_{p,(2,1)} := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{p,(3,2)} := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{p,(2,3)} := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

When the route activation is *not* propagated to some adjacent, available neighbor as a result of one of the priority-rankings as defined above, we add a binary stack activation to the ignored tile. We then count the age of this stack activation with the stack_rank channel. When the pebble becomes stuck, the least stack_rank activations will correspond to the edges that the pebble has ignored most recently. Since, if the pebble ignores multiple edges at a given iteration (e.g., by moving downward when both right and left neighbors are also available), there will be multiple equivalent stack_rank activations, we further encode their directional priority on the stack via the stack_direction channel.

Chanel relationship	Weight
source $\rightarrow$ route	$\mathbf{W}_2$
route $\rightarrow$ route	$\mathbf{W}_2$
wall $\rightarrow$ route	$-\mathbf{W}_2$
stack $\rightarrow$ route	$-\mathbf{W}_2$
route $\rightarrow$ route$_{i,j}$	$\mathbf{W}_{5,(i,j)}$
route $\rightarrow$ route$_{i,j}$	$\mathbf{W}_{p,(i,j)}$
empty $\rightarrow$ route$_{i,j}$	$-\mathbf{W}_{p,(i,j)}$
source $\rightarrow$ route$_{i,j}$	$-\mathbf{W}_{p,(i,j)}$
target $\rightarrow$ route$_{i,j}$	$-\mathbf{W}_{p,(i,j)}$
pebble $\rightarrow$ stack	$\mathbf{W}_a$
stack $\rightarrow$ stack	$\mathbf{W}_2$
wall $\rightarrow$ stack	$-2\mathbf{W}_2$
route $\rightarrow$ stack	$-2\mathbf{W}_2$
pebble $\rightarrow$ stack_direction	$\mathbf{W}_p$
stack_direction $\rightarrow$ stack_direction	$\mathbf{W}_2$
wall $\rightarrow$ stack_direction	$-2\mathbf{W}_2$
route $\rightarrow$ stack_direction	$-2\mathbf{W}_2$
pebble $\rightarrow$ stack_rank	$\mathbf{W}_2$
stack_direction $\rightarrow$ stack_rank	$\mathbf{W}_2$
wall $\rightarrow$ stack_rank	$-2\mathbf{W}_2$
route $\rightarrow$ stack_rank	$-2\mathbf{W}_2$
stack $\rightarrow$ stack_rank	$\mathbf{W}_2$
pebble $\rightarrow$ since_binary	$\mathbf{W}_2$
since_binary $\rightarrow$ since_binary	$\mathbf{W}_1$
since_binary $\rightarrow$ since_binary	$\mathbf{W}_1$
since $\rightarrow$ since	$\mathbf{W}_1$

Table 9: hand-coded weights between channels of NCA for **DFS**.

For the priority of the direction, we define $\mathbf{W}_p \in \mathbb{R}^{5\times5}$ and $W_p^{1,2} = 0.2$, $W_p^{2,1} = 0.4$, $W_p^{3,2} = 0.6$, $W_p^{2,3} = 0.8$; otherwise, $W_p^{k,l} = 0$. Then, we define $\mathbf{W}_a \in \mathbb{R}^{5\times5}$ to check the neighbor tiles to follow the 'pebble' and $W_a^{(m,n)} = 1$ if $(m,n) \in \{(1,2),(2,1),(3,2),(2,3)\}$; otherwise, $W_a^{(m,n)} = 0$.

$$\mathbf{W}_p \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{4}{5} & 0 & \frac{2}{5} & 0 \\ 0 & 0 & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{W}_a \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that in the event a cell which is already "on the stack" is added to it again (i.e., the pebble passes by it at two neighboring cells, without having moved onto it), the stack_rank and stack_direction activations at this tile are effectively reset or overwritten, to correspond to this more "recent" edge in terms of the pebble's traversal over the graph.

Having defined these elementary weights, we give the full specification of the hand-coded DFS-NCA in Table 9. The result is a convolutional weight matrix $\mathbf{W}_{\text{DFS}} \in \mathbb{R}^{3\times3}$

The source tile will result in the initial route activation. Once a tile has an active route, it will be maintained at following timesteps. wall and stack activations will inhibit the appearance of a route activation. Directional route activations will appear if the tile is empty and if any of the higher priority neighbors in $\mathbf{W}_{p,(i,j)}$ are available (that is, they are empty, source or target tiles, without any existing route activation already present on them).

If the pebble passes by a some tile at any of its neighbors (given by $\mathbf{W}_a$), this tile is added to the stack, with the stack activation sustaining itself across following timesteps. This stack activation

is nullified, however, if there is a wall on the current tile, or is received route activation at the next iteration. (A stack activation thus only becomes active when the pebble "ignores" an available tile as a result of directional priority.)

The $\mathrm{stack\_direction}$ channel is similarly self-sustaining, and is activated when the pebble passes by the current tile without having been inhibited by a wall and without moving onto the tile at the next iteration, though this time it is activated according to the directional priority matrix $\mathbf{W}_p$.

The $\mathrm{stack\_direction}$ channel is, again, self-sustaining and activated when ignored by a pebble. Additionally, it increments at each following time-step as long as a stack remains present at the current tile.

Forward pass. We apply a step function to the directional $\mathrm{route}_{i,j}$ activation channels. The result will indicate whether, given the presence of adjacent route activations, the priority allotted to any relevant and available second-neighbors, and the presence of any wall or stack activations, route activation if flowing into a given tile in a given direction. As a first step toward, determining the route activation at a given tile, we take the sum of the directional route-activations at this tile, and apply a step function to the result. This will indicate whether route activation is flowing into this tile from *any* direction. We then add the result to the route activation and take the step function again: this will ensure that any blocking wall or stack activations prevent the final route activation.

We apply ReLU functions to the stack, stack rank, and stack direction channels. To address the event of a given tile being re-added to the stack before being popped from it, we apply a sawtooth to detect tiles with a stack activation equal to 2, resulting in a binary array, $\mathbf{dbl\_stacked} \in \{0,1\}^{w\times h}$. To overwrite the old $\mathrm{stack\_rank}$ and $\mathrm{stack\_direction}$ activations (and reset the stack activation to 1), we subtract from each of these channels the product of their value at the preceding time-step $(t-1)$ with $\mathbf{dbl\_stacked}$. For the $\mathrm{stack\_rank}$ channel, we must subtract its previous value $+1$, since its $\mathrm{stack\_rank}$ has incremented during the most recent convolution (as occurs at each time-step) in addition to having increased as a result of its new position at the top of the stack.

The pebble channel is given by the difference between the route activations at the current time-step and those at the previous time-step, so that there is at most a single pebble on the board, in the place of the single newly-added route activation. If the pebble is stuck, there is no pebble activation, so we take the spatial max-pool over the pebble channel and apply a $\mathrm{sawtooth}_0$ to obtain a binary value corresponding to the pebble's being stuck.

In case the pebble is stuck, we need to know which tile on the stack to pop next, taking into account both the directional priority and time spent on the stack of each tile. We thus calculate the $\mathbf{total\_rank}$ of each tile (stored as a temporary variable, with $\mathbf{total\_rank} \in \mathbb{R}^{w\times h}$) by taking the sum of its $\mathrm{stack\_rank}$ and $\mathrm{stack\_direction}$ channels. The tile with least non-zero $\mathrm{total\_rank}$ is the next tile to be popped. We first replace all 0 activations in the $\mathrm{total\_rank}$ array by adding to this array the result of $\mathrm{step}_0(\mathbf{total\_rank}$ multiplied by a large integer L (i.e. the maximum value of $\mathrm{stack\_rank}+1$ Then, we take the minimum over the resulting $\mathbf{total\_rank}$: this is the $\mathrm{min\_total\_rank} \in \mathbb{Z}_0$ (the same could be achieved by flipping signs and using a max-pooling layer). (Note that if $\mathrm{min\_total\_rank} = L$, then the stack is empty, we are effectively done, and the following steps will have no effect.)

Wherever the difference $\mathrm{total\_rank} - \mathrm{min\_rank}$ is 0, we must pop from the stack, provided also that the pebble is stuck. Thus, to obtain a binary array $\mathbf{is\_popped} \in \{0,1\}^{w\times h}$ with a 1 in at the one tile to be popped (if any), we take the product of $\mathrm{sawtooth}_0$ applied to the above difference, with the binary variable indicating the pebble's being stuck. Finally, to release the given tile from the stack, we subtract from the stack, $\mathrm{stack\_rank}$ and $\mathrm{stack\_direction}$ channels their product with $\mathrm{is\_popped}$, zeroing-out these channels and dis-inhibiting the flow of route activation to these tiles at following time-steps.

We note that since channel activation happens when pebble triggers the stack channel so pebble is next to this node. The reason of the since channel is to count how many iterations passed after the node is added to the stack. Therefore, since channel increases by 1 after the activation.

$$\mathbf{X}_t \leftarrow \text{concatenate}\left(\mathbf{X}_t, \mathbf{X}_{\text{maze}}\right)$$

$$\forall x, y,\ \mathbf{X}_{t+1}^{x,y} \leftarrow \sum_{i=0}^{2}\sum_{j=0}^{2} \mathbf{W}_{\text{DFS}}^{i,j} \odot \mathbf{X}_t^{x+i-1,y+j-1}$$

$$\forall i, j,\ \mathbf{X}_{t+1} \leftarrow \text{step}\left(\mathbf{X}_{t+1}^{\text{route}_{i,j}}\right)$$

$$\mathbf{X}_{t+1}^{\text{route}} \leftarrow \text{step}\left(\mathbf{X}_{t+1}^{\text{route}} + \text{step}\left(\sum_{(i,j)} \mathbf{X}^{\text{route}_{i,j}}\right)\right)$$

$\forall \text{chan} \in \{\text{stack}, \text{stack\_rank}, \text{stack\_direction}\},$

$$\mathbf{X}_{t+1}^{\text{chan}} \leftarrow \text{ReLU}\left(\mathbf{X}^{\text{chan}}\right)$$

$$\mathbf{dbl\_stacked} \leftarrow \text{sawtooth}_2\left(\mathbf{X}^{\text{stack}}\right)$$

$\forall \text{chan} \in \{\text{stack}, \text{stack\_direction}\},$

$$\mathbf{X}_{t+1}^{\text{chan}} \leftarrow \mathbf{X}_{t+1}^{\text{chan}} - \mathbf{X}_t^{\text{chan}} \odot \mathbf{dbl\_stacked}$$

$$\mathbf{X}_{t+1}^{\text{stack}} \leftarrow \mathbf{X}_{t+1}^{\text{stack}} - \left(\mathbf{X}_t^{\text{stack}} + 1\right) \odot \mathbf{dbl\_stacked}$$

$$\mathbf{X}_{t+1}^{\text{pebble}} \leftarrow \mathbf{X}_{t+1}^{\text{route}} - \mathbf{X}_t^{\text{route}}$$

$$\text{is\_stuck} \leftarrow \text{step}\left(\sum_{x,y} \mathbf{X}_{t+1}^{\text{pebble},x,y}\right)$$

$$\mathbf{total\_rank} \leftarrow \mathbf{X}_{t+1}^{\text{stack\_rank}} + \mathbf{X}_{t+1}^{\text{stack\_direction}}$$

$$\mathbf{total\_rank} \leftarrow \text{total\_rank} + \text{sawtooth}_0\left(\mathbf{total\_rank}\right) \cdot L$$

$$\text{min\_total\_rank} \leftarrow \min_{x,y}\left(\mathbf{total\_rank}_{t+1}^{x,y}\right)$$

$$\mathbf{is\_popped} \leftarrow \text{sawtooth}_0\left(\mathbf{total\_rank} - \text{min\_total\_rank}\right) \cdot \text{is\_stuck}$$

$\forall \text{chan} \in \{\text{stack}, \text{stack\_rank}, \text{stack\_direction}\},$

$$\mathbf{X}_{t+1}^{\text{chan}} \leftarrow \mathbf{X}_{t+1}^{\text{chan}} - \mathbf{X}_{t+1}^{\text{chan}} \odot \mathbf{is\_popped}$$

A.3 Experimental details

For each experiment (each row in a table), 5 trials were conducted, involving $50,000$ model update steps. Each trial took up to 5, running on nodes with a single GPU on a High Performance Computing cluster, with GPUs comparable to the GTX 2080 Ti. In tables, we report standard deviations of each experiment in each metric over all trials.

A.4 Extended results

In Table 11, we compare Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs) on the pathfinding task. Surprisingly, GATs are outperformed by GCNs on this task, even though the anisotropic capacity of GATs more closely ressembles that of NCAs. In Table 10, we find that representing the grid-maze as a graph in which only traversable tiles and edges are included in the graph leads to significantly increased performance in the GCN model.

In Table 12 we conduct a hyperparameter sweep over various model architectures.

model	traversable edges only	n. layers	n. hid chan	model	train	test		
				—	16x16	16x16		32x32
				n. params	accuracy	accuracy	pct. complete	accuracy
GCN	False	16	**96**	81,600	27.22 ± 9.68	27.09 ± 9.62	10.67 ± 4.10	0.04 ± 2.80
			128	143,616	15.49 ± 16.35	15.41 ± 16.26	6.21 ± 6.59	0.36 ± 1.28
			256	565,760	20.94 ± 17.78	20.36 ± 17.29	8.85 ± 7.66	-1.14 ± 2.69
		32	**96**	158,400	27.31 ± 14.44	27.13 ± 14.34	11.74 ± 6.44	-1.52 ± 3.52
			128	278,784	28.81 ± 15.18	28.54 ± 15.04	13.14 ± 7.37	-1.43 ± 3.19
			256	1,098,240	28.12 ± 19.30	27.24 ± 18.71	14.92 ± 10.61	-4.64 ± 6.94
		64	**96**	312,000	17.87 ± 18.31	17.87 ± 18.30	8.36 ± 8.93	-2.02 ± 5.27
			128	549,120	7.73 ± 15.80	7.69 ± 15.72	3.92 ± 8.26	-1.65 ± 5.13
			256	2,163,200	12.11 ± 19.33	11.96 ± 19.09	6.66 ± 10.76	-0.51 ± 2.43
	True	16	**96**	81,600	50.05 ± 26.80	50.16 ± 26.87	22.87 ± 13.01	-45.97 ± 215.37
			128	143,616	56.78 ± 30.19	56.97 ± 30.30	31.74 ± 17.11	-42.62 ± 150.45
			256	565,760	55.57 ± 38.38	55.59 ± 38.39	32.49 ± 22.45	-85.07 ± 229.68
		32	**96**	158,400	39.26 ± 34.24	39.27 ± 34.25	19.10 ± 17.12	27.99 ± 25.41
			128	278,784	45.65 ± 40.03	45.68 ± 40.05	25.50 ± 22.98	33.75 ± 29.60
			256	1,098,240	**69.17** ± 36.46	**68.95** ± 36.35	**40.32** ± 21.27	**50.21** ± 28.55
		64	**96**	312,000	39.04 ± 82.35	38.42 ± 84.73	28.22 ± 17.24	16.01 ± 99.26
			128	549,120	42.11 ± 37.59	42.15 ± 37.61	23.21 ± 20.91	29.47 ± 29.44
			256	2,163,200	51.27 ± 44.19	51.26 ± 44.18	29.87 ± 25.75	44.59 ± 38.42

Table 10: Shortest path problem: graph representation. Representing the grid-maze as a graph containing only traversable nodes and edges increases generalization.

model	positional edge features	n. layers	n. hid chan	model	train	test		
				—	16x16	16x16		32x32
				n. params	accuracy	accuracy	pct. complete	accuracy
GAT	False	16	**96**	153,600	40.98 ± 23.59	41.20 ± 23.72	5.84 ± 5.69	31.06 ± 17.73
			128	270,336	55.75 ± 4.41	56.11 ± 4.56	13.11 ± 5.10	41.76 ± 3.02
			256	1,064,960	55.22 ± 30.90	55.38 ± 30.99	16.04 ± 12.82	41.43 ± 23.17
		32	**96**	307,200	24.93 ± 34.14	24.99 ± 34.22	5.97 ± 9.22	19.04 ± 26.08
			128	540,672	56.63 ± 31.71	56.62 ± 31.70	17.92 ± 11.88	44.11 ± 24.82
			256	2,129,920	62.84 ± 35.22	62.78 ± 35.19	22.37 ± 13.54	51.25 ± 28.74
		64	**96**	614,400	53.83 ± 30.27	53.85 ± 30.27	16.03 ± 11.84	42.17 ± 23.61
			128	1,081,344	61.71 ± 34.60	61.78 ± 34.64	21.99 ± 12.56	50.76 ± 28.65
			256	4,259,840	44.59 ± 41.08	44.66 ± 41.15	14.23 ± 15.33	37.58 ± 34.97
	True	16	**96**	156,672	27.56 ± 25.32	27.69 ± 25.45	5.24 ± 6.67	21.47 ± 19.66
			128	274,432	44.97 ± 25.86	45.14 ± 25.96	12.28 ± 9.92	33.99 ± 19.51
			256	1,073,152	26.54 ± 36.38	26.57 ± 36.41	4.17 ± 5.72	20.10 ± 27.57
		32	**96**	313,344	39.26 ± 36.02	39.31 ± 36.06	10.10 ± 11.75	29.72 ± 27.29
			128	548,864	54.86 ± 30.75	54.97 ± 30.82	19.12 ± 11.04	41.53 ± 23.28
			256	2,146,304	15.51 ± 34.68	15.45 ± 34.55	5.58 ± 12.47	12.64 ± 28.27
		64	**96**	626,688	**73.62** ± 2.12	**73.77** ± 2.13	26.01 ± 5.62	58.15 ± 2.16
			128	1,097,728	71.93 ± 6.94	72.10 ± 6.97	26.18 ± 3.82	49.45 ± 23.78
			256	4,292,608	15.48 ± 34.61	15.46 ± 34.58	2.50 ± 5.58	12.96 ± 28.98
GCN	False	16	**96**	153,600	53.07 ± 30.04	53.26 ± 30.14	26.36 ± 15.27	39.91 ± 22.68
			128	270,336	68.96 ± 4.56	69.19 ± 4.65	37.93 ± 4.27	51.16 ± 3.53
			256	1,064,960	31.27 ± 42.82	31.29 ± 42.85	18.66 ± 25.55	24.07 ± 32.97
		32	**96**	307,200	40.64 ± 37.91	40.58 ± 37.86	19.96 ± 19.50	30.95 ± 29.40
			128	540,672	44.89 ± 41.90	44.95 ± 41.95	24.19 ± 23.36	34.92 ± 32.78
			256	2,129,920	68.75 ± 38.43	68.51 ± 38.30	39.55 ± 22.11	56.53 ± 31.62
		64	**96**	614,400	65.21 ± 36.49	65.33 ± 36.55	35.88 ± 20.19	52.38 ± 29.40
			128	1,081,344	49.12 ± 44.84	49.13 ± 44.86	27.60 ± 25.22	39.17 ± 35.80
			256	4,259,840	69.48 ± 38.85	69.35 ± 38.77	**40.27** ± 22.52	**58.19** ± 32.54

Table 11: Shortest path problem – GNN model architectures (without weight-sharing): Despite being more closely aligned to the NCA, the GAT does not outperform GCNs, potentially due to insufficient training time.

In Tables 13 and 14, we conduct hyperparameter sweeps on NCA models on the pathfinding and diameter problems, respectively (all prior tables focusing on NCAs comprise select rows from this larger sweep).

			model	train	test		
			—	16x16	16x16		32x32
			n. params	accuracies	accuracies	pct. complete	accuracies
model	**n. layers**	**n. hid chan**					
GCN	**16**	**96**	9,600	47.02 ± 26.29	47.09 ± 26.32	19.38 ± 10.84	-131.51 ± 291.67
		128	16,896	44.59 ± 40.73	44.77 ± 40.89	25.56 ± 23.36	-136.19 ± 169.80
		256	66,560	79.85 ± 1.99	79.90 ± 1.97	46.31 ± 1.69	-194.49 ± 296.77
	32	**96**	9,600	37.89 ± 34.59	37.95 ± 34.65	18.24 ± 16.65	25.01 ± 23.75
		128	16,896	46.41 ± 42.99	46.41 ± 42.99	26.80 ± 25.24	32.57 ± 29.88
		256	66,560	69.61 ± 38.92	69.36 ± 38.78	41.09 ± 22.97	43.92 ± 27.12
	64	**96**	9,600	12.87 ± 110.48	11.50 ± 114.14	20.54 ± 10.68	-20.56 ± 134.55
		128	16,896	35.10 ± 32.34	35.19 ± 32.41	18.83 ± 17.30	19.77 ± 20.87
		256	66,560	33.88 ± 46.40	33.98 ± 46.54	18.94 ± 26.10	29.56 ± 40.55
MLP	**16**	**96**	16,257,024	77.94 ± 1.32	14.75 ± 0.68	2.73 ± 0.18	0.00 ± 0.00
		128	21,565,440	75.90 ± 3.30	12.42 ± 1.74	2.77 ± 0.10	0.00 ± 0.00
		256	42,799,104	52.11 ± 29.83	8.13 ± 4.57	1.88 ± 1.11	0.00 ± 0.00
	32	**96**	16,257,024	31.90 ± 43.68	5.85 ± 8.04	1.24 ± 1.71	0.00 ± 0.00
		128	21,565,440	29.47 ± 40.36	4.83 ± 6.63	1.02 ± 1.39	0.00 ± 0.00
		256	42,799,104	2.82 ± 42.89	-8.34 ± 25.59	0.54 ± 1.20	0.00 ± 0.00
	64	**96**	16,257,024	70.69 ± 3.29	13.91 ± 1.04	2.74 ± 0.20	0.00 ± 0.00
		128	21,565,440	63.58 ± 3.09	12.48 ± 1.74	2.45 ± 0.20	0.00 ± 0.00
		256	42,799,104	52.92 ± 6.63	12.82 ± 1.78	1.94 ± 0.17	0.00 ± 0.00
NCA	**16**	**96**	86,400	99.78 ± 0.05	96.04 ± 0.33	93.86 ± 0.59	62.71 ± 35.01
		128	152,064	**99.97** ± 0.01	96.67 ± 0.20	95.16 ± 0.26	87.08 ± 1.26
		256	599,040	79.97 ± 44.71	78.03 ± 43.62	77.58 ± 43.37	70.12 ± 39.22
	32	**96**	86,400	99.67 ± 0.28	96.79 ± 0.70	96.26 ± 0.47	84.75 ± 6.69
		128	152,064	99.92 ± 0.09	**97.78** ± 0.32	97.61 ± 0.30	**91.78** ± 1.51
		256	599,040	79.68 ± 44.54	78.33 ± 43.79	78.27 ± 43.76	74.24 ± 41.55
	64	**128**	152,064	99.39 ± 0.15	97.44 ± 0.13	**97.61** ± 0.20	90.06 ± 2.93
		256	599,040	79.14 ± 44.24	77.90 ± 43.55	78.35 ± 43.80	73.92 ± 41.36

Table 12: Shortest path problem: model architecture. Neural Cellular Automata generalize best, while Graph Convolutional Networks generalize better than Multilayer Perceptrons.

model	evo. data	kernel	max-pool	shared weights	cut corners	n. hid chan	model	train		test	
							—	16x16		16x16	32x32
							n. params	sol len	accuracies	accuracies	accuracies
NCA	False	3	False	False	False	128	9,584,640	24.09 ± 0.00	99.82 ± 0.06	78.52 ± 0.71	16.85 ± 14.55
					True	128	5,324,800	24.09 ± 0.00	99.85 ± 0.07	73.38 ± 2.33	12.77 ± 5.20
				True	False	128	149,760	24.09 ± 0.00	95.93 ± 0.42	83.65 ± 0.53	33.46 ± 9.10
					True	128	83,200	24.09 ± 0.00	78.05 ± 43.63	66.89 ± 37.40	-9.86 ± 12.18
			True	False	False	128	9,584,640	24.09 ± 0.00	99.79 ± 0.05	79.21 ± 1.90	30.88 ± 12.92
					True	128	5,324,800	24.09 ± 0.00	99.67 ± 0.15	82.57 ± 0.93	46.06 ± 4.74
				True	False	128	149,760	24.09 ± 0.00	77.41 ± 43.27	67.20 ± 37.57	27.47 ± 52.59
					True	128	83,200	24.09 ± 0.00	95.72 ± 0.45	86.54 ± 1.07	56.31 ± 26.80
		5	False	False	False	128	26,624,000	24.09 ± 0.00	**99.93** ± 0.04	59.30 ± 4.76	-8.37 ± 9.73
					True	128	13,844,480	24.09 ± 0.00	99.84 ± 0.03	68.04 ± 6.29	-1.71 ± 25.78
				True	False	128	416,000	24.09 ± 0.00	95.09 ± 0.94	77.63 ± 1.01	17.58 ± 2.79
					True	128	216,320	24.09 ± 0.00	95.06 ± 1.69	80.73 ± 1.64	26.02 ± 7.79
			True	False	False	128	26,624,000	24.09 ± 0.00	99.90 ± 0.06	67.60 ± 1.17	2.78 ± 8.84
					True	128	13,844,480	24.09 ± 0.00	99.79 ± 0.14	75.09 ± 2.22	23.18 ± 12.67
				True	False	128	416,000	24.09 ± 0.00	96.62 ± 0.96	76.89 ± 0.93	21.85 ± 14.69
					True	128	216,320	24.09 ± 0.00	96.30 ± 0.54	81.43 ± 0.45	49.74 ± 2.58
		3	False	False	False	128	9,584,640	29.45 ± 0.72	86.88 ± 0.87	88.37 ± 1.58	39.32 ± 17.32
					True	128	5,324,800	29.56 ± 0.21	76.45 ± 0.97	89.88 ± 0.98	46.38 ± 2.70
				True	False	128	149,760	26.34 ± 1.27	69.32 ± 38.76	71.53 ± 39.99	-6.90 ± 42.96
					True	128	83,200	24.81 ± 0.54	69.91 ± 39.08	72.31 ± 40.43	-62.54 ± 95.29
			True	False	False	128	9,584,640	30.58 ± 0.58	82.15 ± 2.07	89.02 ± 1.56	55.48 ± 15.45
					True	128	5,324,800	30.24 ± 0.37	83.18 ± 2.97	89.41 ± 0.86	**70.35** ± 2.62
				True	False	128	149,760	27.43 ± 0.15	87.76 ± 0.79	90.02 ± 0.44	30.02 ± 64.80
					True	128	83,200	26.19 ± 0.36	86.96 ± 1.41	89.54 ± 0.91	31.89 ± 44.56
	True	5	False	False	False	128	26,624,000	30.52 ± 0.41	82.21 ± 0.81	**90.50** ± 0.21	43.19 ± 4.75
					True	128	13,844,480	29.62 ± 0.52	80.37 ± 1.31	88.96 ± 1.64	47.23 ± 3.04
				True	False	128	416,000	28.03 ± 0.13	86.06 ± 1.22	88.95 ± 1.36	38.04 ± 5.62
					True	128	216,320	27.61 ± 0.27	85.77 ± 1.01	89.96 ± 1.71	29.37 ± 26.44
			True	False	False	128	26,624,000	**30.68** ± 0.20	87.87 ± 1.00	88.82 ± 0.36	40.32 ± 12.18
					True	128	13,844,480	30.24 ± 0.18	84.83 ± 2.20	88.24 ± 1.02	53.65 ± 7.34
				True	False	128	416,000	28.03 ± 0.35	85.54 ± 1.06	89.88 ± 0.55	48.12 ± 16.01
					True	128	216,320	27.44 ± 0.29	86.44 ± 1.09	89.58 ± 0.52	24.46 ± 29.35

Table 13: Diameter problem: hyperparameter sweep.

model	evo. data	kernel	max-pool	shared weights	cut corners	n. hid chan	model	train		test		
							—	16x16		16x16		32x32
							n. params	sol len	accuracies	accuracies	pct. complete	accuracies
NCA	False	3	False	False	False	128	9,732,096	9.02 ± 0.00	99.92 ± 0.08	97.12 ± 0.42	97.07 ± 0.59	86.72 ± 3.29
					True	128	5,406,720	9.02 ± 0.00	99.89 ± 0.11	97.01 ± 0.38	96.75 ± 0.76	86.85 ± 1.66
				True	False	128	152,064	9.02 ± 0.00	98.91 ± 0.69	97.38 ± 0.65	97.66 ± 0.93	90.89 ± 2.09
					True	128	84,480	9.02 ± 0.00	79.12 ± 44.24	78.09 ± 43.66	78.30 ± 43.77	73.87 ± 41.30
			True	False	False	128	9,732,096	9.02 ± 0.00	99.94 ± 0.02	96.47 ± 0.13	96.07 ± 0.06	86.12 ± 2.09
					True	128	5,406,720	9.02 ± 0.00	99.91 ± 0.04	97.11 ± 0.42	96.76 ± 0.57	86.30 ± 1.38
				True	False	128	152,064	9.02 ± 0.00	98.96 ± 0.46	97.09 ± 0.56	97.76 ± 0.41	87.00 ± 3.14
					True	128	84,480	9.02 ± 0.00	99.58 ± 0.12	98.08 ± 0.28	98.32 ± 0.31	88.61 ± 4.07
		5	False	False	False	128	27,033,600	9.02 ± 0.00	79.95 ± 44.69	74.95 ± 41.90	74.97 ± 41.92	51.93 ± 35.75
					True	128	14,057,472	9.02 ± 0.00	**99.95** ± 0.03	94.89 ± 0.79	95.05 ± 0.61	78.46 ± 3.80
				True	False	128	422,400	9.02 ± 0.00	39.70 ± 54.36	38.89 ± 53.26	39.01 ± 53.41	36.43 ± 49.89
					True	128	219,648	9.02 ± 0.00	59.48 ± 54.30	58.29 ± 53.21	58.48 ± 53.39	54.40 ± 49.66
			True	False	False	128	27,033,600	9.02 ± 0.00	99.95 ± 0.04	92.95 ± 0.55	92.10 ± 1.36	64.62 ± 16.19
					True	128	14,057,472	9.02 ± 0.00	99.87 ± 0.07	93.27 ± 1.19	93.09 ± 0.99	75.81 ± 3.94
				True	False	128	422,400	9.02 ± 0.00	39.40 ± 53.95	38.24 ± 52.36	38.53 ± 52.77	32.80 ± 44.95
					True	128	219,648	9.02 ± 0.00	99.40 ± 0.14	97.36 ± 0.37	97.67 ± 0.18	87.28 ± 4.18
		3	False	False	False	128	9,732,096	28.08 ± 0.21	98.73 ± 0.11	99.25 ± 0.27	98.61 ± 0.96	97.91 ± 0.50
					True	128	5,406,720	24.04 ± 8.40	78.71 ± 44.00	79.61 ± 44.51	79.59 ± 44.49	78.55 ± 43.91
				True	False	128	152,064	23.75 ± 1.36	96.81 ± 1.54	99.29 ± 0.53	99.53 ± 0.32	98.47 ± 1.01
					True	128	84,480	24.50 ± 1.17	97.54 ± 0.88	**99.64** ± 0.12	**99.75** ± 0.07	**98.95** ± 0.40
			True	False	False	128	9,732,096	28.06 ± 0.23	99.49 ± 0.31	99.30 ± 0.35	98.77 ± 0.97	95.22 ± 3.48
					True	128	5,406,720	26.43 ± 1.21	98.77 ± 0.39	99.48 ± 0.09	99.25 ± 0.34	94.82 ± 1.32
				True	False	128	152,064	19.88 ± 6.13	76.83 ± 42.96	79.39 ± 44.38	79.60 ± 44.50	78.43 ± 43.85
					True	128	84,480	22.39 ± 7.48	78.07 ± 43.64	79.64 ± 44.52	79.62 ± 44.51	78.95 ± 44.13
		5	False	False	False	128	27,033,600	24.55 ± 8.68	79.59 ± 44.49	79.44 ± 44.41	79.12 ± 44.24	76.94 ± 43.02
					True	128	14,057,472	**28.62** ± 0.19	98.79 ± 1.33	98.72 ± 0.82	98.04 ± 1.73	96.19 ± 1.12
				True	False	128	422,400	25.52 ± 0.30	95.90 ± 1.40	99.04 ± 0.43	99.23 ± 0.54	97.80 ± 0.88
					True	128	219,648	22.69 ± 3.80	95.29 ± 2.23	98.40 ± 1.01	98.66 ± 1.20	96.13 ± 3.56
			True	False	False	128	27,033,600	24.61 ± 0.55	99.57 ± 0.09	98.67 ± 0.36	97.78 ± 0.85	88.58 ± 9.19
					True	128	14,057,472	24.21 ± 0.65	99.42 ± 0.22	99.13 ± 0.05	98.53 ± 0.23	94.94 ± 1.11
				True	False	128	422,400	22.07 ± 1.38	92.99 ± 4.99	95.28 ± 6.31	93.76 ± 9.44	66.13 ± 64.35
					True	128	219,648	18.43 ± 5.20	91.74 ± 4.23	97.78 ± 1.14	98.40 ± 0.71	94.08 ± 3.15

Table 14: Pathfinding problem: hyperparameter sweep.