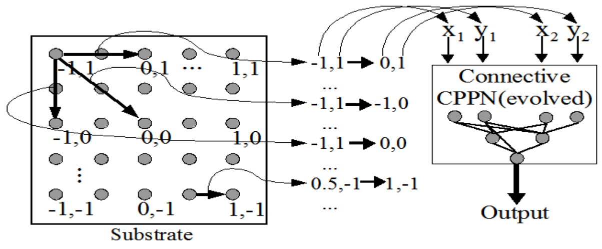
Introduction to Hypernetworks

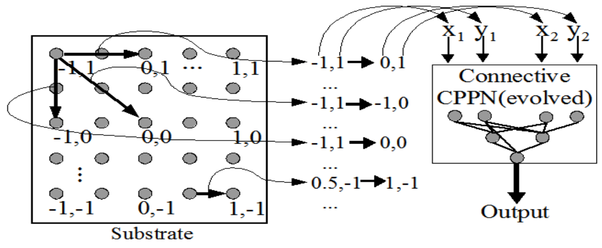
- Hypernetworks encode the weights of another network in their output.
  - Indirect encoding of the network that actually performs the task.
- No need for local interactions or temporal unfolding.
- Perform well in many standard benchmarks and extend to various domains like 3D robot morphologies.



(Stanley et al. 2009)

Compositional Pattern Producing Networks (CPPNs)

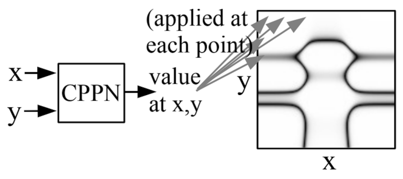
- Hypernetwork implementation: patterns are the weights.
- The idea is to embed the task network into a substrate.
  - Each neuron then has specific coordinates.
  - Connections can be specified geometrically.

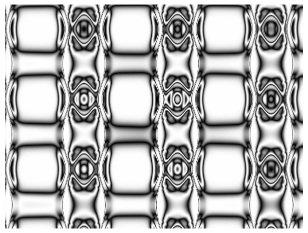


(Stanley et al. 2009)

CPPN Function Composition

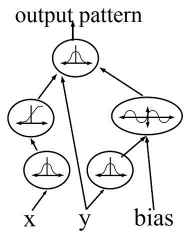
- CPPNs generate regular patterns, e.g. in 2D:





(D'Ambrosio et al. 2014; Stanley 2007)

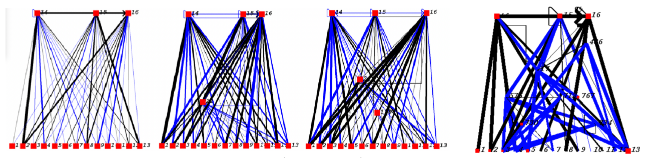
- They result from composition of activation functions (e.g., Gaussian, sigmoid, sine).
- They mimic natural phenomena like symmetry and repetition with variation.
- They can be useful as weights, e.g. similar to convolution.
- But is is also fun just to evolve images!



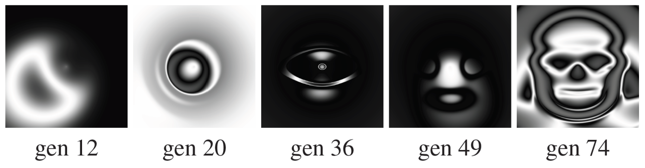
(D'Ambrosio et al. 2014)

Evolving CPPNs with NEAT

- CPPNs are traditionally evolved using the NEAT algorithm.
- NEAT allows CPPNs to complexify gradually and evolve increasingly complex patterns.
- Structural mutations add nodes with different activation functions to evolve new patterns.



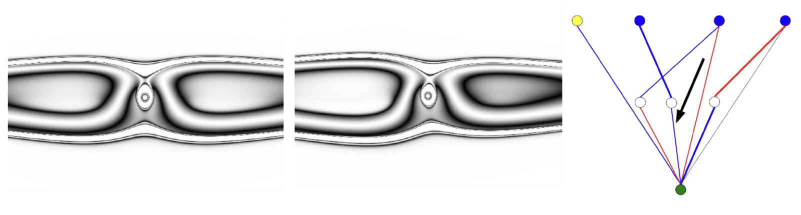
(Stanley 2004)



(Stanley 2007)

Repetition with Variation in CPPNs

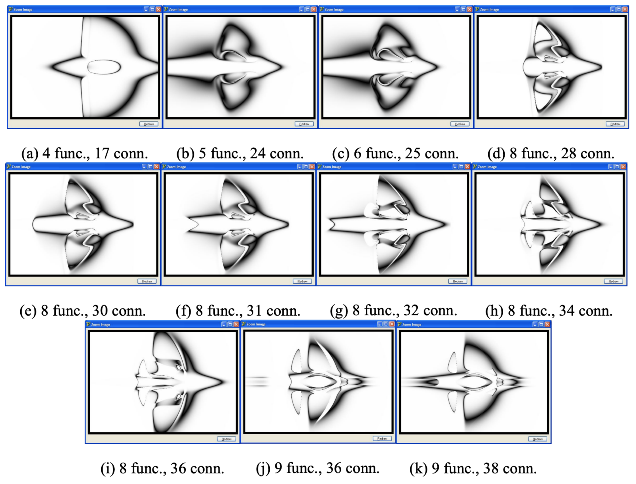
- CPPNs can encode symmetric patterns, like mirror-image sunglasses.
- Altering a single connection can introduce subtle variations in symmetry.
- These changes preserve overall coherence of the pattern.


(c) (d) (e)

(Stanley 2007)

Elaborating on Discovered Patterns

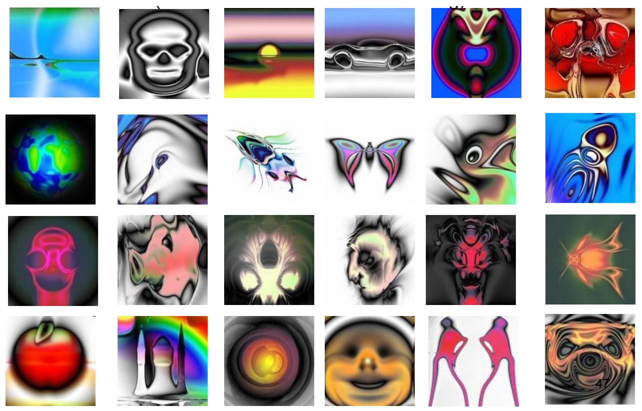
- CPPNs can evolve patterns and elaborate upon them across generations.
- Early designs are refined into more complex structures while preserving their core.
- This property mirrors how biological structures evolve over time.


(a) 4 func., 17 conn. (b) 5 func., 24 conn. (c) 6 func., 25 conn. (d) 8 func., 28 conn.
(e) 8 func., 30 conn. (f) 8 func., 31 conn. (g) 8 func., 32 conn. (h) 8 func., 34 conn.
(i) 8 func., 36 conn. (j) 9 func., 36 conn. (k) 9 func., 38 conn.

(Stanley 2007)

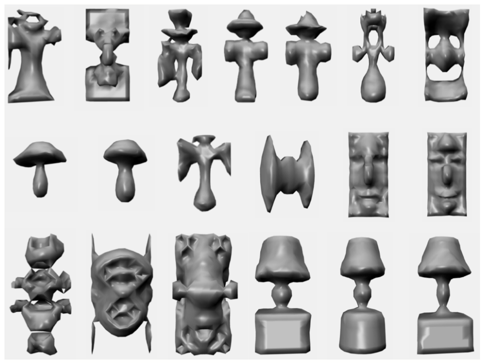
Evolving 2D Pictures

- With human selection, can evolve a range of images
- E.g. Picbreeder (https://nbenko1.github.io/)


(Stanley 2014)

Evolving 3D Forms

- Adding a third z-input enables generating 3D structures and morphologies
- E.g. endlessforms.com
- Can be 3D-printed in silver, bronze, plastic...

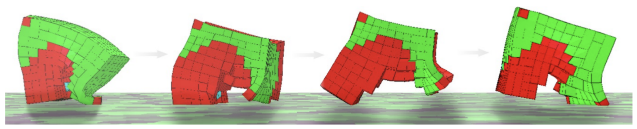


(Clune et al. 2012)

Evolving 4-D Forms: Virtual Creatures that Move

- Virtual creatures are digital entities interacting in a simulated environment.
- The challenge is to create not just viable forms but also effective behaviors —without a brain or a controller!

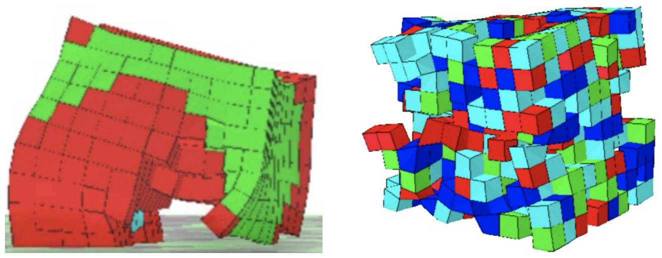
- Example: 3D soft robots are made of voxels of four different materials.
- Voxels are assigned properties like actuation and rigidity.
- The pattern of these materials determines the robot's behavior, such as locomotion.



(Cheney et al. 2014)

Comparing CPPN Encoding vs Direct Encoding

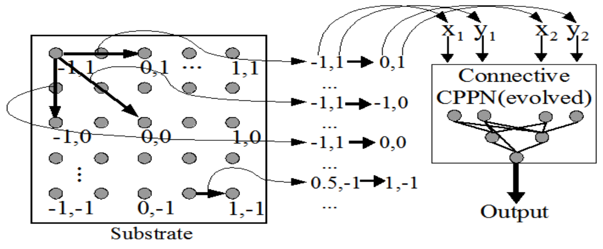
- In CPPN-based encoding, symmetries and repeating motifs are easily produced.
- CPPNs generate globally coordinated behaviors necessary for efficient locomotion.
- Direct encoding optimizes each voxel independently, often leading to irregular structures.
- Direct evolution fails to discover locomotion.



Demo link: `https://youtu.be/EXuR_soDnFo`

Evolving Neural Networks: HyperNEAT

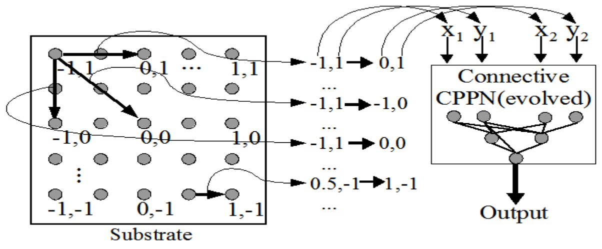
- HyperNEAT applies CPPNs to generate neural network connectivity patterns.
- It exploits the geometry of input and output domains to create regularity in connections.
- Intended take advantage of repeating patterns and symmetry in biological neural networks.



(Stanley et al. 2009)

HyperNEAT Substrates

- A substrate defines the spatial arrangement and roles of neurons.
- The CPPN is queried with each pair of neuron positions, producing a weight for the connection.
- Can discover patterns such as convolution and attention automatically.
- Often a 2D plane is a natural substrate:

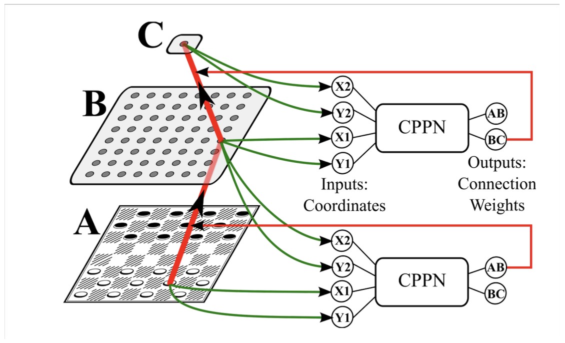


(Stanley et al. 2009)

A HyperNEAT substrate arranged in a 2D plane.

Taking Advantage of the Substrate: Checkers Game

- In the checkers game task, the substrate mirrors the geometry of the checkerboard.
- One CPPN with separate outputs for input (AB) and output (BC) weights.
- The task network evaluates how good the board position is (C).
- The network evolves to generalize across the board.

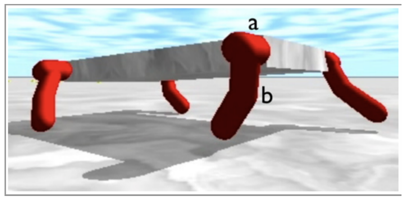


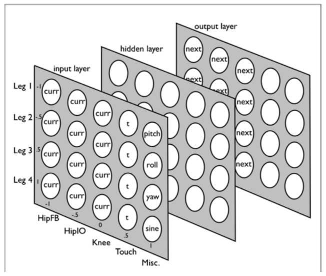
(Gauci & Stanley 2010)

HyperNEAT substrate designed for the checkers game.

Taking Advantage of the Substrate: Quadruped Robot

- Substrate with three layers: input, hidden, and output.
- The input and output layers are arranged according to the sensor and motor geometry.
- As a result, HyperNEAT can discover consistent gait patterns.

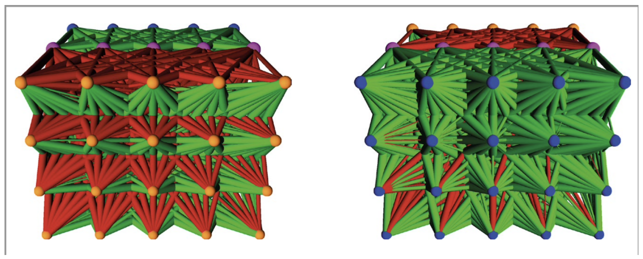




(Clune et al. 2011)

Regularities in Weights and Gaits

- These regularities are reflected in the HyperNEAT weight patterns.
  - Inhibitory and excitatory connections are arranged in geometric patterns based on neuron positions (front and back view; input yellow, output blue).



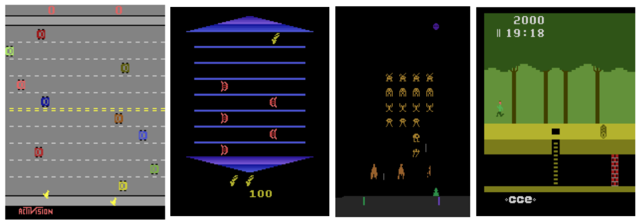
(Clune et al. 2011)

- This regularity with variation results in smooth, coordinated gaits.
- Gaits include synchronized leg movement (pace) and variations like gallop.

(Clune et al. 2011)

Scaling up to Large Networks

- HyperNEAT can efficiently encode large networks with millions of connections using compact CPPNs.
  - Sample the substrate in finer resolution!
- This approach was first used to train neural networks to play Atari games from pixels (before Deep RL).
- Methods for systematic such scaleup are ongoing work.

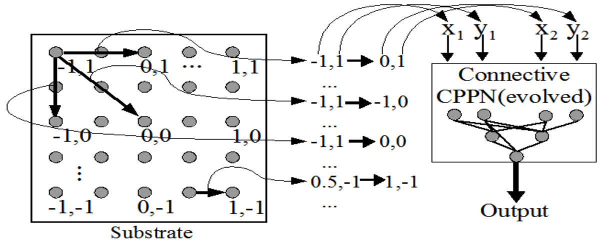


(a) Freeway (b) Asterix (c) Space Invaders (d) Pitfall

(Hausknecht et al. 2014)

Evolving the Substrate

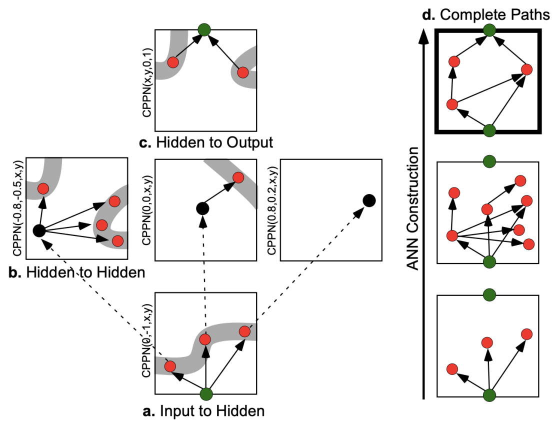
- Well-designed substrates are important;
  Maybe we could optimize them for the task?
- Evolvable substrates HyperNEAT:
  Discover the number and locations of hidden nodes automatically.



(Stanley et al. 2009)

ES-HyperNEAT

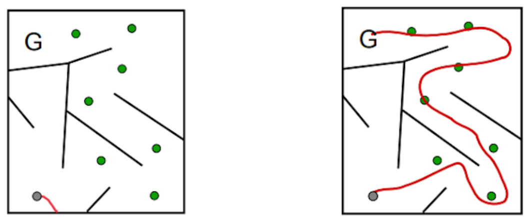
- Place nodes in areas where CPPN generates high variance in weights:
- There is more information in those areas; more representation is needed.
  - Start from input and identify good hidden neuron locations.
  - Then good locations for the next layer of hidden neurons.
  - Then good hidden locations to connect to the output.
  - Construct the network and prune.



(Risi & Stanley 2012)

Example: Maze Navigation Task

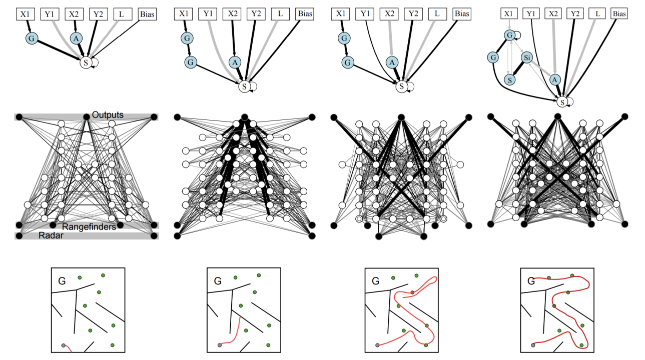
- ES-HyperNEAT was tested on a hard maze navigation task where an agent uses rangefinder sensors.
- The goal is to navigate the maze by learning to reach predefined waypoints (not visible to the agent).



(Risi & Stanley 2012)

Performance in Maze Domain

- ES-HyperNEAT improves from 45% successful runs to 95%.
- ES-HyperNEAT is better at discovering and elaborating on useful stepping stones.
- Over generations, ES-HyperNEAT evolves increasingly complex networks with more hidden nodes and connections.

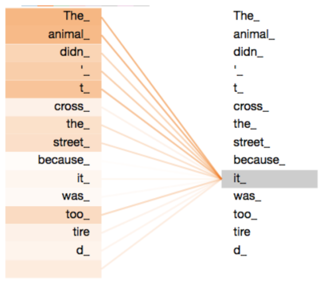


(a) Generation 24, ANN: 30 n, 184 c, CPPN: 2 n, 9 c, fitness = 0.85
(b) Generation 30, ANN: 52 n, 280 c, CPPN: 3 n, 10 c, fitness = 0.93
(c) Generation 106, ANN: 42 n, 310 c, CPPN: 3 n, 10 c, fitness = 5.96
(d) Generation 237, ANN: 40 n, 356 c, CPPN: 5 n, 18 c, fitness = 10.00

(Risi & Stanley 2012)

Self-Attention as Dynamic Indirect Encoding

- So far, the indirect encoding methods have been static during performance.
- Self-attention in transformer networks can be seen as a dynamic form of indirect encoding: It adjusts representations based on input data.
- This flexibility allows models to adapt internal structures and relationships depending on context.



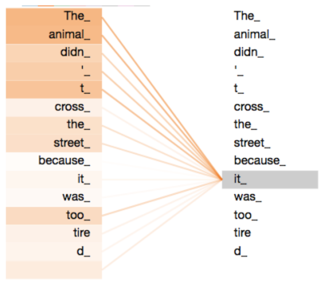
(Alammar 2018)

Self-Attention Implementation

- Recall that the Query and Key matrices (W_{q} and W_{k}) are used to calculate the association matrix A from input X:

$$A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \mathrm{softmax}\left(\frac{(XW_{\mathrm{q}})(XW_{\mathrm{k}})^{\intercal}}{\sqrt{d_k}}\right)$$

- Thus, A identifies the input tokens that are in agreement.

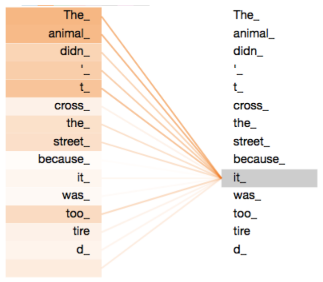


(Alammar 2018)

Input-Dependent Association Architecture

$$A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \mathrm{softmax}\left(\frac{(XW_{\mathrm{q}})(XW_{\mathrm{k}})^{\intercal}}{\sqrt{d_k}}\right)$$

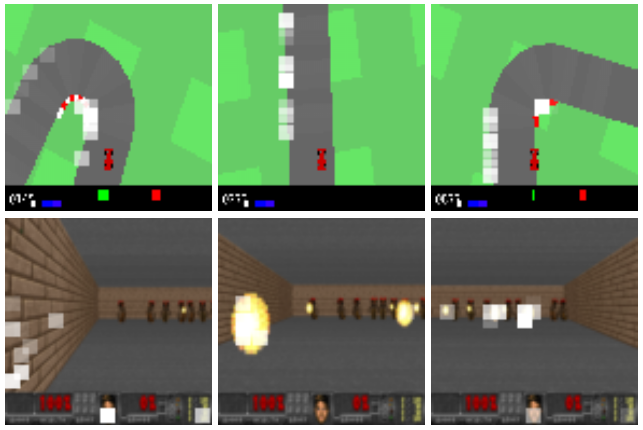
- Now think of W_{q} and W_{k} as the *genotype* and the attention matrix A as the *phenotype*.
  - That is, think of A is part of the network architecture.
- Then A is indirectly encoded by this mapping.
- The mapping is input-dependent, and therefore the indirect encoding is dynamic.



(Alammar 2018)

Self-Attention-Based Agents

- AttentionAgent: Leverages self-attention to focus on relevant elements in task environment.
- It assigns attention only to task-relevant inputs and ignores distractions.
- It improves interpretability in pixel-space reasoning.

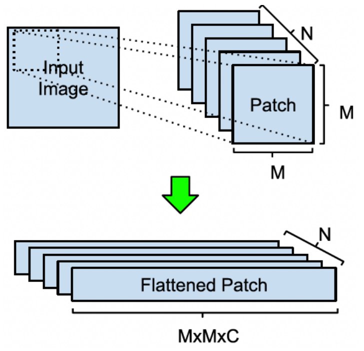


(Tang et al. 2020)

White patches indicate high attention areas

Patch Segmentation in AttentionAgent

- Input game screen is divided into patches (like a convolution layer).
- Each patch is flattened to 1D, forming the input to the self-attention module.

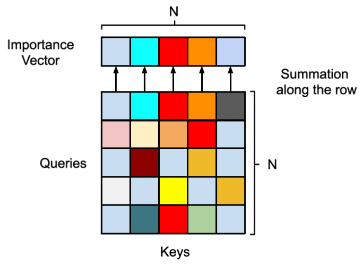


(Tang et al. 2020)

Generating the Attention Matrix

- A simplification: Condense the attention matrix into $X_q X_k^T$
- Such an A represents the importance of each patch relative to others.
  - Each row of A can be interpreted as how a patch distributes its "votes" across other patches.

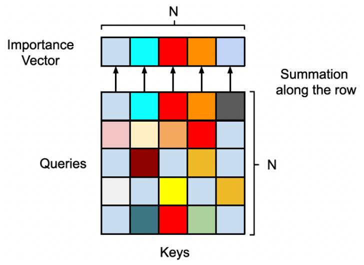
$$A = \text{softmax}\left(\frac{X_q X_k^\intercal}{\sqrt{d}}\right)$$



(Tang et al. 2020)

Patch Importance Vector

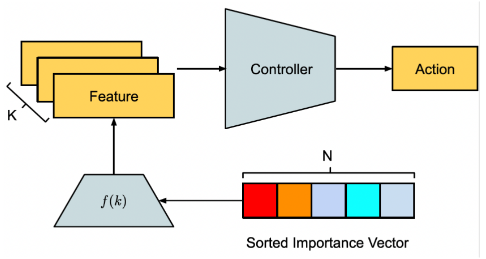
- Summing the columns of the attention matrix A results in a patch importance vector.
- This vector ranks the importance of each patch.
- Only the top k patches are retained for further processing, improving focus on critical elements.



(Tang et al. 2020)

Action Selection

- After obtaining the top-k patches, their features are fed into a neural controller.
- Thus, they are used instead of the Value as output of the attention mechanism.
- The controller processes these features to output the agent's actions.
- Patches of low importance are discarded, allowing the agent to focus solely on relevant elements.

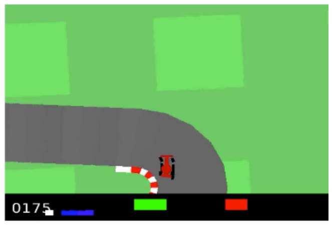


(Tang et al. 2020)

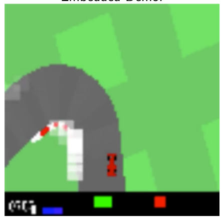
Regular vs. Self-Attention Agent

- Evaluated in CarRacing and DoomTakeCover domains.
- The agent pays attention to crucial parts of input and performs well.
  - Left: Actual game environment presented to humans.
  - Right: Resized images presented to AttentionAgent as input
  - Attention highlighted with white patches.

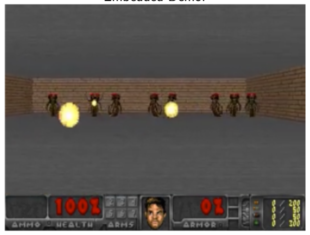
Embedded Demo:



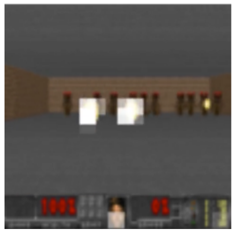
Embedded Demo:



Embedded Demo:



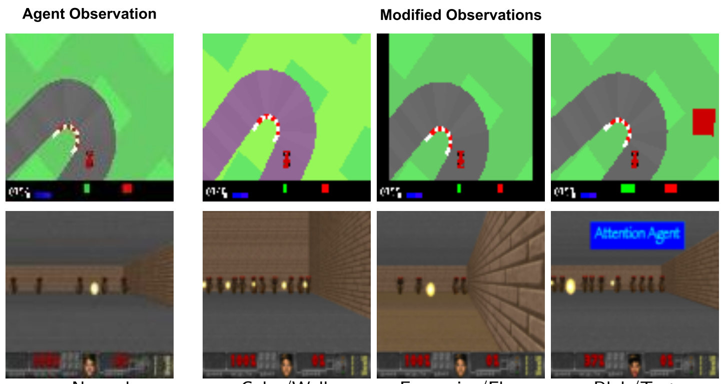
Embedded Demo:



(Tang et al. 2020)

Robustness in Dynamic Environments

- AttentionAgent's focus on key patches allows it to remain robust to external changes.
- Experiments show the agent's ability to ignore distractions such as changing background colors or added text.
- AttentionAgent thus demonstrates the power of self-attention as a dynamic indirect encoding mechanism.



Normal Color/Walls Framesize/Floor Blob/Text

(Tang et al. 2020)

Why Indirect Encodings are Useful

- Solve the problem of scaling neuroevolution by reducing the number of genotypic parameters.
- Allow for the automatic discovery of regularities, such as symmetry, repetition, and modularity.
- Integrate evolutionary learning with individual learning.

Successes of Indirect Encodings

- Synergetic development: better than evolution, better than learning alone.
- Complexity and regularity from grammar-based encodings.
- HyperNEAT: Connectivity patterns with geometric regularities.
- Self-attention agents: Dynamic encoding enables agents to focus on relevant information.

Future Opportunities

- Combining indirect encodings with deep learning for hybrid systems.
- Utilizing biologically-inspired mechanisms like genetic regulatory networks.