Cellular Automata for RL and Beyond

June 12th, 2020

Justin Stevens

Outline

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

- $\mathbb{Z}^d$ represents the Euclidean space we are in. For pathfinding on maps, this is typically $d = 2$.

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

- $\mathbb{Z}^d$ represents the Euclidean space we are in. For pathfinding on maps, this is typically $d = 2$.
- Q represents a set of finite discrete integer states, which the values of the cellular automata can take on.

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

- $\mathbb{Z}^d$ represents the Euclidean space we are in. For pathfinding on maps, this is typically $d = 2$.
- Q represents a set of finite discrete integer states, which the values of the cellular automata can take on.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

- $\mathbb{Z}^d$ represents the Euclidean space we are in. For pathfinding on maps, this is typically $d = 2$.
- Q represents a set of finite discrete integer states, which the values of the cellular automata can take on.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to Q$ is the transition function, which takes in an N-tuple and outputs a new state to transition to.

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

- $\mathbb{Z}^d$ represents the Euclidean space we are in. For pathfinding on maps, this is typically $d = 2$.
- Q represents a set of finite discrete integer states, which the values of the cellular automata can take on.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to Q$ is the transition function, which takes in an N-tuple and outputs a new state to transition to.

A **configuration** of the cellular automata is a function $c : \mathbb{Z}^d \to Q$, which assigns a specific value to every state.

Definition [4]

A **cellular automata** is defined as a 4-tuple $(\mathbb{Z}^d, Q, N, \delta)$, where:

- $\mathbb{Z}^d$ represents the Euclidean space we are in. For pathfinding on maps, this is typically $d = 2$.
- Q represents a set of finite discrete integer states, which the values of the cellular automata can take on.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to Q$ is the transition function, which takes in an N-tuple and outputs a new state to transition to.

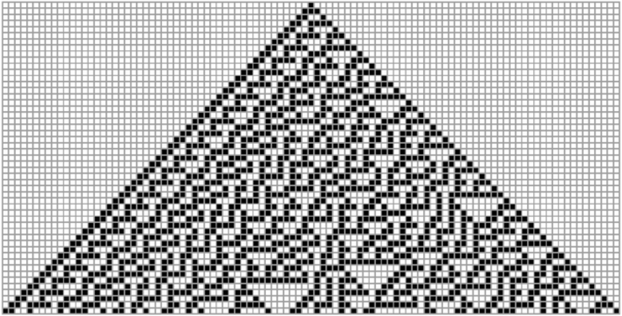
A **configuration** of the cellular automata is a function $c : \mathbb{Z}^d \to Q$, which assigns a specific value to every state. A **computation** on the cellular automata is given by the global function G specifying

$$\forall c \in Q^{\mathbb{Z}^d} \; \forall i \in \mathbb{Z}^d, G(c, i) = \delta(c(i + n_1), c(i + n_2), \cdots, c(i + n_{|N|})).$$

Example of Cellular Automata

In Wolfram's rules $d = 1$, $Q = \{0, 1\}$, $N = (-1, 0, 1)$ for the values to the left, center, and right, and $\delta : Q^3 \to Q$ is determined by the rule number. For rule 30 since $30 = 11110_2$ this is given by

Showing how the grid is evolved over time with rule 30 is shown below.



Learning Automata [1]

Fundamentally a learning automata is a policy iterator.

Learning Automata [1]

Fundamentally a learning automata is a policy iterator. There are r actions on each step. An initial probability vector $p(0) = [p_1(0), p_2(0), \cdots, p_r(0)]^T$ is initialized, where $p_i(t)$ represents the probability action i is selected on time step t.

Learning Automata [1]

Fundamentally a learning automata is a policy iterator. There are r actions on each step. An initial probability vector $p(0) = [p_1(0), p_2(0), \cdots, p_r(0)]^T$ is initialized, where $p_i(t)$ represents the probability action i is selected on time step t. There are two outcomes of the action: either a favourable or unfavourable response is generated.

Learning Automata [1]

Fundamentally a learning automata is a policy iterator. There are r actions on each step. An initial probability vector $p(0) = [p_1(0), p_2(0), \cdots, p_r(0)]^T$ is initialized, where $p_i(t)$ represents the probability action i is selected on time step t. There are two outcomes of the action: either a favourable or unfavourable response is generated. In the case of a favourable response, the probability vector is updated according to

$$p_i(t+1) = \begin{cases} p_i(t) + \alpha(1 - p_i(t)) & \text{if action i was selected} \\ (1-\alpha)p_i(t) & \text{otherwise} \end{cases}.$$

Learning Automata [1]

Fundamentally a learning automata is a policy iterator. There are r actions on each step. An initial probability vector $p(0) = [p_1(0), p_2(0), \cdots, p_r(0)]^T$ is initialized, where $p_i(t)$ represents the probability action i is selected on time step t. There are two outcomes of the action: either a favourable or unfavourable response is generated. In the case of a favourable response, the probability vector is updated according to

$$p_i(t+1) = \begin{cases} p_i(t) + \alpha(1 - p_i(t)) & \text{if action i was selected} \\ (1-\alpha)p_i(t) & \text{otherwise} \end{cases}.$$

In the case of an unfavourable response, the probability vector is updated:

$$p_i(t+1) = \begin{cases} (1-\beta)p_i(t) & \text{if action i was selected} \\ \frac{\beta}{r-1} + (1-\beta)p_i(t) & \text{otherwise} \end{cases}.$$

The parameters of the algorithm are α and β. Notice the above equations assure the probabilities sum to 1.

Learning Automata Properties

- When $\alpha = \beta$ the algorithm is called *linear reward-penalty*, L_{RP}. When $\beta = 0$ so only reward is considered, it's called *linear reward-inaction*, L_{RI}. When $a \ll b$, it's called *linear reward-* ϵ *penalty*.

Learning Automata Properties

- When $\alpha = \beta$ the algorithm is called *linear reward-penalty*, L_{RP}. When $\beta = 0$ so only reward is considered, it's called *linear reward-inaction*, L_{RI}. When $a \ll b$, it's called *linear reward-* ϵ *penalty*.
- The algorithms are useful in multi-agent settings when traditional reinforcement learning techniques don't guarantee convergence.

Learning Automata Properties

- When $\alpha = \beta$ the algorithm is called *linear reward-penalty*, L_{RP}. When $\beta = 0$ so only reward is considered, it's called *linear reward-inaction*, L_{RI}. When $a \ll b$, it's called *linear reward-* ϵ *penalty*.
- The algorithms are useful in multi-agent settings when traditional reinforcement learning techniques don't guarantee convergence.
- Combines fast convergence with low computation complexity.

Learning Automata Properties

- When $\alpha = \beta$ the algorithm is called *linear reward-penalty*, L_{RP}. When $\beta = 0$ so only reward is considered, it's called *linear reward-inaction*, L_{RI}. When $a \ll b$, it's called *linear reward-* ϵ *penalty*.
- The algorithms are useful in multi-agent settings when traditional reinforcement learning techniques don't guarantee convergence.
- Combines fast convergence with low computation complexity.
- Originated in mathematical psychology, but now is used in engineering.
- Also used in games with stochastic payoffs, solving NP-complete problems like graph partitioning more efficiently, data network routing, and neural network engineering.

Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

- $\mathbb{Z}^d$ represents the Euclidean space we are in.
- Q represents a set of states, which are the actions selected by the LA.

Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

- $\mathbb{Z}^d$ represents the Euclidean space we are in.
- Q represents a set of states, which are the actions selected by the LA.
- A represents the set of learning automata, each of which is assigned to one cell of the CLA.

Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

- $\mathbb{Z}^d$ represents the Euclidean space we are in.
- Q represents a set of states, which are the actions selected by the LA.
- A represents the set of learning automata, each of which is assigned to one cell of the CLA.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to \beta$ is the transition function, which takes in an N-tuple of actions around the cell and outputs the reinforcement signal.

Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

- $\mathbb{Z}^d$ represents the Euclidean space we are in.
- Q represents a set of states, which are the actions selected by the LA.
- A represents the set of learning automata, each of which is assigned to one cell of the CLA.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to \beta$ is the transition function, which takes in an N-tuple of actions around the cell and outputs the reinforcement signal.

The CLA initializes each of the LA in each cell either at random or based on some past knowledge. Then each LA selects an action based on this probability distribution.

Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

- $\mathbb{Z}^d$ represents the Euclidean space we are in.
- Q represents a set of states, which are the actions selected by the LA.
- A represents the set of learning automata, each of which is assigned to one cell of the CLA.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to \beta$ is the transition function, which takes in an N-tuple of actions around the cell and outputs the reinforcement signal.

The CLA initializes each of the LA in each cell either at random or based on some past knowledge. Then each LA selects an action based on this probability distribution. The CA then looks at the actions of the $|N|$ neighbouring cells and passes this to the δ function, which returns a reinforcement signal.

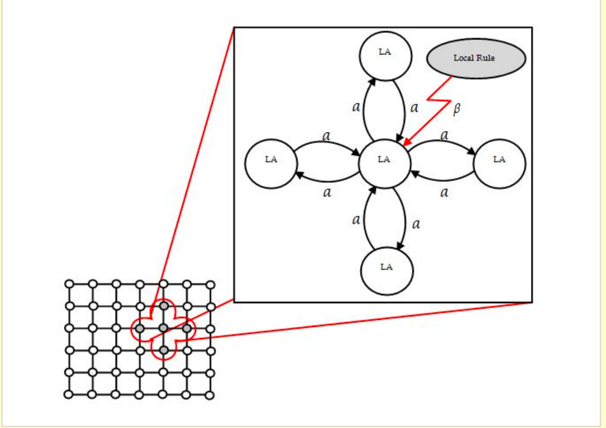
Cellular Learning Automata Definition [2]

A **cellular learning automata** is defined as a 5-tuple $(\mathbb{Z}^d, Q, A, N, \delta)$:

- $\mathbb{Z}^d$ represents the Euclidean space we are in.
- Q represents a set of states, which are the actions selected by the LA.
- A represents the set of learning automata, each of which is assigned to one cell of the CLA.
- $N = (n_1, n_2, \cdots, n_{|N|})$ represents the neighbourhood function, which defines for a specific cell, which cells around it to consider in its computation, where $n_i \in \mathbb{Z}^d$.
- $\delta : Q^{|N|} \to \beta$ is the transition function, which takes in an N-tuple of actions around the cell and outputs the reinforcement signal.

The CLA initializes each of the LA in each cell either at random or based on some past knowledge. Then each LA selects an action based on this probability distribution. The CA then looks at the actions of the $|N|$ neighbouring cells and passes this to the δ function, which returns a reinforcement signal. Based on this signal, each LA updates its probability distributions with equations similar to those before.

Cellular Learning Automata Intuition [2]



Assuming for the current cell, each of the neighbouring cells have m_i options for which action to select. Then the update rules is expressed as a $m_1 \times m_2 \cdots m_{|N|}$ hypermatrix.

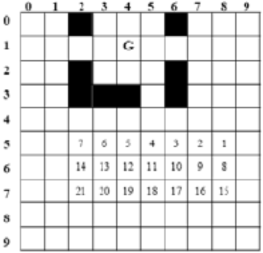
Applications of CLAs

- In [2] applications are mentioned including image processing, rumour diffusion, and modelling commerce networks.
- They also mention that if an asynchronous CLA is applied, where each cell may contain multiple LAs, but only one cell is active at a current time. This was used successfully in modelling adaptive controllers.
- In [3], the authors use CLAs to solve the channel assignment problem.
- The behaviour of CLAs can be modelled with differential equation.

Outline

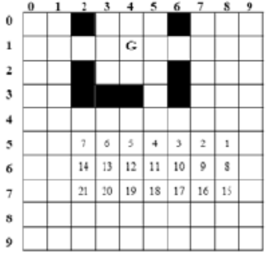
Stationary Approach using Heuristics [4]

Assume we have 21 agents all trying to reach the goal with no conflicts.



Stationary Approach using Heuristics [4]

Assume we have 21 agents all trying to reach the goal with no conflicts.



We can repeatedly apply the Bellman optimality updates to each cell.

$$
q_{i,j}^{t+1} = \begin{cases} 1 & ; \ q_{i,j}^{t} = 1 \quad \begin{pmatrix} \text{1 stands for obstacle} \\ \text{in cellular automaton} \end{pmatrix} \\[2ex] \min \left\{ \begin{array}{l} q_{i,j}^{t} \,,\; q_{i-1,j-1}^{t} + 14 \,,\; q_{i-1,j} + 10, \\ q_{i-1,j+1}^{t} + 14 \,,\; q_{i,j-1}^{t} + 10, \\ q_{i,j+1} + 10 \,,\; q_{i+1,j-1}^{t} + 14, \\ q_{i+1,j}^{t} + 10 \,,\; q_{i+1,j+1} + 14 \end{array} \right\} & \underset{\substack{r=-1,0,1 \\ s=-1,0,1}}{\forall\, q_{i+r,j+s}^{t} \neq 1} \ ;O.W. \end{cases}
$$

Centralized Planner

The proposed algorithm in [4] uses a centralized planner. Their basic idea is to store the current q values for each cell in a table, call it h. Then an optimal direction, dir for each cell can be computed. If multiple cells want to move to the same cell, we record this in the variable *collisions*.

Centralized Planner

The proposed algorithm in [4] uses a centralized planner. Their basic idea is to store the current q values for each cell in a table, call it h. Then an optimal direction, dir for each cell can be computed. If multiple cells want to move to the same cell, we record this in the variable *collisions*.
The formula for each cell c is then given by

$$f(i, j, \text{dir}) = g(\text{dir}) + h(c + \text{dir}) + \gamma \cdot \text{collisions}(c + \text{dir}).$$

Centralized Planner

The proposed algorithm in [4] uses a centralized planner. Their basic idea is to store the current q values for each cell in a table, call it h. Then an optimal direction, dir for each cell can be computed. If multiple cells want to move to the same cell, we record this in the variable *collisions*.
The formula for each cell c is then given by

$$f(i, j, \text{dir}) = g(\text{dir}) + h(c + \text{dir}) + \gamma \cdot \text{collisions}(c + \text{dir}).$$

In the above formula, g stands for the cost of taking a move in a direction, h calculates the heuristic to the goal from the new location, $c + \text{dir}$, and collisions computes the number of collisions at this new location.

Centralized Planner

The proposed algorithm in [4] uses a centralized planner. Their basic idea is to store the current q values for each cell in a table, call it h. Then an optimal direction, dir for each cell can be computed. If multiple cells want to move to the same cell, we record this in the variable *collisions*.
The formula for each cell c is then given by

$$f(i, j, \mathrm{dir}) = g(\mathrm{dir}) + h(c + \mathrm{dir}) + \gamma \cdot \mathrm{collisions}(c + \mathrm{dir}).$$

In the above formula, g stands for the cost of taking a move in a direction, h calculates the heuristic to the goal from the new location, $c + \mathrm{dir}$, and collisions computes the number of collisions at this new location.

The agents then move in a direction so as to minimize f. If $\gamma = 0$, the performance will be equivalent to $A^\star$.

Centralized Planner

The proposed algorithm in [4] uses a centralized planner. Their basic idea is to store the current q values for each cell in a table, call it h. Then an optimal direction, dir for each cell can be computed. If multiple cells want to move to the same cell, we record this in the variable *collisions*.
The formula for each cell c is then given by

$$f(i, j, \mathrm{dir}) = g(\mathrm{dir}) + h(c + \mathrm{dir}) + \gamma \cdot \mathrm{collisions}(c + \mathrm{dir}).$$

In the above formula, g stands for the cost of taking a move in a direction, h calculates the heuristic to the goal from the new location, $c + \mathrm{dir}$, and collisions computes the number of collisions at this new location.

The agents then move in a direction so as to minimize f. If $\gamma = 0$, the performance will be equivalent to $A^\star$. The time complexity of the algorithm on an $m \times m$ grid with n agents is $O(n^3 m^4)$.

Modified Value Iteration Algorithm

In [5], they used the value iteration on each step to compute a direction. They began by initializing $V_0(s) = 0 \forall s$.

Modified Value Iteration Algorithm

In [5], they used the value iteration on each step to compute a direction. They began by initializing $V_0(s) = 0 \forall s$. They then computed on every step

$$Q_{t+1}(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma V_t(s')]$$

$$V_{t+1}(s) = \max_a Q_{t+1}(s, a),$$

$$\pi_{t+1}(s) = \text{argmax}_a Q_{t+1}(s, a).$$

Modified Value Iteration Algorithm

In [5], they used the value iteration on each step to compute a direction. They began by initializing $V_0(s) = 0 \forall s$. They then computed on every step

$$Q_{t+1}(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V_t(s')]$$

$$V_{t+1}(s) = \max_a Q_{t+1}(s,a),$$

$$\pi_{t+1}(s) = \operatorname{argmax}_a Q_{t+1}(s,a).$$

The directions were used in a local collision avoidance model. They discretized the RL space into smaller grids to allow more agents to pass. They then created the local navigation map which can be seen on the right.

(a) (b) (c)

Benefits of Using Cellular Automata

- Most approaches grow exponentially with the number of agents, but since this is based on local updates, it does not.

Benefits of Using Cellular Automata

- Most approaches grow exponentially with the number of agents, but since this is based on local updates, it does not.
- Furthermore, it is able to respond dynamically to new obstacles being placed, which is useful in RTHS.

Benefits of Using Cellular Automata

- Most approaches grow exponentially with the number of agents, but since this is based on local updates, it does not.
- Furthermore, it is able to respond dynamically to new obstacles being placed, which is useful in RTHS.

The CA computes using a *scatter* and *gather* model in order to avoid collisions. For each cell, there is a predefined starting location, S and speed parameter $0 \leq \delta \leq 1$.

Benefits of Using Cellular Automata

- Most approaches grow exponentially with the number of agents, but since this is based on local updates, it does not.
- Furthermore, it is able to respond dynamically to new obstacles being placed, which is useful in RTHS.

The CA computes using a *scatter* and *gather* model in order to avoid collisions. For each cell, there is a predefined starting location, S and speed parameter $0 \leq \delta \leq 1$. In the scatter phase, if an agent is in a cell, then it determines a local goal to move towards. In the gather phase, for cells that an agent wishes to move towards, if there are no collisions, then it moves there. Otherwise, it allows the agent with the largest δ value to move.

Benefits of Using Cellular Automata

- Most approaches grow exponentially with the number of agents, but since this is based on local updates, it does not.
- Furthermore, it is able to respond dynamically to new obstacles being placed, which is useful in RTHS.

The CA computes using a *scatter* and *gather* model in order to avoid collisions. For each cell, there is a predefined starting location, S and speed parameter $0 \leq \delta \leq 1$. In the scatter phase, if an agent is in a cell, then it determines a local goal to move towards. In the gather phase, for cells that an agent wishes to move towards, if there are no collisions, then it moves there. Otherwise, it allows the agent with the largest δ value to move.

The reinforcement learning and CA interact together in order to determine a solution, however, this is not a complete CA solution to the problem.

Crowd Simulation Demo [5]

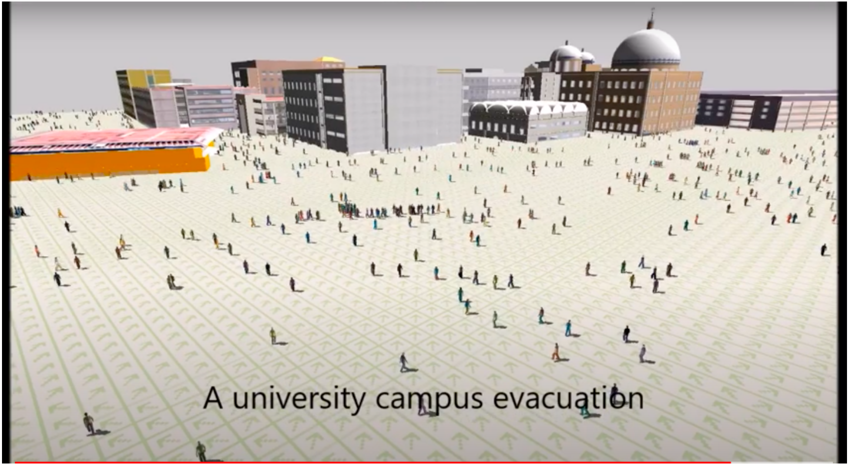


Figure 1: https://www.youtube.com/watch?v=dkx87F10x6k

Outline

Growing Neural Cellular Automata [6]

- Inspired by how human cells are able to self-organize and create complex behaviour, a team of researchers at Google attempted the same phenomen with regrowing images.

Growing Neural Cellular Automata [6]

- Inspired by how human cells are able to self-organize and create complex behaviour, a team of researchers at Google attempted the same phenomen with regrowing images.
- They developed a simple model that showed some surprising results!

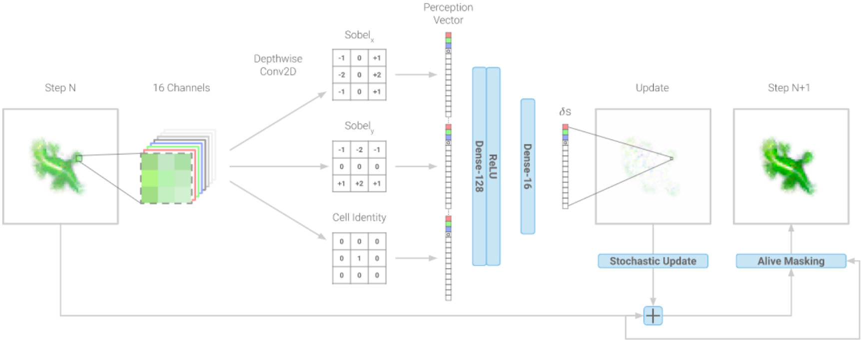
Growing Neural Cellular Automata [6]

- Inspired by how human cells are able to self-organize and create complex behaviour, a team of researchers at Google attempted the same phenomen with regrowing images.
- They developed a simple model that showed some surprising results!
- Each cell is represented by a vector in $\mathbb{R}^{16}$: the first 3 values are RGB, the fourth value is a state value $0 \leq \alpha \leq 1$ that determines if the cell is alive or dead, and the other 12 are undetermined, similar to how our cells use chemical and electrical signals to grow.
- Each cell can detect the Moore neighbourhood (8-connected) around it, in addition to its own value.
- In their model they used continuous cellular automata instead of discrete so the function is differentiable. This was useful since they used a neural network to determine the updates to each cell.

Growing Neural Cellular Automata [6]

- Inspired by how human cells are able to self-organize and create complex behaviour, a team of researchers at Google attempted the same phenomen with regrowing images.
- They developed a simple model that showed some surprising results!
- Each cell is represented by a vector in $\mathbb{R}^{16}$: the first 3 values are RGB, the fourth value is a state value $0 \leq \alpha \leq 1$ that determines if the cell is alive or dead, and the other 12 are undetermined, similar to how our cells use chemical and electrical signals to grow.
- Each cell can detect the Moore neighbourhood (8-connected) around it, in addition to its own value.
- In their model they used continuous cellular automata instead of discrete so the function is differentiable. This was useful since they used a neural network to determine the updates to each cell.

Neural Architecture [6]



A single update step of the model.

Explanation of Model [6]

The Moore neighbourhood plus the current cell form a 3×3 table, to which three separate convolution filters are applied to each of the 16 channels.

Explanation of Model [6]

The Moore neighbourhood plus the current cell form a 3×3 table, to which three separate convolution filters are applied to each of the 16 channels. The convolutional filters are the Sobel filters, which measure the partial derivatives in the x and y direction, plus an indicator of the current cell:

Explanation of Model [6]

The Moore neighbourhood plus the current cell form a 3×3 table, to which three separate convolution filters are applied to each of the 16 channels. The convolutional filters are the Sobel filters, which measure the partial derivatives in the x and y direction, plus an indicator of the current cell:

$$\mathrm{Sobel}_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \mathrm{Sobel}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \mathrm{Id} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

These are then used to create a Perception vector with $16 \times 3 = 48$ components. This vector is passed through a neural network with a Dense-128 layer followed by an RELU, and then another Dense-16 layer.

Explanation of Model [6]

The Moore neighbourhood plus the current cell form a 3×3 table, to which three separate convolution filters are applied to each of the 16 channels. The convolutional filters are the Sobel filters, which measure the partial derivatives in the x and y direction, plus an indicator of the current cell:

$$\mathsf{Sobel}_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \mathsf{Sobel}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \mathsf{Id} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

These are then used to create a Perception vector with $16 \times 3 = 48$ components. This vector is passed through a neural network with a Dense-128 layer followed by an RELU, and then another Dense-16 layer.

This is applied as a residual neural network, so the final δ values are added to the current cell vector in $\mathbb{R}^{16}$.

Explanation of Model [6]

The Moore neighbourhood plus the current cell form a 3×3 table, to which three separate convolution filters are applied to each of the 16 channels. The convolutional filters are the Sobel filters, which measure the partial derivatives in the x and y direction, plus an indicator of the current cell:

$$\mathrm{Sobel}_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \mathrm{Sobel}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \mathrm{Id} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

These are then used to create a Perception vector with $16 \times 3 = 48$ components. This vector is passed through a neural network with a Dense-128 layer followed by an RELU, and then another Dense-16 layer.

This is applied as a residual neural network, so the final δ values are added to the current cell vector in $\mathbb{R}^{16}$. There are two exceptions: we don't assume the cells have a global clock and randomly dropout some updates.

Explanation of Model [6]

The Moore neighbourhood plus the current cell form a 3×3 table, to which three separate convolution filters are applied to each of the 16 channels. The convolutional filters are the Sobel filters, which measure the partial derivatives in the x and y direction, plus an indicator of the current cell:

$$\mathrm{Sobel}_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \mathrm{Sobel}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \mathrm{Id} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

These are then used to create a Perception vector with $16 \times 3 = 48$ components. This vector is passed through a neural network with a Dense-128 layer followed by an RELU, and then another Dense-16 layer.

This is applied as a residual neural network, so the final δ values are added to the current cell vector in $\mathbb{R}^{16}$. There are two exceptions: we don't assume the cells have a global clock and randomly dropout some updates. Secondly, if all the cells in the Moore neighbourhood are 'dead' ($\alpha < 0.1$), then all channels are set to 0 so they don't participate in the computation.

Experiments [6]

- In the first experiment, they started with a single pixel on the grid and learned how to grow various images such as a smiley face, a lizard, and so forth. They used backpropagation and L_2 norm for the difference between the resulting image of the CAs model and the target image.

Experiments [6]

- In the first experiment, they started with a single pixel on the grid and learned how to grow various images such as a smiley face, a lizard, and so forth. They used backpropagation and L_2 norm for the difference between the resulting image of the CAs model and the target image.
- In the second experiment, they attempted to create a model that would persist over time. To do this, they attempted to find an *attractor* in the dynamical systems sense. They periodically sampled the loss from random states during the iteration so that a large number of states would learn to go to the target image.

Experiments [6]

- In the first experiment, they started with a single pixel on the grid and learned how to grow various images such as a smiley face, a lizard, and so forth. They used backpropagation and L_2 norm for the difference between the resulting image of the CAs model and the target image.
- In the second experiment, they attempted to create a model that would persist over time. To do this, they attempted to find an *attractor* in the dynamical systems sense. They periodically sampled the loss from random states during the iteration so that a large number of states would learn to go to the target image.
- In the third experiment, they attempted to increase the basin of attraction by randomly damaging the images by erasing parts.

Experiments [6]

- In the first experiment, they started with a single pixel on the grid and learned how to grow various images such as a smiley face, a lizard, and so forth. They used backpropagation and L_2 norm for the difference between the resulting image of the CAs model and the target image.
- In the second experiment, they attempted to create a model that would persist over time. To do this, they attempted to find an *attractor* in the dynamical systems sense. They periodically sampled the loss from random states during the iteration so that a large number of states would learn to go to the target image.
- In the third experiment, they attempted to increase the basin of attraction by randomly damaging the images by erasing parts.
- Finally in the fourth experiment, they were able to generalize over rotations by multiplying the Sobel filters by the rotation matrix

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Self-Organizing in ALife [7]

- A self-organizing system is “one whose dynamics lead it to decrease its entropy or increase its information content" not by a central controller.

Self-Organizing in ALife [7]

- A self-organizing system is “one whose dynamics lead it to decrease its entropy or increase its information content" not by a central controller.
- Evolution typically has a central controller by choosing the most fit individuals. Open-ended evolution takes away this central controller by allowing for a mixed population of interacting individuals.

Self-Organizing in ALife [7]

- A self-organizing system is “one whose dynamics lead it to decrease its entropy or increase its information content" not by a central controller.
- Evolution typically has a central controller by choosing the most fit individuals. Open-ended evolution takes away this central controller by allowing for a mixed population of interacting individuals.
- **Soft** ALife: Computer simulations, where CA is most prominently used.
- Partial Differential Equations are the continuous counterpart to CAs and are also shown to have self-organizing properties.
- Also found in simulation models of artificial societies.

Self-Organizing in ALife [7]

- A self-organizing system is “one whose dynamics lead it to decrease its entropy or increase its information content" not by a central controller.
- Evolution typically has a central controller by choosing the most fit individuals. Open-ended evolution takes away this central controller by allowing for a mixed population of interacting individuals.
- **Soft** ALife: Computer simulations, where CA is most prominently used.
- Partial Differential Equations are the continuous counterpart to CAs and are also shown to have self-organizing properties.
- Also found in simulation models of artificial societies.
- **Hard** ALife: Robotics. Much more realistic than soft alife, since it captures physical phenomena that computer simulations can't.
- Controllers can be designed with methods such as artificial evolution.
- Can solve collective-decision making problems like avoiding obstacles.

Self-Organizing in ALife [7]

- A self-organizing system is “one whose dynamics lead it to decrease its entropy or increase its information content" not by a central controller.
- Evolution typically has a central controller by choosing the most fit individuals. Open-ended evolution takes away this central controller by allowing for a mixed population of interacting individuals.
- **Soft** ALife: Computer simulations, where CA is most prominently used.
- Partial Differential Equations are the continuous counterpart to CAs and are also shown to have self-organizing properties.
- Also found in simulation models of artificial societies.
- **Hard** ALife: Robotics. Much more realistic than soft alife, since it captures physical phenomena that computer simulations can't.
- Controllers can be designed with methods such as artificial evolution.
- Can solve collective-decision making problems like avoiding obstacles.
- **Wet** ALife: Physico-chemical systems such as the Belousov-Zhabotinsky reaction (the play example in the CDL paper).

Self-Organizing in ALife [7]

- A self-organizing system is “one whose dynamics lead it to decrease its entropy or increase its information content" not by a central controller.
- Evolution typically has a central controller by choosing the most fit individuals. Open-ended evolution takes away this central controller by allowing for a mixed population of interacting individuals.
- **Soft** ALife: Computer simulations, where CA is most prominently used.
- Partial Differential Equations are the continuous counterpart to CAs and are also shown to have self-organizing properties.
- Also found in simulation models of artificial societies.
- **Hard** ALife: Robotics. Much more realistic than soft alife, since it captures physical phenomena that computer simulations can't.
- Controllers can be designed with methods such as artificial evolution.
- Can solve collective-decision making problems like avoiding obstacles.
- **Wet** ALife: Physico-chemical systems such as the Belousov-Zhabotinsky reaction (the play example in the CDL paper).

Other Representations of Cellular Automata

- [8] attempts to solve the problem of given the history of some CA, determine the rule that was used to generate it.

Other Representations of Cellular Automata

- [8] attempts to solve the problem of given the history of some CA, determine the rule that was used to generate it.
- They focused on class 3 and class 4 CAs: rules that lead to chaotic behaviour and complex long-lived behaviour (such as the game of life).

Other Representations of Cellular Automata

- [8] attempts to solve the problem of given the history of some CA, determine the rule that was used to generate it.
- They focused on class 3 and class 4 CAs: rules that lead to chaotic behaviour and complex long-lived behaviour (such as the game of life).
- In the former case our goal is to learn about the attractors after a fixed number of steps, in the latter case we only focus on the transients.

Other Representations of Cellular Automata

- [8] attempts to solve the problem of given the history of some CA, determine the rule that was used to generate it.
- They focused on class 3 and class 4 CAs: rules that lead to chaotic behaviour and complex long-lived behaviour (such as the game of life).
- In the former case our goal is to learn about the attractors after a fixed number of steps, in the latter case we only focus on the transients.
- They trained neural networks on three tasks: learning the rule, learning the dynamics from any initial condition after a few steps, and learning to continue the dynamics only from the given initial condition.

Other Representations of Cellular Automata

- [8] attempts to solve the problem of given the history of some CA, determine the rule that was used to generate it.
- They focused on class 3 and class 4 CAs: rules that lead to chaotic behaviour and complex long-lived behaviour (such as the game of life).
- In the former case our goal is to learn about the attractors after a fixed number of steps, in the latter case we only focus on the transients.
- They trained neural networks on three tasks: learning the rule, learning the dynamics from any initial condition after a few steps, and learning to continue the dynamics only from the given initial condition.
- With weight sharing, they found learning the dynamics on the $1d$ rules to be simple. Furthermore, they learned the rules for the game of life. Without weight sharing they found these tasks far more difficult.

Other Representations of Cellular Automata

- [8] attempts to solve the problem of given the history of some CA, determine the rule that was used to generate it.
- They focused on class 3 and class 4 CAs: rules that lead to chaotic behaviour and complex long-lived behaviour (such as the game of life).
- In the former case our goal is to learn about the attractors after a fixed number of steps, in the latter case we only focus on the transients.
- They trained neural networks on three tasks: learning the rule, learning the dynamics from any initial condition after a few steps, and learning to continue the dynamics only from the given initial condition.
- With weight sharing, they found learning the dynamics on the $1d$ rules to be simple. Furthermore, they learned the rules for the game of life. Without weight sharing they found these tasks far more difficult.
- [9] represents cellular automata as Convolutional Neural Networks.

Outline

Sources I

A. Nowe, K. Verbeeck, and M. Peeters
Learning Automata as a Basis for Multi Agent Reinforcement Learning
International Workshop on Learning and Adaption in Multi-Agent Systems LAMAS 2005: Learning and Adaption in Multi-Agent Systems.

H. Beigy and M. R. Meybodi
A Mathematical Framework For Cellular Learning Automata
Advances in Complex Systems 2004.

R. Vafashoar and M. Reza Meybodi
Reinforcement learning in learning automata and cellular learning automata via multiple reinforcement signals.
Science Direct, Knowledge-Based Systems 2018.

Y. Tavakoli, H. Seyyed Javadi, S. Adabi.
A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with A Common Goal.
IJCSNS, 2008.

S. Ruiz and B. Hernandez
A Hybrid Reinforcement Learning and Cellular Automata Model for Crowd Simulation on the GPU
High Performance Computing. CARLA 2018. Communications in Computer and Information Science, vol 979. Springer.

A. Mordvintsev E. Randazzo E. Niklasson M. Levin
Growing Neural Cellular Automata: Differentiable Model of Morphogenesis
Distill Publication 2020.

C. Gershenson, V. Trianni, J. Werfel and H. Sayama
Self-Organization and Artificial Life: A Review
ALife 2018.

N. H Wulff and J. A Hertz
Learning Cellular Automaton Dynamics with Neural Networks
Proceedings of the 5th International Conference on Neural Information Processing Systems 1992.

W. Gilpin
Cellular automata as convolutional neural networks
Stanford

Evolutionary Techniques

W. Elmenreich and I. Fehervari
Evolving Self-organizing Cellular Automata Based on Neural Network Genotypes
Self-Organizing Systems 2011.

S. Nichele, M.B. Ose, S. Risi and G. Tufte
CA-NEAT: Evolved Compositional Pattern Producing Networks for Cellular Automata Morphogenesis and Replication
IEEE Transactions on Cognitive and Developmental Systems 2018.

J. Miller
Evolving a Self-Repairing, Self-Regulating, French Flag Organism
University of York 2004.