Week 6

Artificial Life

Goal for each pre:
5 minutes information
2 minutes “treasure”
3 minutes Discussion
------
10 minutes for logistics, feedback

CSSE290

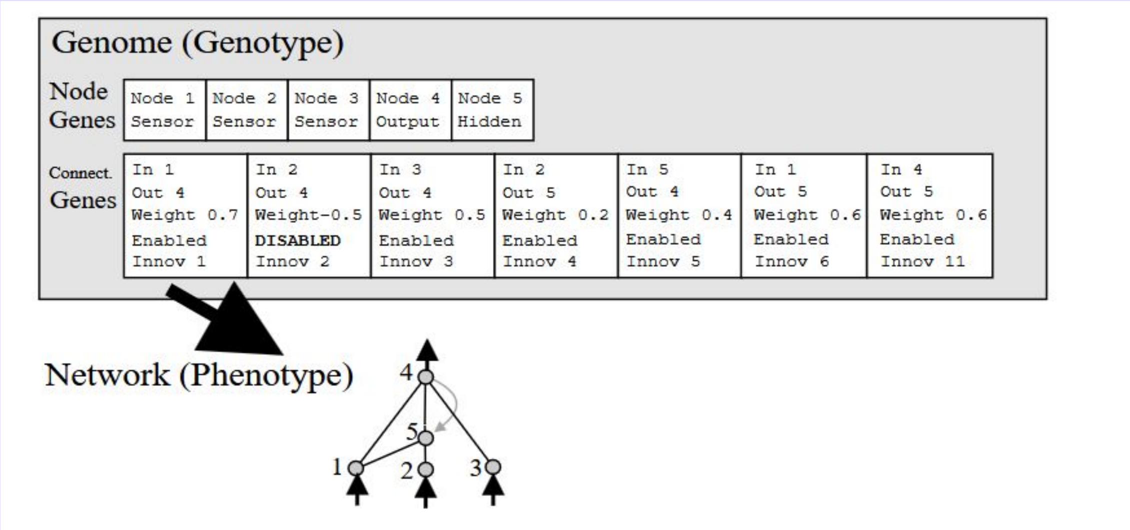
01

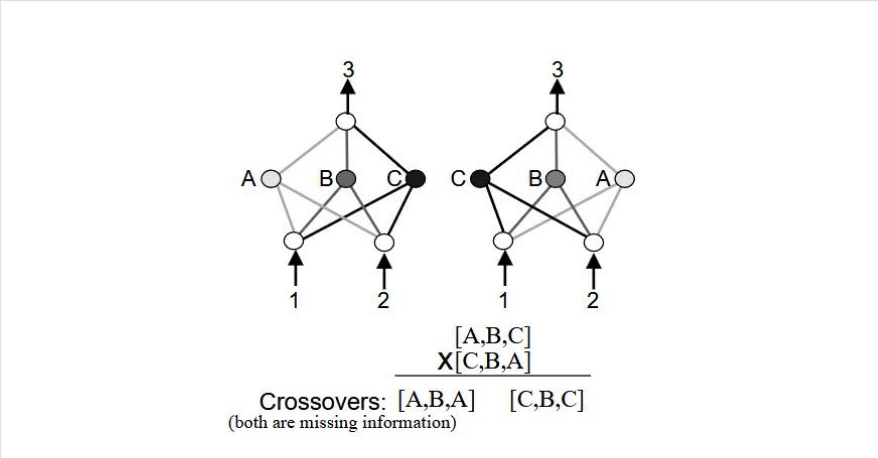
Will Greenlee

Will Greenlee

NEAT

- A way of encoding, mutating, and recombining neural networks.
- evolves both network weights and topology (other approaches at the time used fixed topology)
- Historical Markings
- Speciation
- Complexification
- Direct encoding, no devo
- Networks are grouped using a compatibility distance based on gene differences and weight similarity

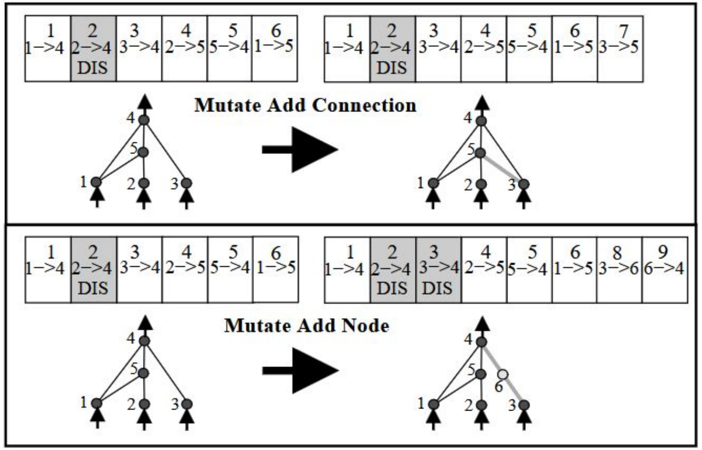


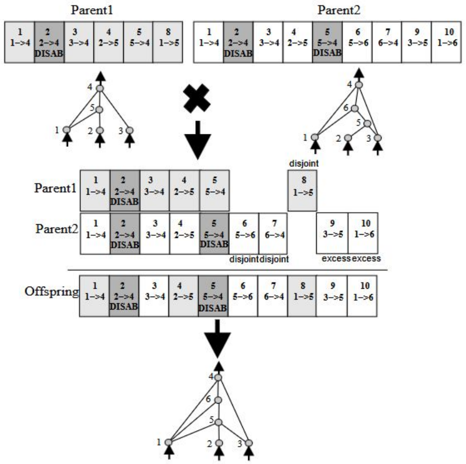


Will Greenlee

Mutation+Crossover

- Mutations
  - Add Connection
  - Add Node
- Crossover & Historical Markings
  - Genes with matching innovation numbers are aligned during crossover.
  - Matching genes are randomly inherited; disjoint/excess genes come from one of the parents.
  - Prevents destructive crossover from differing topologies.





Will Greenlee

CPPNs

- Biological development reuses genes to build complex phenotypes efficiently.
- CPPNs abstract development by directly generating patterns from coordinate inputs.
- Avoids time-based simulation or cell interaction while capturing developmental regularities.
- Map spatial coordinates to phenotype properties (e.g., color, presence).
- Generate the final form directly without simulating growth over time (no "temporal unfolding").
- CPPNs can be evolved with NEAT, complexifying networks over time.

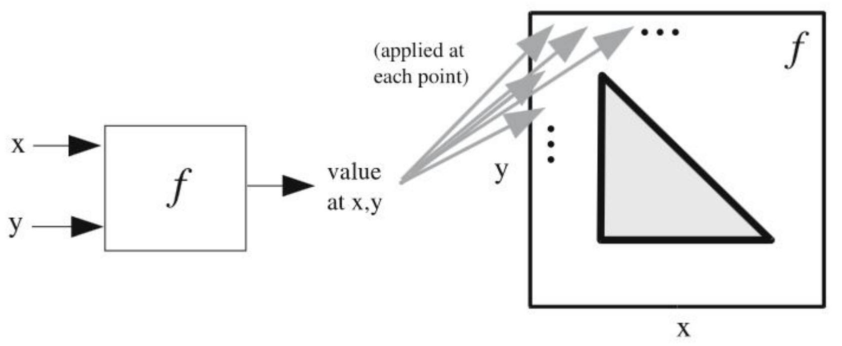


Fig. 1 *A function produces a phenotype*. The function *f* takes arguments *x* and *y*, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of *f* at that coordinate, the result is a pattern, which can be conceived as a phenotype whose genotype is *f*. In this example, *f* produces a triangular phenotype

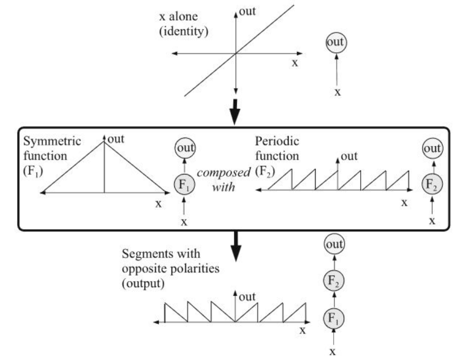


Fig. 3 *Composing gradient functions*. This example illustrates how a simple composition of functions in one dimension can produce a pattern with multiple regularities. A network representation of each composition is depicted to the right of each function graph. The initial asymmetric gradient *x* is composed with both a symmetric function (F_1) and a periodic function (F_2), resulting in two sets of segments with opposite polarity. Thus, new coordinate frames can be built upon preexisting ones without local interaction

Will Greenlee

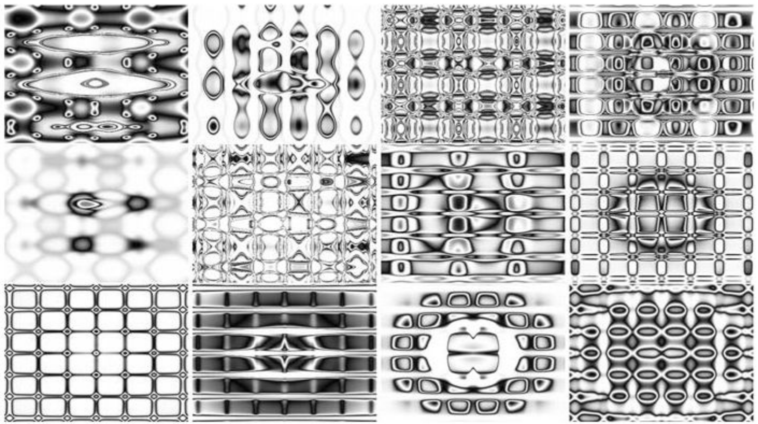
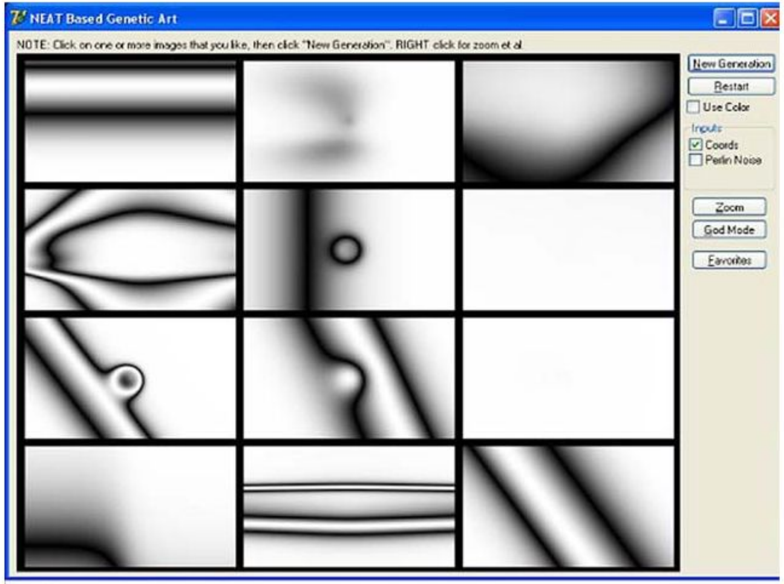


Fig. 14 *Ubiquitous repetition with variation.* CPPNs easily produce such patterns when the right coordinate frames are introduced. As the figure shows, there are a great variety of such images



(a) DelphiNEAT-based Genetic Art (DNGA)

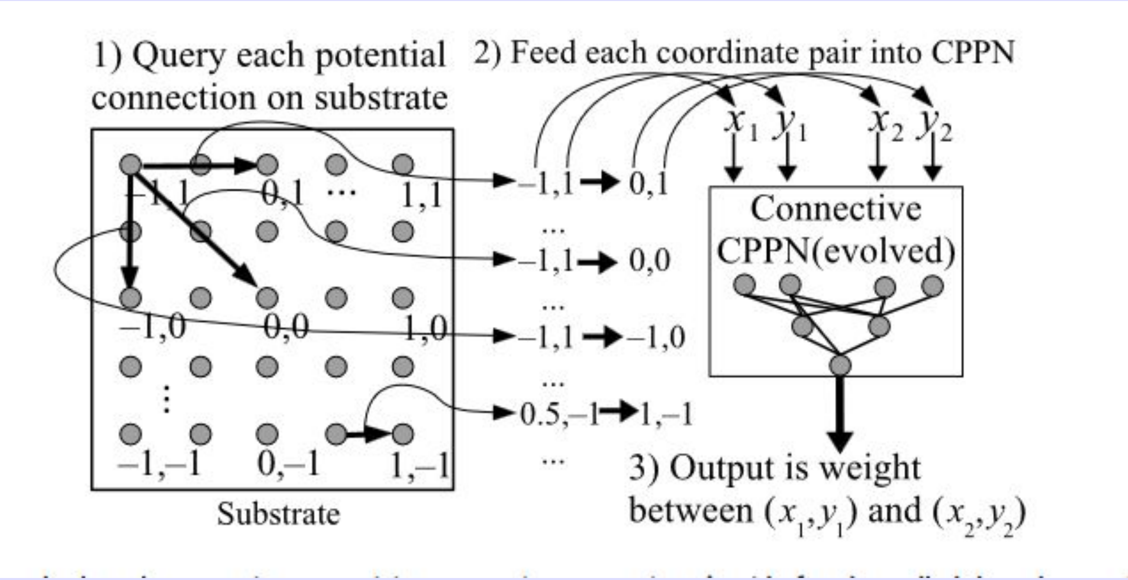
Art (Treasure)

Will Greenlee

HyperNEAT

Putting it all together!

- NEAT used to generate/evolve CPPN
- CPPN used to generate heat map of a fixed space
- HyperNEAT uses that heat map (directly queried through the CPPN) to connect neurons and have various different weights depending on the difference between the points.
- Different substrate types
    - Evolvable!
- Scalable (no temporal dependence)



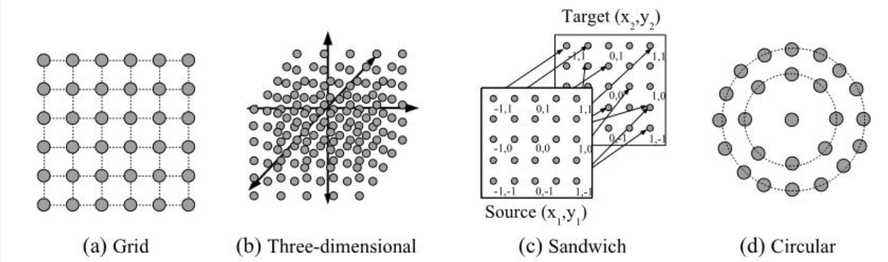


Figure 6. Alternative substrate configurations. This figure shows (a) the original grid configuration introduced in Figure 4, (b) a three-dimensional configuration of nodes centered at (0, 0, 0), (c) a *state-space sandwich* configuration in which a source sheet of neurons connects directly to a target sheet, and (d) a circular configuration. Different configurations are likely suited to problems with different geometric properties.

Discussion Slide

02

Steven Johnson

Steven

NCA - early paper - 2015

- Multi-layer feed-forward ANN chosen over perceptron
  - "can learn any function, as long as the hidden layer contains enough neurons"
- Fitness function
  - Test set not chosen due to being over-restrictive
  - Two complexity options instead:
    - State changes within each column of the CA matrix
    - Kolmogorov complexity
      - # of bits of shortest computer program that can fully generate the final sequence

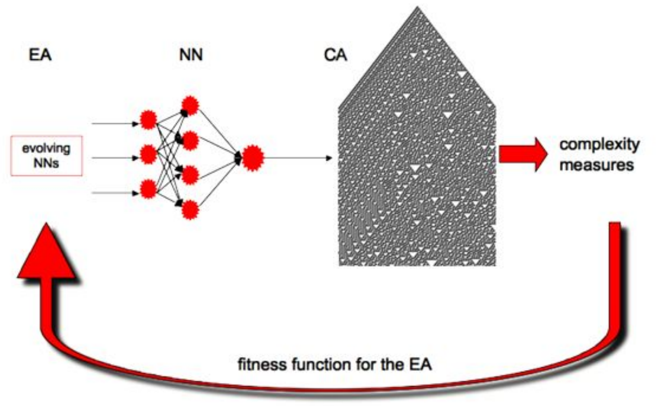


Fig. 6. The evolution of the Neuro-Cellular Automata.

Steven

NCA - early paper - 2015

"A novel kind of Cellular Automata was introduced in this work. The common way of representing the set of rules is replaced by an Artificial Neural Network. This provides a **biologically inspired automata**." pg. 11

Steven

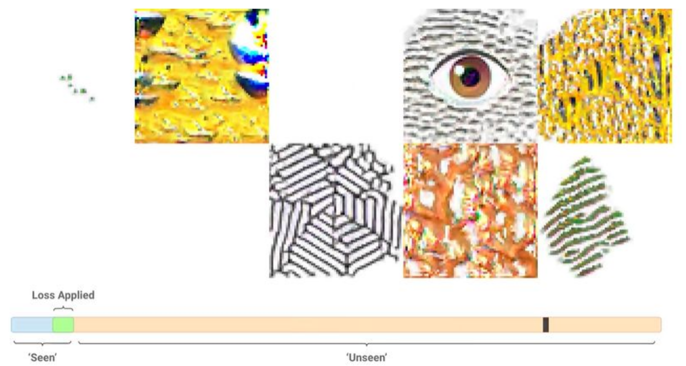
Newer NCA Mordvinstev - 2020

- CA rule in this experiment composed of 4 things:
- Perception
  - defines what each cell perceives of the environment around it
  - implemented with a 3x3 convolution with a fixed kernel
- Update rule
  - every cell runs the same update rule
  - outputs an incremental update to the cell's state
- Stochastic cell update
  - random per-cell mask to update vectors
  - used to mimic the randomness of real life
- Living cell masking
  - explicitly setting all channels of empty cells to zero to avoid any hidden states

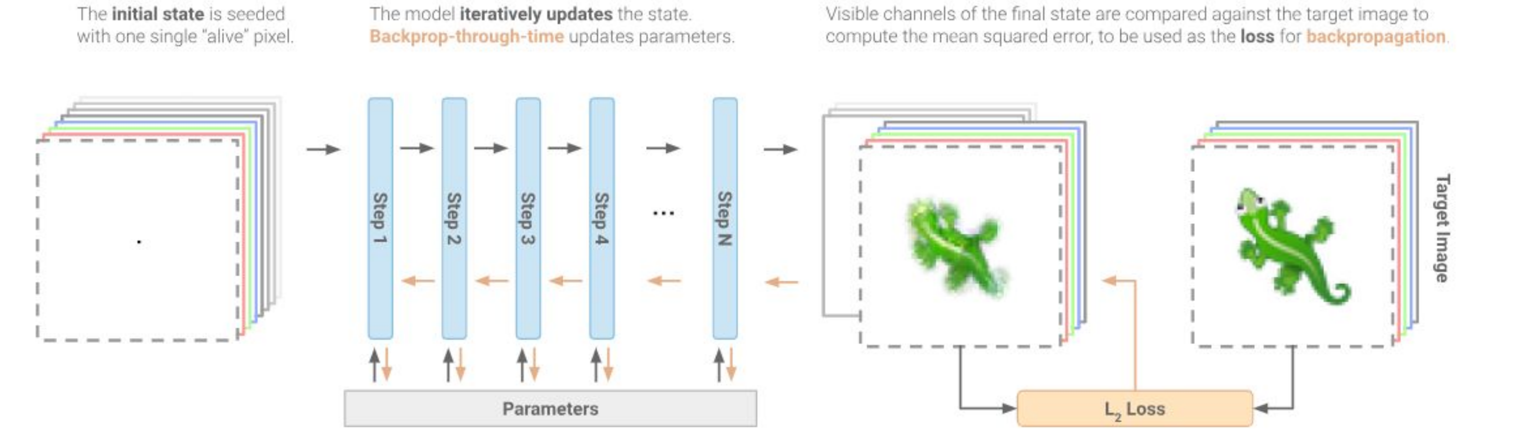
Steven

Newer NCA Mordvinstev - 2020

- Experiment 1: Learning to Grow
  - Optimize the update rule using backpropagation-through-time
  - Select the best performing
  - Run for steps beyond what was trained on



Many of the patterns exhibit instability for longer time periods.



Training regime for learning a target pattern.

Steven

Newer NCA Mordvinstev - 2020

- Experiment 2: What persists, exists (the fix)
  - First thought: train for more steps
    - Not optimal due to long training time and memory requirements
  - Solution: **“sample pool” based strategy**
    - Sample the previous final states and use them as new starting points to force our CA to learn how to persist or even improve an already formed pattern
      - Prevent “catastrophic forgetting” by replacing one sample with the original, single-pixel seed state

CA behaviour at training steps 100, 500, 1000, 4000.

Steven

Newer NCA Mordvinstev - 2020

- Experiment 3: Learning to regenerate
  - Lizards naturally most regenerative
  - Only trained for growth, not repair, leaving regeneration open-ended
  - Solution: **increase the basin of attraction for our target pattern** (# of configurations that lead to our target)
    - Accomplished by damaging a few pool-sampled states before each training step
    - Sample 8 states from the pool
      - Replace highest-loss sample with the seed state
      - Damage the lowest-loss states by setting a random circular region within the pattern to zeros

Patterns exhibit some regenerative properties upon being damaged, but not full re-growth.

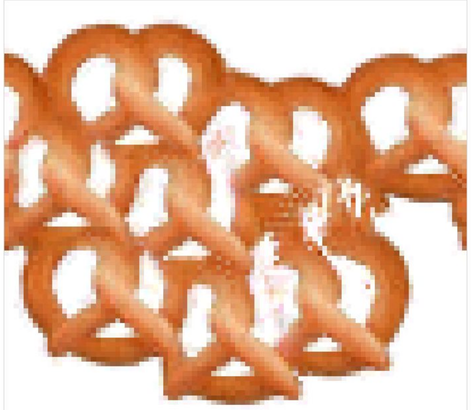
Damaging samples in the pool encourages the learning of robust regenerative qualities. Row 1 are samples from the pool, Row 2 are their respective states after iterating the model.

Steven

Newer NCA Mordvinstev - 2020

- Experiment 4: Rotating the perceptive field
  - Solution: Rotating the Sobel kernels
    - (The kernels used in the convolution perception method mentioned earlier)
  - Problem: Rotating pixel based graphics involves interpolating b/t pixels to get the desired result
    - Will likely overlap several pixels
  - Like the oddly regenerative lizards, successful growth in some rotations suggest a certain robustness to the underlying conditions

Rotating the axis along which the perception step computes gradients brings about rotated versions of the pattern.

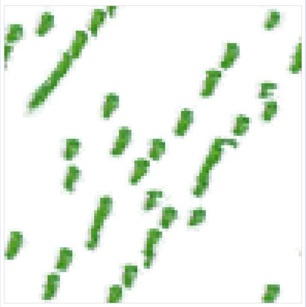


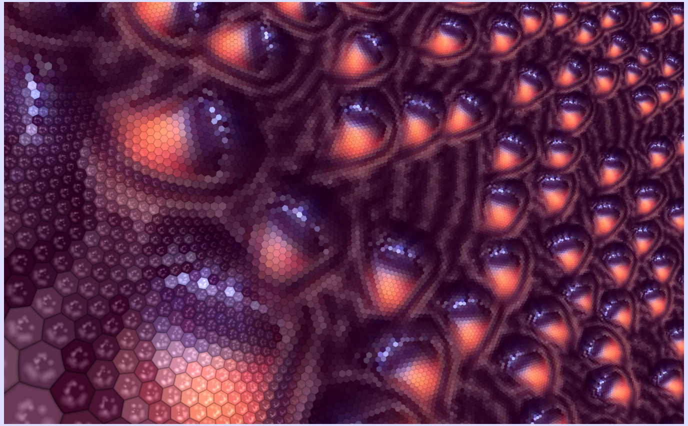
Pretzel after being rotated 360 degrees

Steven

Extra Inspiration from Alex Mordvintsev

- https://x.com/zzznah?lang=en
  - His twitter just has a bunch of cool stuff to scroll through
  - For example: NCA has
    - Hexells
    - znah.net/hexells
    - Further hexell reading





Fun freeze frame from deleting part of the lizard

03

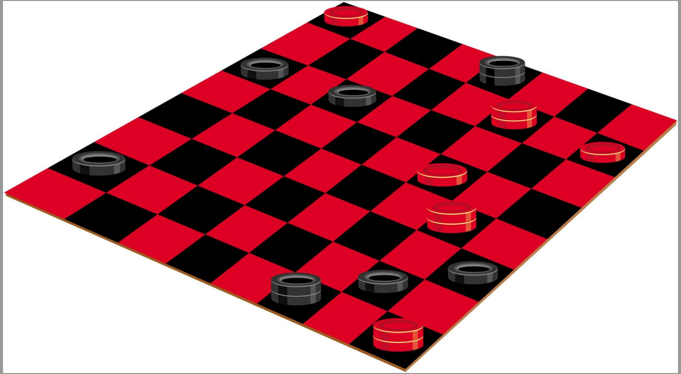
Alex Brickley

Meta-Learning

- In world full of changes and dynamic issues, programs will need to learn as they function without forgetting prior information.
- Limited storage and energy.
- This process is known as lifelong learning- three main categories of AI
  - **rehearsal**: which store or generate data from past tasks for replay
  - **architectural**: which expand the model parameters
  - **regularization-based**: which penalize changes to parameters important to past tasks or use meta-learning

Meta-Learning

Board games examples



Key features of lifelong learning

Transfer and adaptation.

- Moving knowledge or reusing knowledge outside of the original environment
- Ability to adapt to a new environment

Task-agnostic learning.

- When dropped in the real world training is not labeled
- A model needs to be able to perform well without a specific task or being told when tasks are switched

Overcoming catastrophic forgetting.

- Learning new things without forgetting old things
- This is not an issue of total memory
- This is an issue of rewriting weights

Noise tolerance.

- Most AIs are trained on cleaned data
- The real world has noise
- Models trained without noise normally poorly with it

Exploiting task similarity.

- Learning about one task from another task
- The more tasks learned, the better tasks should go

Resource efficiency and sustainability.

- Storing everything is not practical
- Energy is a limited resource
    - (or at least our budget is)

Biological mechanisms	Key features					
	Transfer and adaptation	Overcoming catastrophic forgetting	Exploiting task similarity	Task-agnostic learning	Noise tolerance	Resource efficiency and sustainability
Neurogenesis	●	●	●	●		●
Episodic replay		●		●		●
Metaplasticity		●		●		●
Neuromodulation	●	●				
Context-dependent perception and gating	●	●	●	●	●	
Hierarchical distributed systems			●		●	
Cognition outside the brain	●		●		●	
Reconfigurable organisms	●	●	●			
Multisensory integration			●		●	

Fig. 2 | Biological mechanisms that support lifelong learning. The matrix illustrates the relationships between the key features of lifelong learning (along the top) and biological mechanisms (along the left edge). A coloured bullet in a cell signifies that the biological mechanism indicated to its left is thought to contribute to the key feature that labels the corresponding column (but not necessarily that the mechanism by itself is sufficient to realize that feature).

Biologically inspired mechanisms	Key features: Transfer and adaptation	Overcoming catastrophic forgetting	Exploiting task similarity	Task-agnostic learning	Noise tolerance	Resource efficiency and sustainability
Neurogenesis		169–174	234	161		174,201,202
Episodic replay		54,175,176,179, 180		54,176	176,177	53,54,175,176, 179,180,203
Metaplasticity		67,89,181–185		7,181,185,198		89,181–183,198
Neuromodulation	70,78,84–86,88, 89,157,159,160	78,79,84,89,164	89	78,159	78,158,199	89
Context-dependent perception and gating	78,79,158,161–167	78,168	79,162–166	70,161	158,162,163	
Hierarchical distributed systems			188–191		113,191,200	191
Cognition outside the brain			195–197			
Reconfigurable organisms	139		139,147		139,147	139,147
Multisensory integration			152,155,192,193		113,162	

Evaluation: Dataset Category	References
Image recognition	7,54,70,78,84, 88,89,160,165, 166,168,171,172, 175–177,179,181, 183,185,198,201, 202,234
Environment interaction	78,79,157,159, 160,163,170,171, 174,180,182,184, 192
Biological simulation	139,147,195–197
Robotics	113,152,160, 164,189,190
Other	53,67,85,86,155, 158,'161,162,167, 169,173,188,191, 193,199,200,203

04

Solving Catastrophic Forgetting

Dominic Reilly

Understanding Catastrophic Forgetting

What It Is:

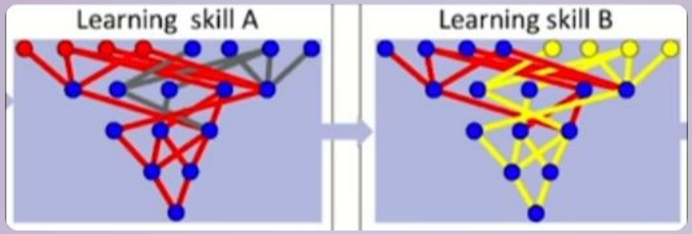
Sudden loss of previously learned tasks when training on new ones.

Why It Happens:

Shared network weights get overwritten during new learning.

Different from our Brains:

Our brains will use neuromodulation and specialized areas to learn multiple tasks



Learning to Continually Learn

Paper by Shawn Beaulieu, Ken Stanley, Jeff Clune

Presentation by Dominic Reilly

A Neuromodulated Meta-Learning Algorithm (ANML)

Neuromodulation-based system

Designed to solve continual learning and catastrophic forgetting in deep networks

ANML learns to control when and where neurons activate and learn

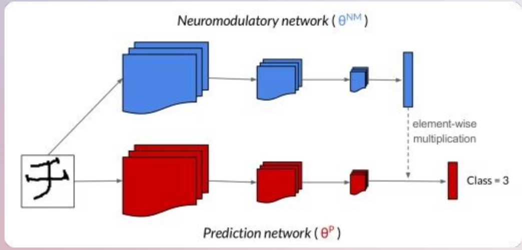
Architecture and Mechanisms

Prediction Learning Network (PLN)

- Standard neural network that performs the primary task

Neuromodulatory Network (NM)

- Secondary neural network that modulates activations in the PLN
- Its output determines which neurons in the PLN activate and learn



Mechanisms

Neuromodulatory Mechanism

- Neuromodulatory network modulates the forward pass of the PLN
- Neuron activations in PLN are multiplied by the NM's outputs (some value between 0-1)
- Initial PLN weights and NM parameters are "meta-learned"

Omniglot Dataset

- Sequentially present different characters from the Omniglot dataset
- Some real alphabets, some fictional
- For each character, there is dozens of examples
- Characters are not labeled until error calculation
- The NM network infers classes through training

Inner Loop: Task Learning

1. **Train the PLN using the current NM**

   NM stays frozen until this character is fully trained on

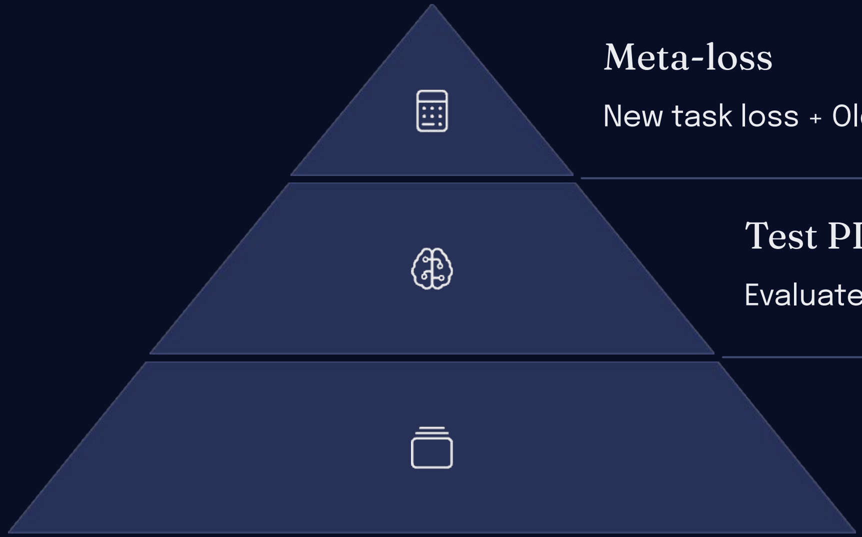
2. **Pass inputs through NM**

   Get neuromodulation gating signals that suppress/enhance parts of the PLN

3. **Process through gated PLN**

   Do a forward pass and backward pass on the gated PLN

Meta-Loss Calculation



Meta-loss

New task loss + Old tasks memory loss

Test PLN on new class

Evaluate performance on newly learned task

Test PLN on previous tasks

Evaluate memory retention of old tasks

This meta-loss calculation encourages learning new tasks while not forgetting old ones

Outer Loop: Meta-Update

Compute gradients

Calculate gradients of meta-loss

Update PLN weights

Adjust initial weights based on gradients

Update NM parameters

Adjust neuromodulatory network

Meta-Testing

Freeze NM, fine-tune PLN, evaluate

Outcomes

64%

ANML Accuracy

After learning 600 classes

18%

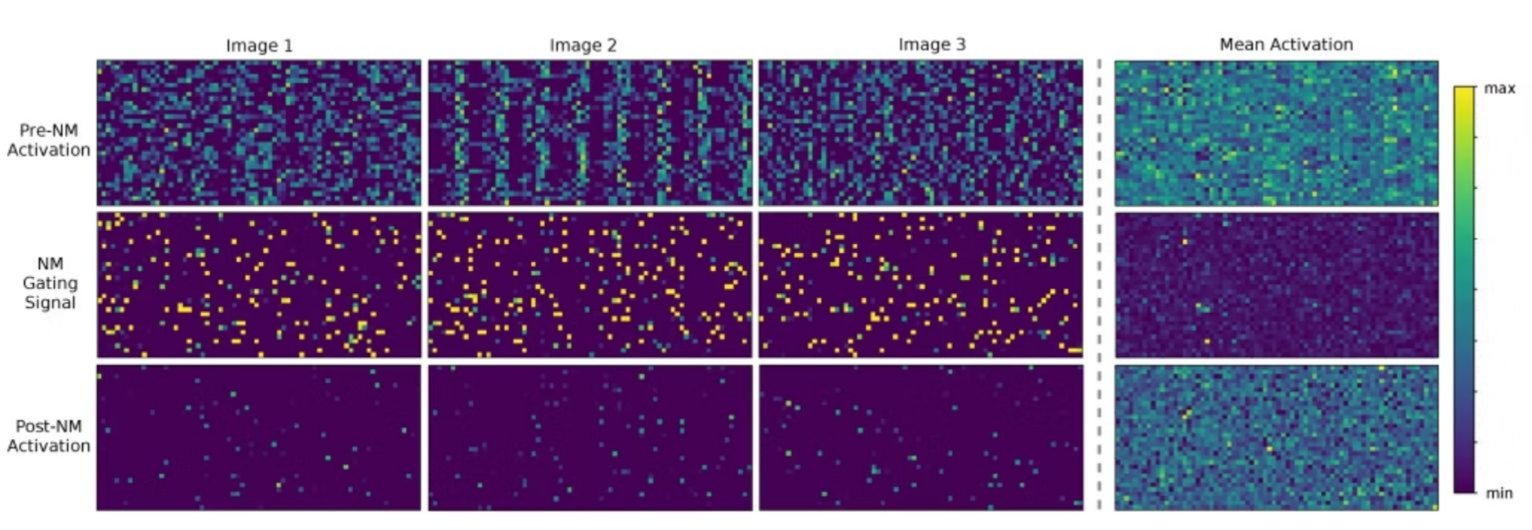
OML Accuracy

Previous state of the art model

6%

Active Neurons

After neuromodulation in PLN



Evolving Developmental Neural Networks to Solve Multiple Problems

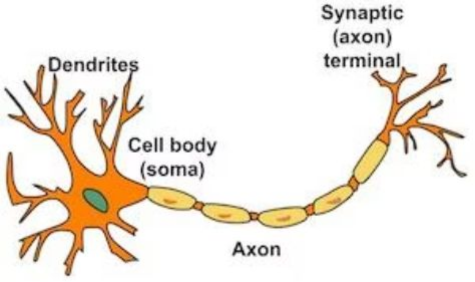
by Julian Francis Miller

Presentation by Dominic Reilly

Key Innovation

Developmental Neural Model

Developmental neural model where two evolving CGP programs control the growth, movement, replication, and death, of neurons and their connections



Soma Program

In the brain

- Main cell body
- Contains the nucleus and decides when to fire an action potential

In the soma program

- Governs neuron health, bias, position, replication, and death

Dendrite Program

In the brain

- Branching structures extending from the soma
- Receive signals from other neurons

In the dendrite program

- Governs dendrite health, weight, extension, replication, and death

The neural network develops over time through interaction of these programs, just like how our brains develop

Multiple ANNs are extracted from this network to solve problems

Pre-Learning Development

Random Initial Network

Start with a random initial network

Each problem has its own set of inputs and output neurons

Developmental Growth

For ~6 steps, the soma and dendrite programs (governed by their respective CGPs) grow and change without feedback

Neurons and dendrites move, change bias/weight, replicate, or die

Uncontrolled Development

No evaluation yet, just free uncontrolled development, like early embryo growth

Cartesian Genetic Programming

A form of evolutionary algorithm where programs are represented as directed graphs

Inputs

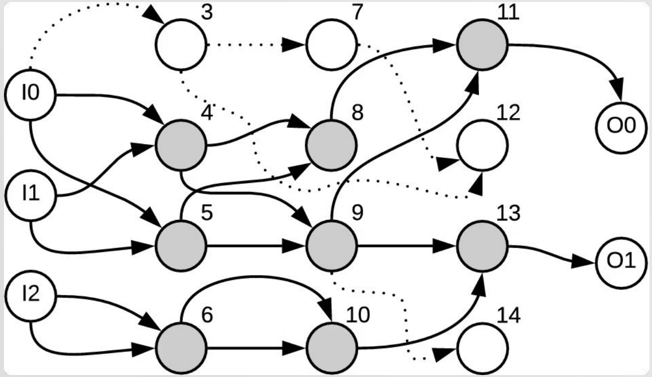
External data fed into the graph on the left side

Nodes

Perform operations on inputs or outputs from other nodes

Outputs

Selected from certain nodes



Learning Development

Feedback Integration

Now, the network will develop with feedback

ANN Extraction

After each timestep, extract ANNs for each problem we are trying to solve

Do this by snapping each dendrite to its nearest neighbor to the left (including inputs)

This forces a feedforward structure with no loops

Performance Testing

Test the performance on each problem

Network Adjustment

For the next step, our dendrites/soma programs adjust health, position, etc.

Pass position, health, weight, etc. of each soma/dendrite as inputs into the CGP network. Performance is an input

Outputs new positions, healths, weights, etc.

Evolution

Fitness Evaluation

After a few epochs, evaluate each problem and find the fitness of the current soma/dendrite programs

Evolutionary Strategy

Use an evolutionary strategy to evolve the programs

- Population of pairs of CGPs
- Mutate, test, select, and repeat

Evolved Programs

After evolution, you have two evolved programs (soma + dendrite) that can:

- Grow a brain from scratch
- Self-adapt during learning
- Solve multiple unrelated problems by extracting networks at runtime

Outcomes

Test Tasks

Model was tested on four tasks

Two classification problems

- Diabetes (binary)
- Glass (multiclass)

Reinforcement learning problems

- Ball throwing
- Double pole balancing

Performance Results

High success rate for single problems: ~85%

Incremental evolution improved performance with multiple tasks

Brains could reuse neurons

Only ~8 non-output neurons could handle multiple tasks

Statistic	Incremental Problem solving	Non-incremental Problem solving
	double pole (DP) ball throwing (BT)	double pole ball throwing
Mean	**0.5844**	0.4002
Median	**0.6050**	0.4449
Maximum	**0.7693**	0.6294
Minimum	0.2735	**0.2742**
No. solved DP	**8**	1
No. solved BT	1	**9**
	Glass diabetes train (test)	Glass diabetes train (test)
Mean	**0.6644 (0.6183)**	0.6561 (0.6009)
Median	**0.6599 (0.6274)**	0.6566 (0.5999)
Maximum	**0.7029 (0.6974)**	0.6948 (0.6502)
Minimum	**0.6342** (0.5158)	0.6015 (**0.5347**)

Wrapping Up

Connection to prior weeks?

Provide Peer Evaluation (including Self)

Portfolio Reflection Entry