Image Source: reddit.com

Neuroevolution for Deep Reinforcement Learning Problems

David Ha

Research Scientist at Google Brain in Tokyo

@hardmaru

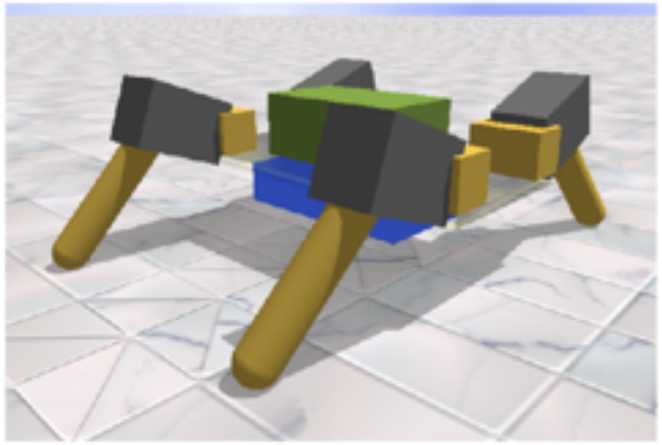
hadavid@google.com

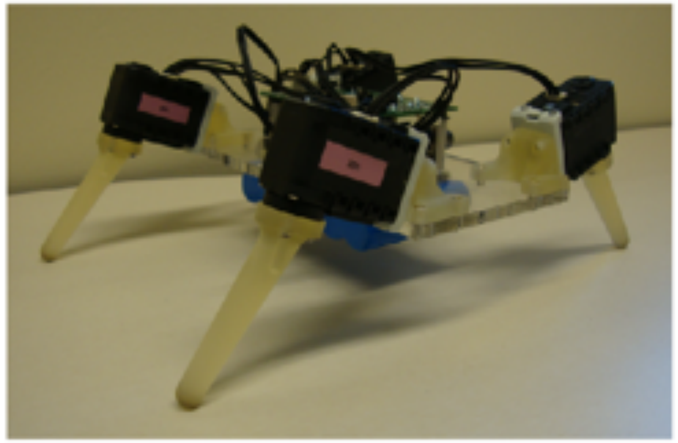
GECCO 2019

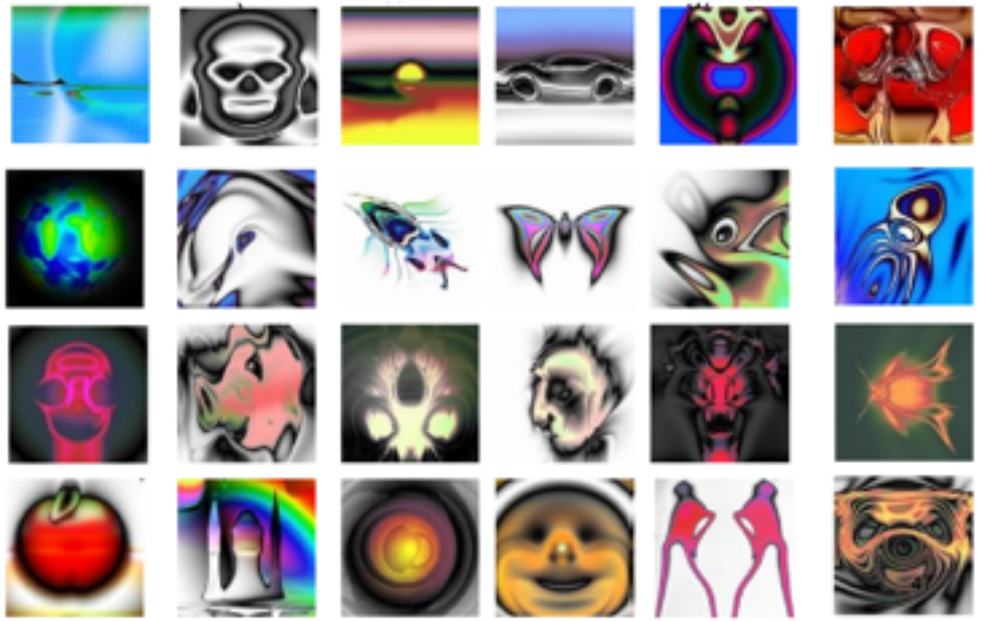
Agenda

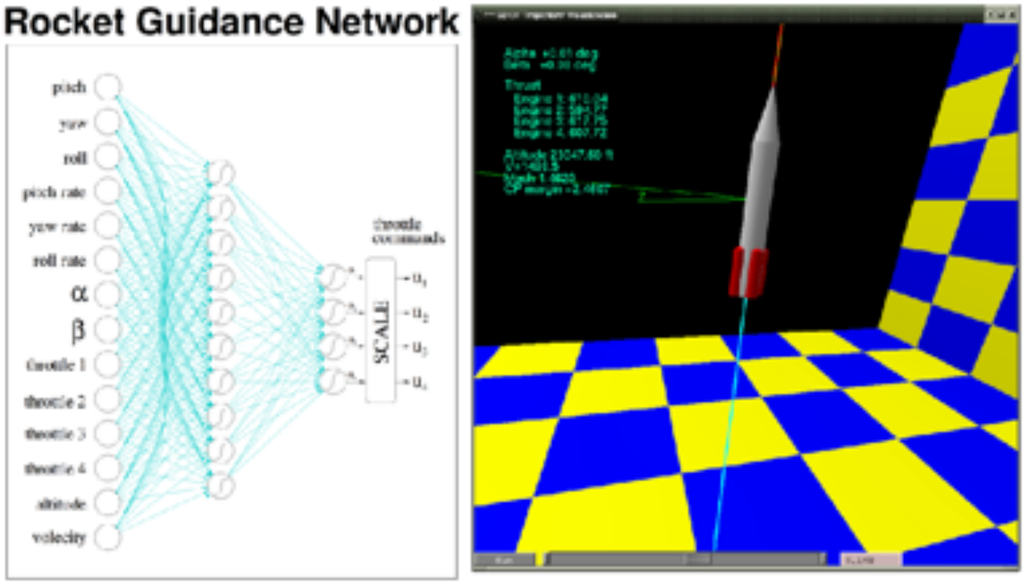
- Brief Review of Neuroevolution
- Review of Reinforcement Learning Environments
- Research inspired by Evolution / Artificial Life
- Combine Generative Models with Evolution
- Revisiting Neural Architecture Search / Morphology Search
- Questions & Discussion

Neuroevolution



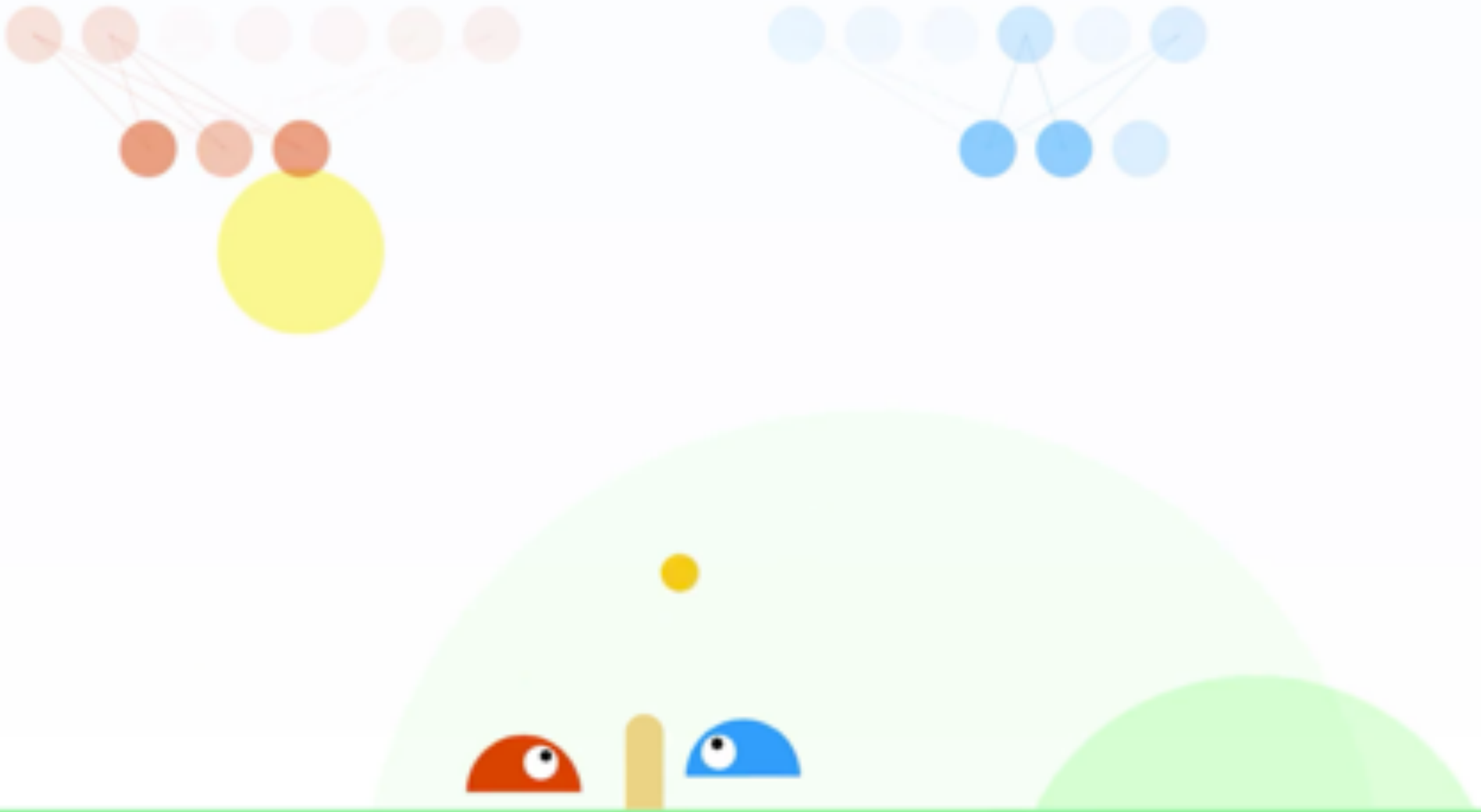


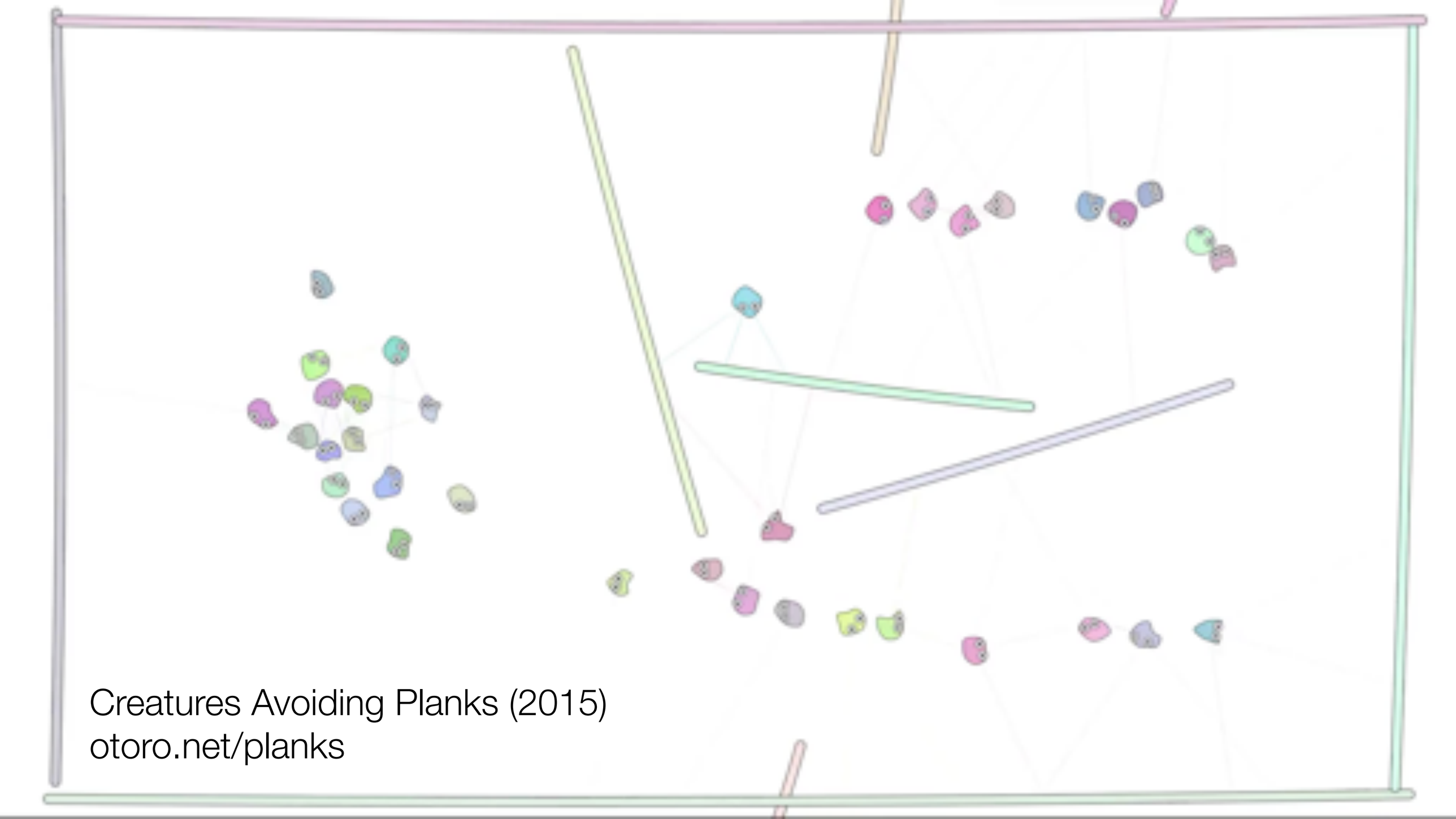




From Risto Miikkulainen's "Evolving Neural Networks" Tutorial (IJCNN 2013, Dallas)

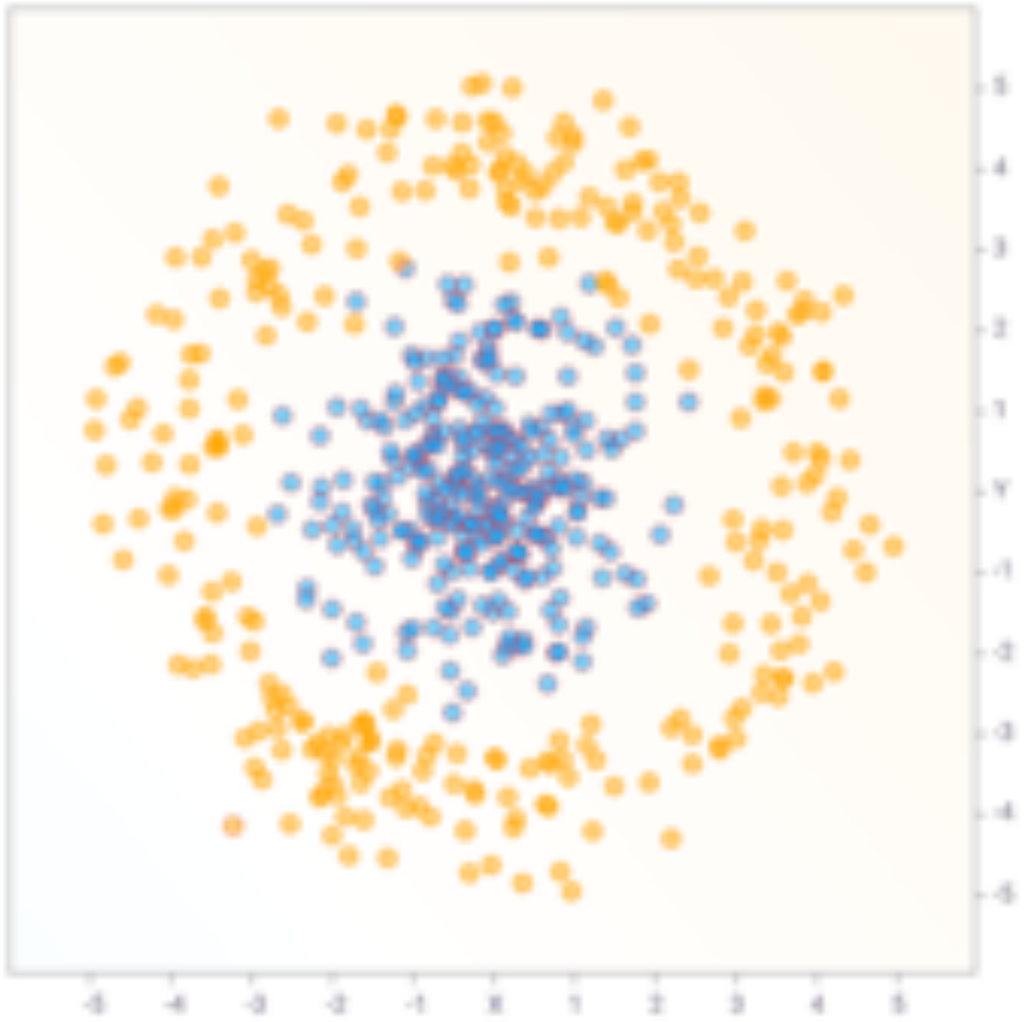
Neural Slime Volleyball (2015)
otoro.net/slimevolley/





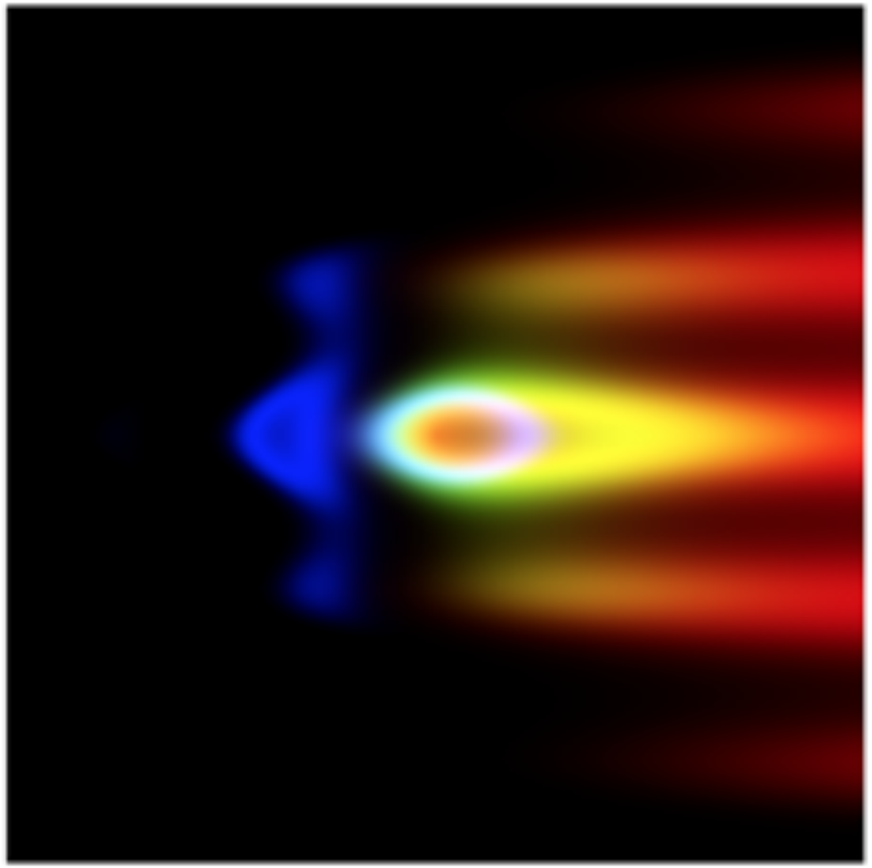
Creatures Avoiding Planks (2015)
otoro.net/planks

Backprop NEAT (2016) blog.otoro.net

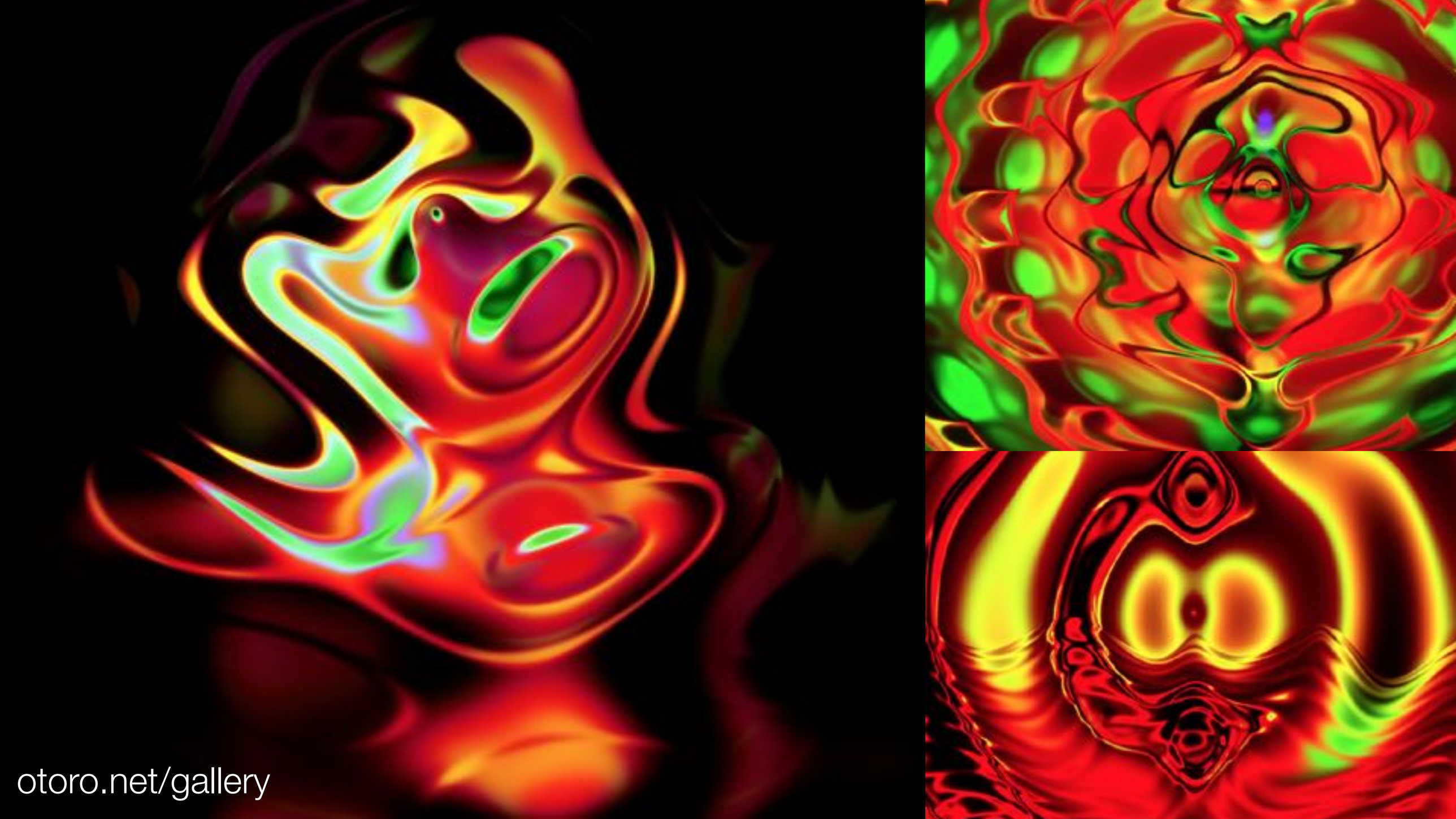


Stanley and Miikkulainen, Evolving Neural Networks Through Augmenting Topologies (NEAT) (Evolutionary Computation, 2002)

Javascript Implementation of CPPN-NEAT (2015)
otoro.net/neurogram (Inspired by picbreeder.com)



Stanley, Compositional Pattern Producing Networks (CPPNs): A novel abstraction of development (2007)
Stanley et al., A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks (HyperNEAT) (Artificial Life, 2009)


otoro.net/gallery

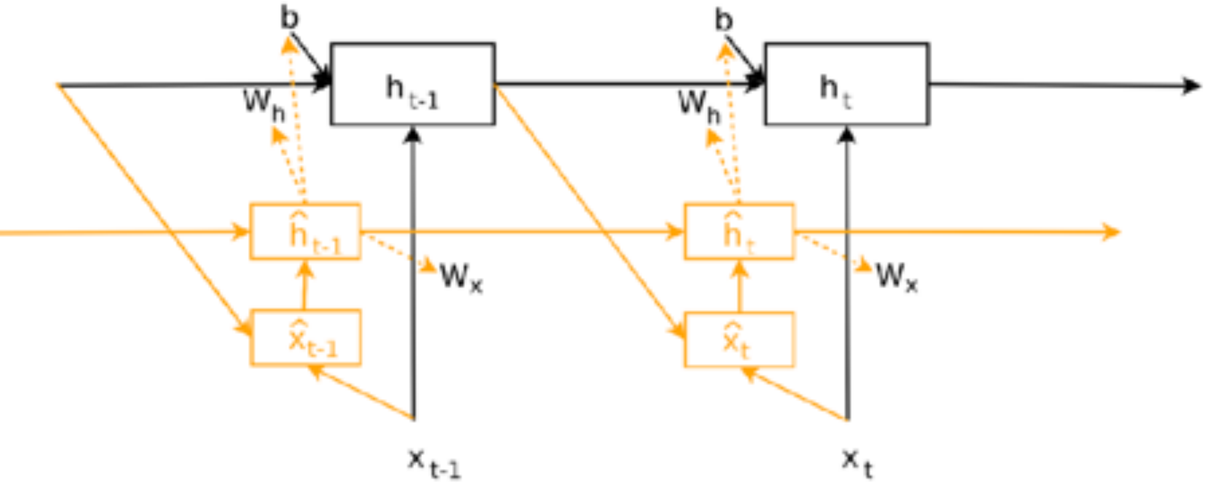
MNIST Classification

28x28 pixel images of handwritten digits

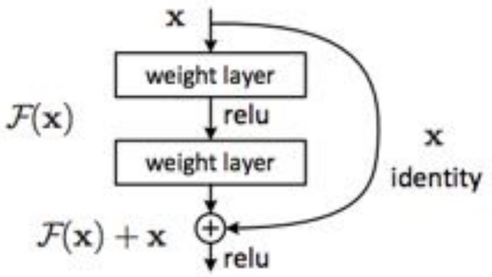
Generating Large Images from Latent Vectors

blog.otoro.net

From HyperNEAT to HyperNetworks to Neural ODEs

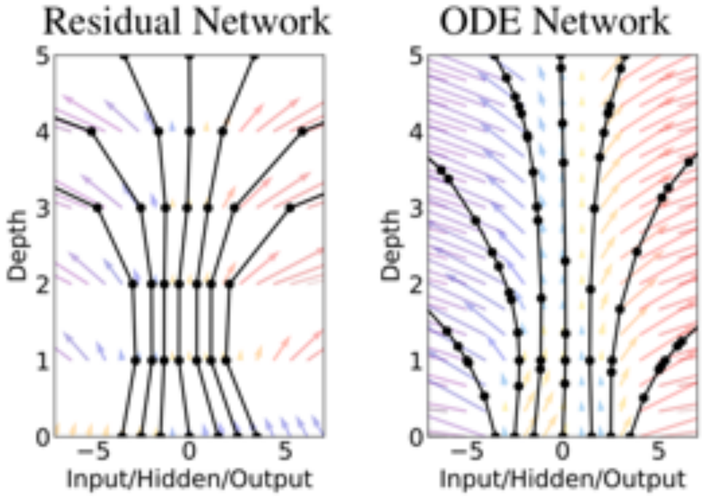


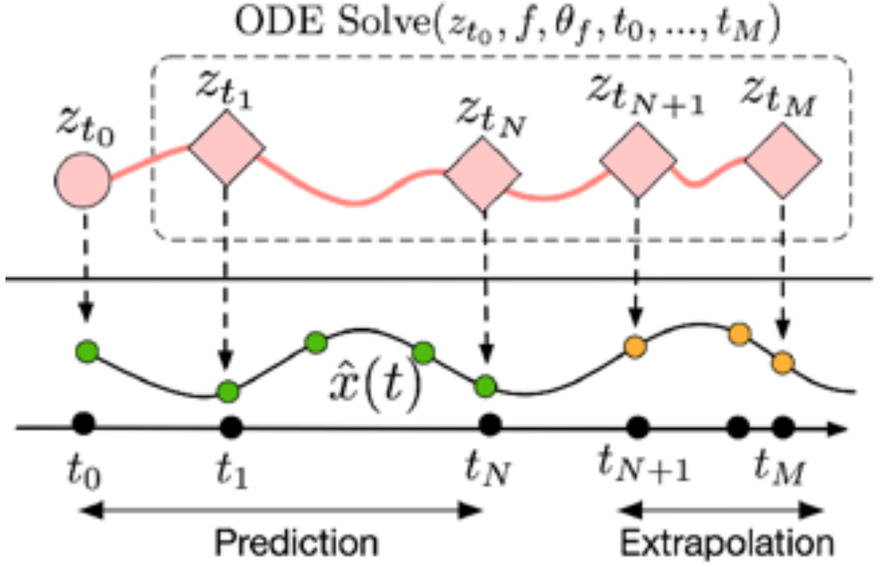
Ha et al., HyperNetworks (ICLR 2017)



$$h_{t+1} = h_t + f(h_t)$$
$$= h_t + \frac{\Delta t}{\Delta t} f(h_t)$$
$$= h_t + \Delta t\, G(h_t)$$

A single step of Euler's method for solving ODE

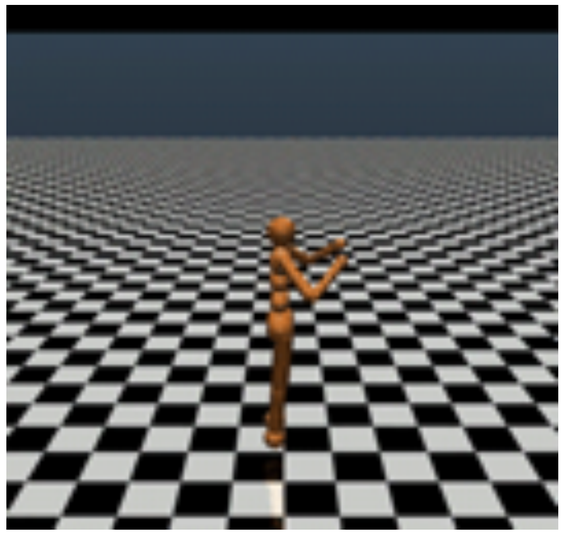


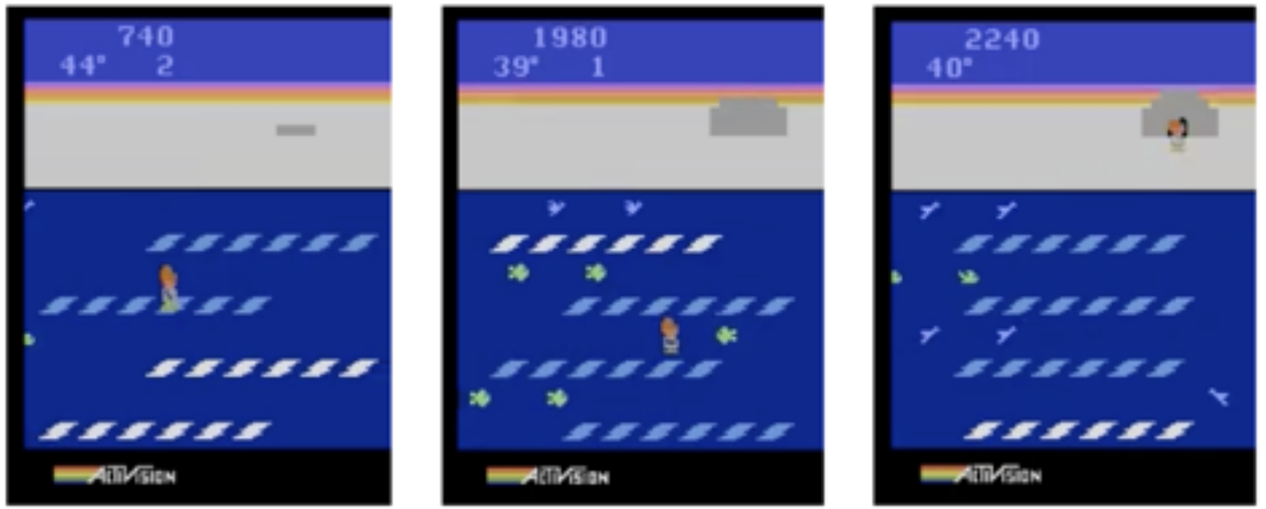


Chen et al., Neural Ordinary Differential Equations (NeurIPS 2018)

Influential Work on Evolution in RL

- Evolution Strategies as a Scalable Alternative to Reinforcement Learning
  Salimans et al. (OpenAI) https://arxiv.org/abs/1703.03864
- Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning
  Such et al. (UberAI) https://arxiv.org/abs/1712.06567
- Simple Random Search Provides a Competitive Approach to Reinforcement Learning
  Mania et al. (Berkeley) (NeurIPS 2018) https://arxiv.org/abs/1803.07055





Sample Efficiency, Wallclock Performance

Environment	Ratio
HalfCheetah	0.58
Hopper	6.94
InvertedDoublePendulum	1.23
InvertedPendulum	0.88
Swimmer	0.30
Walker2d	7.88

MuJoCo tasks: Ratio of ES timesteps to TRPO timesteps needed to reach TRPO's performance at 5M timesteps

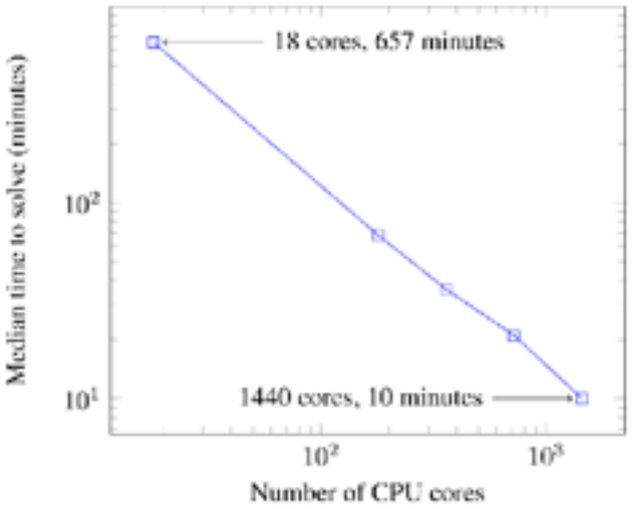


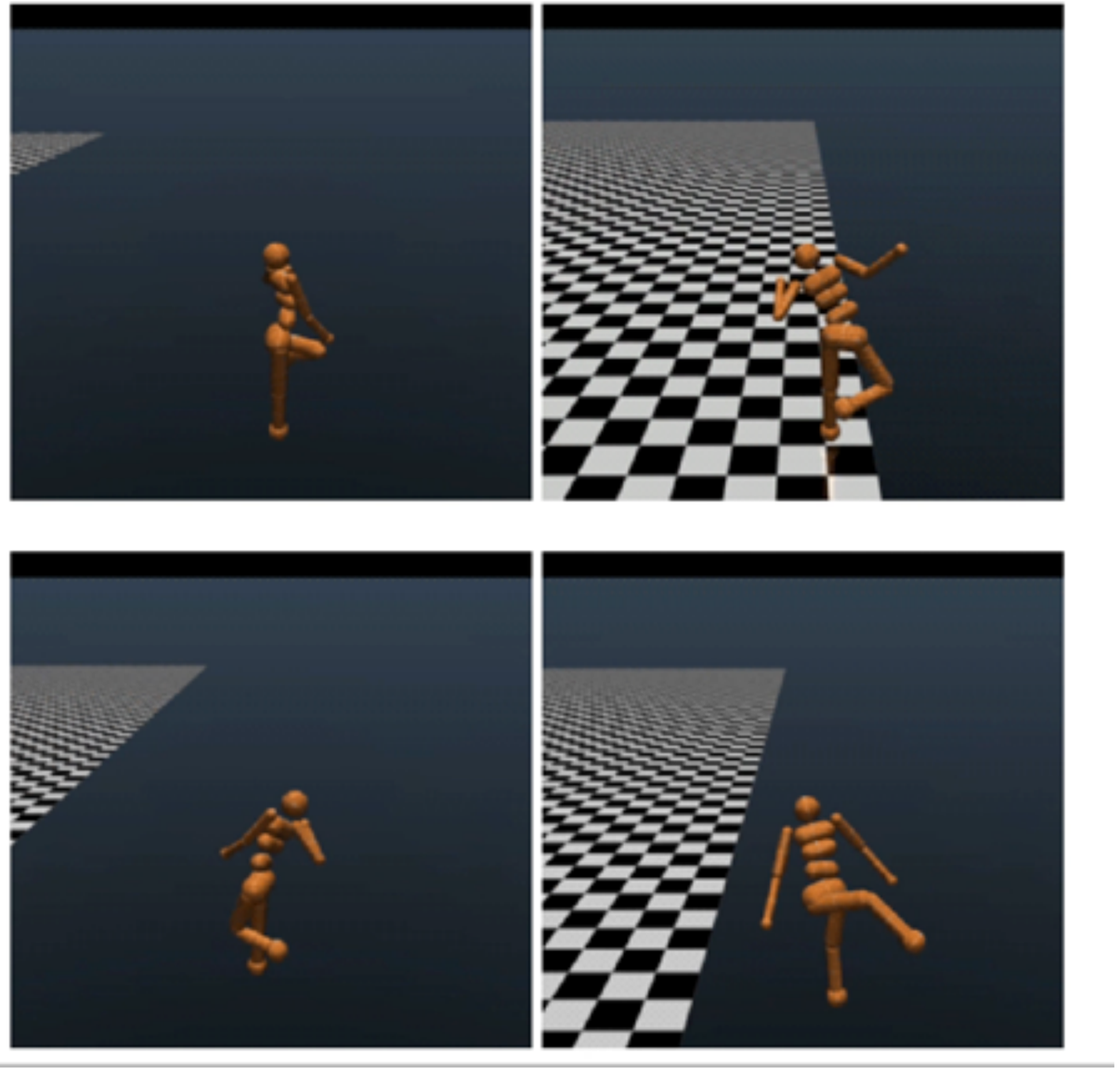
Figure 1: Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

	DQN	ES	A3C	GA
Frames	200M	1B	1B	6B
Time	~7-10d	~ 1h	~ 4d	~ 6h or 24h
Forward Passes	450M	250M	250M	1.5B
Backward Passes	400M	0	250M	0
Operations	1.25B U	250M U	1B U	1.5B U
amidar	**978**	112	264	377
assault	4,280	1,674	**5,475**	814
asterix	4,359	1,440	**22,140**	2,255
asteroids	1,365	1,562	**4,475**	2,700
atlantis	279,987	**1,267,410**	911,091	129,167
enduro	**729**	95	-82	80
frostbite	797	370	191	**6,220**
gravitar	473	**805**	304	764
kangaroo	7,259	**11,200**	94	**11,254**
seaquest	**5,861**	1,390	2,355	850
skiing	-13,062	-15,443	-10,911	†**-5,541**
venture	163	760	23	†**1,422**
zaxxon	5,363	6,380	**24,622**	7,864

Salimans et al., Evolution strategies as a scalable alternative to reinforcement learning (2007)

Such et al., Deep neuroevolution: GAs are a competitive alternative for training deep neural networks for RL (2007)

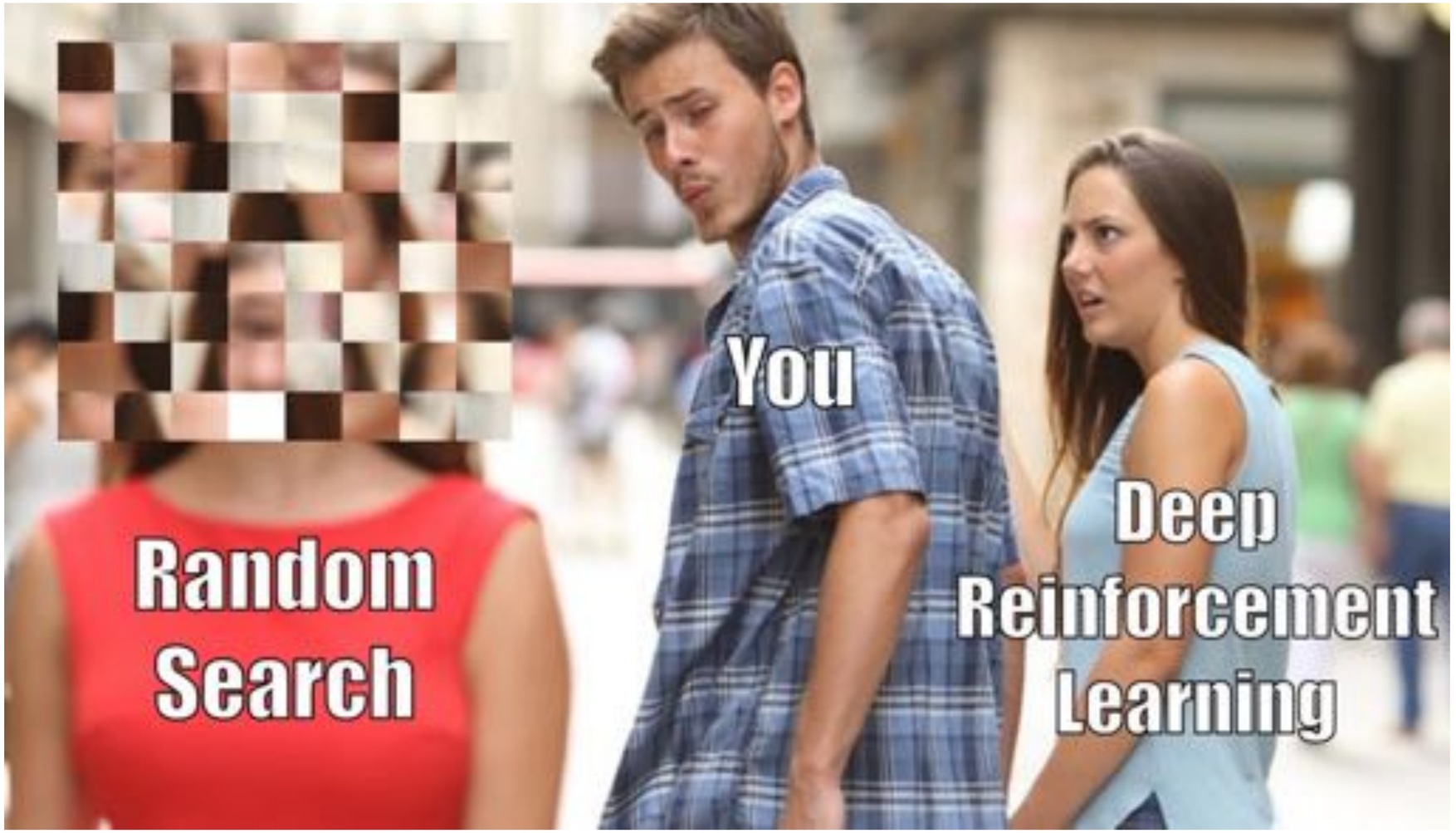
You may not like it, but this is what peak performance looks like.



Mania et al., Simple random search provides a competitive approach to reinforcement learning (NeurIPS 2018)

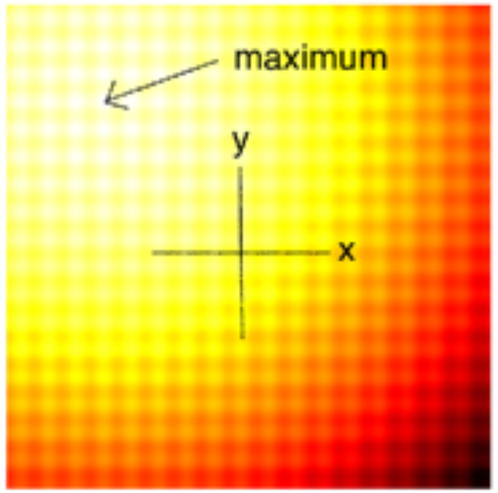
“Hype Cycle”

Tired — Deep RL
Wired — Evolution Strategies
Inspired — Random Search

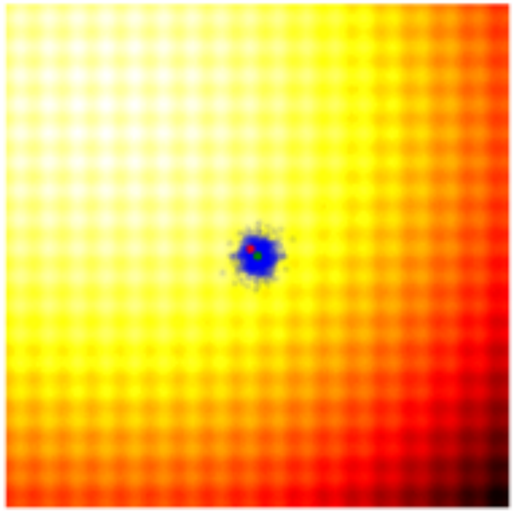


Neuroevolution Algorithms

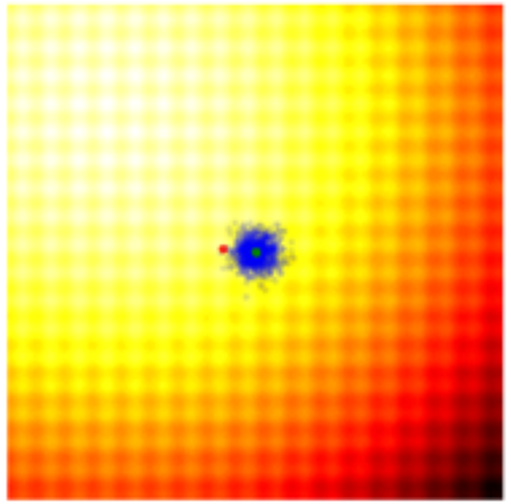
- Many out there: ES (CMA-ES, OpenAI-ES, PEPG, ARS), GA



Optimisation Problem:
Shifted 2D Rastrigin



Simple Evolution Strategies
i.e. (1 + 1)-ES

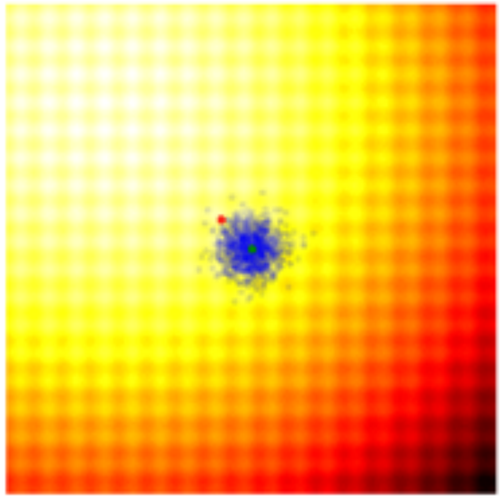


Simple Genetic Algorithm

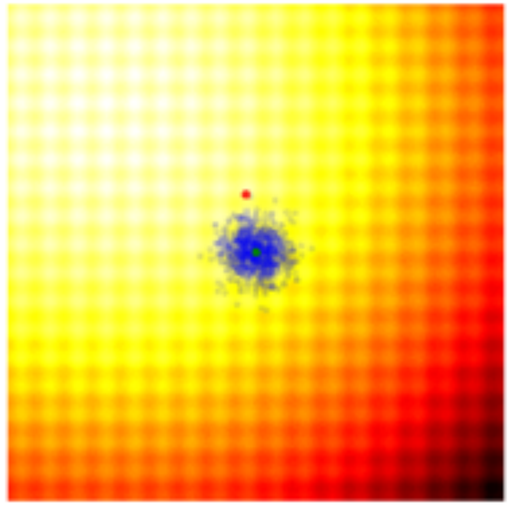
- Benchmark Tasks with Many Local Optima

A Visual Guide to Evolution Strategies (2017) blog.otoro.net

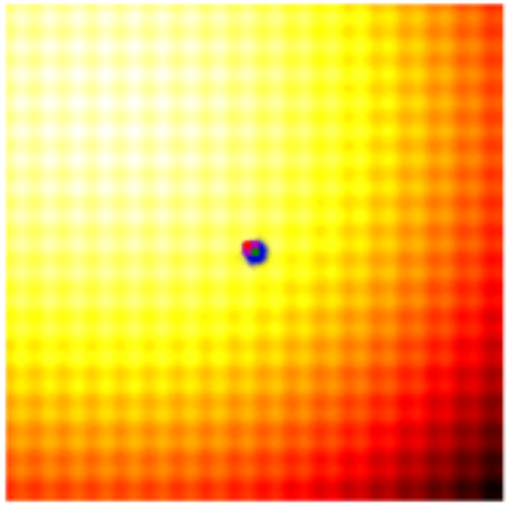
Neuroevolution Algorithms



OpenAI ES
(update mu,
but constant sigma)



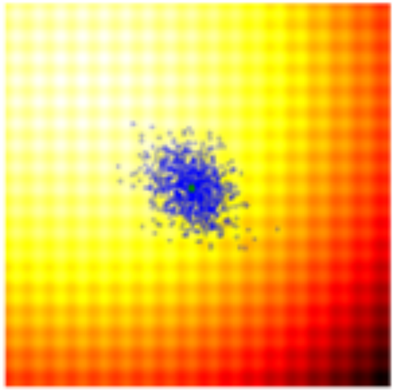
Parameter-Exploring
Policy Gradients
(update mu,
update sigma,
but zero correlation)



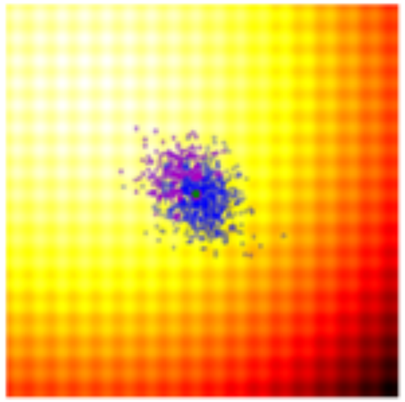
CMA-ES
(update mu,
update sigma,
update correlation)

- We want to use a standard interface for all these methods.
- Standardize Gym Environments to use with Neuroevolution.

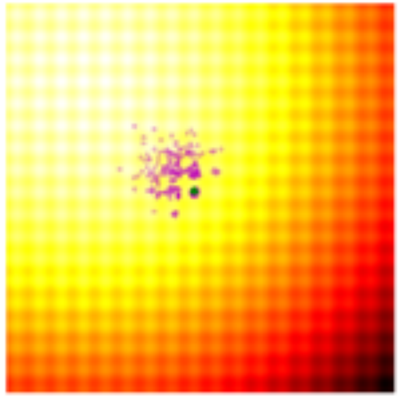
CMA-ES for Dummies



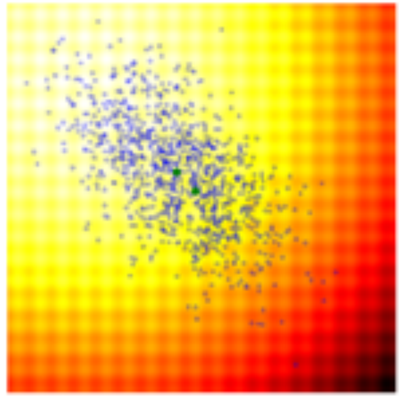
Step 1



Step 2

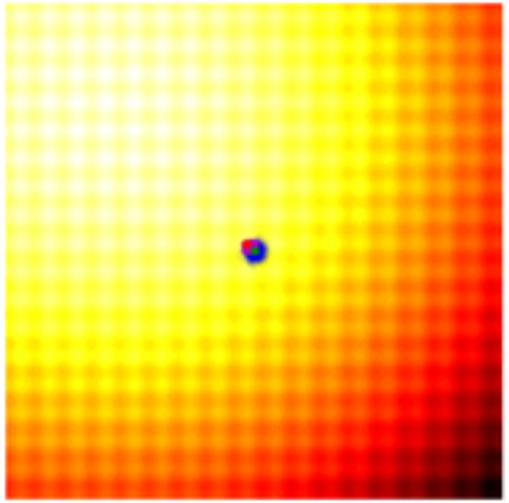


Step 3



Step 4

1. Calculate the fitness score of each candidate solution in generation.
2. Isolates the best 25% of the population in generation, in purple.
3. Using only the best solutions, and using the mean of the *current* generation (the green dot), calculate the covariance matrix of the next generation.
4. Sample a new set of candidate solutions using the updated mean and covariance matrix.



Check out “CMA-ES” Tutorial by Hansen tomorrow at GECCO!

Population-based REINFORCE (OpenAI-ES, PEPG)

Imagine if you had built an artificial life simulator, and you sample a different neural network to control the behavior of each ant inside an ant colony.

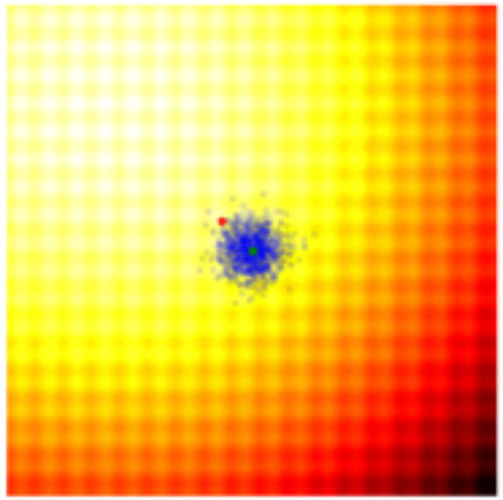
Using an elitist evolution algorithm for this task will optimize for traits and behaviors that benefit individual ants, and with each successive generation, our population will be full of alpha ants who only care about their own well-being.

Instead of using a rule that is based on the survival of the fittest ants, what if you take an alternative approach where you take the sum of all fitness values of the entire ant population, and optimize for this sum instead to maximize the well-being of the entire ant population over successive generations?

Well, you would end up creating a Marxist utopia.

Population-based REINFORCE (OpenAI-ES, PEPG)

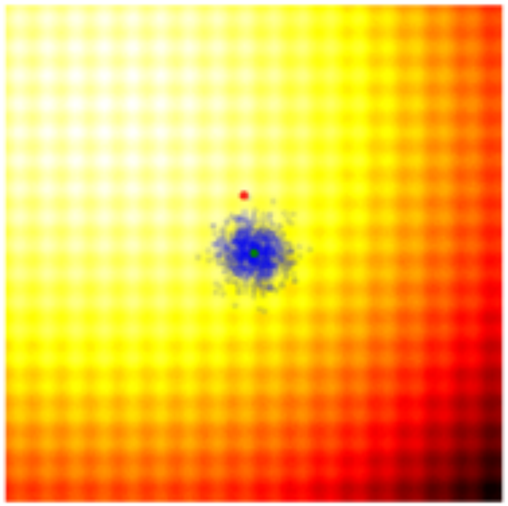
$\pi(z, \theta)$



OpenAI ES

θ: μ only (σ is fixed)

$\pi(z, \theta)$



PEPG

θ: μ and σ

$z_j \sim N(\mu_j, \sigma_j)$ z is a 2D vector (x, y)

$$\nabla_{\mu_j} \log N(z^i, \mu, \sigma) = \frac{z_j^i - \mu_j}{\sigma_j^2},$$

$$\nabla_{\sigma_j} \log N(z^i, \mu, \sigma) = \frac{(z_j^i - \mu_j)^2 - \sigma_j^2}{\sigma_j^3}.$$

$$J(\theta) = E_\theta[F(z)] = \int F(z)\, \pi(z, \theta)\, dz$$

By using "log-likehood trick" gradient of expectation becomes:

$$\nabla_\theta J(\theta) = E_\theta[\, F(z)\, \nabla_\theta \log \pi(z, \theta)\,]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} F(z^i)\, \nabla_\theta \log \pi(z^i, \theta)$$

Once gradient is estimated, we can update our population:

$$\theta \to \theta + \alpha \nabla_\theta J(\theta)$$

Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning (Machine learning, 1992)

REINFORCE (Williams, 1992) is also an ES!

6 REINFORCE With Multiparameter Distributions

An interesting application of the REINFORCE framework is to the development of learning algorithms for units that determine their scalar output stochastically from multiparameter distributions rather than the single-parameter distributions used by stochastic semilinear units.

A REINFORCE algorithm for this unit thus has the form

$$\Delta\mu = \alpha_\mu (r - b_\mu)\frac{y - \mu}{\sigma^2} \tag{13}$$

and

$$\Delta\sigma = \alpha_\sigma (r - b_\sigma)\frac{(y - \mu)^2 - \sigma^2}{\sigma^3}, \tag{14}$$

where α_μ, b_μ, α_σ, and b_σ are chosen appropriately. A reasonable algorithm is obtained by setting

$$\alpha_\mu = \alpha_\sigma = \alpha\sigma^2,$$

where α is a suitably small positive constant,[2] and letting $b_\mu = b_\sigma$ be determined according to a reinforcement comparison scheme.

REINFORCE (Williams, 1992) and Other ES methods

REINFORCE (Section 6 describes Population-based REINFORCE)

Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning (Machine learning, 1992)

PEPG (Population-based REINFORCE with nice tricks to make it work better)

Sehnke et al., Parameter-exploring policy gradients, (Neural Networks, 2010)

NES (Population-based REINFORCE with Fisher Information Matrix and relationship to natural gradients)

Wierstra et al., Natural evolution strategies, (CES 2008) and (Journal of Machine Learning Research, 2014) 🤔

OpenAI-ES (A simpler version of population-based REINFORCE with a constant variance for all parameters, easy to scale up)

Salimans et al., Evolution strategies as a scalable alternative to reinforcement learning (arxiv.org, 2007)

CMA-ES

Hansen and Ostermeier, Completely derandomized self-adaptation in evolution strategies, (Evolutionary computation, 2001)

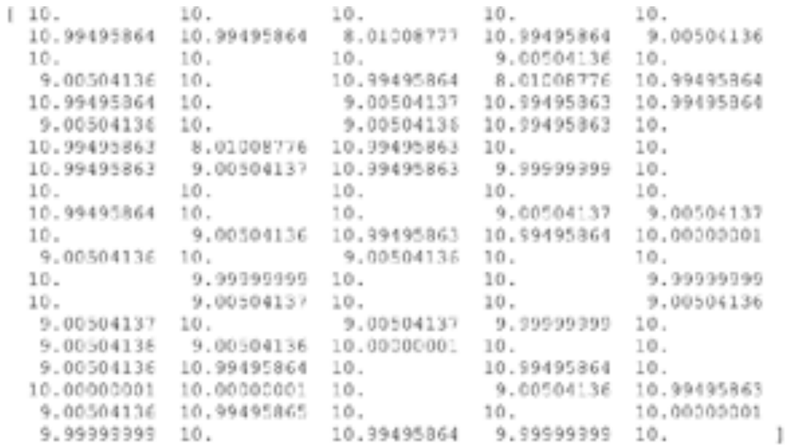
Relationship between CMA-ES and NES

Akimoto et al., Bidirectional relation between CMA evolution strategies and natural evolution strategies,
(International Conference on Parallel Problem Solving from Nature, 2010)

Neuroevolution with Python

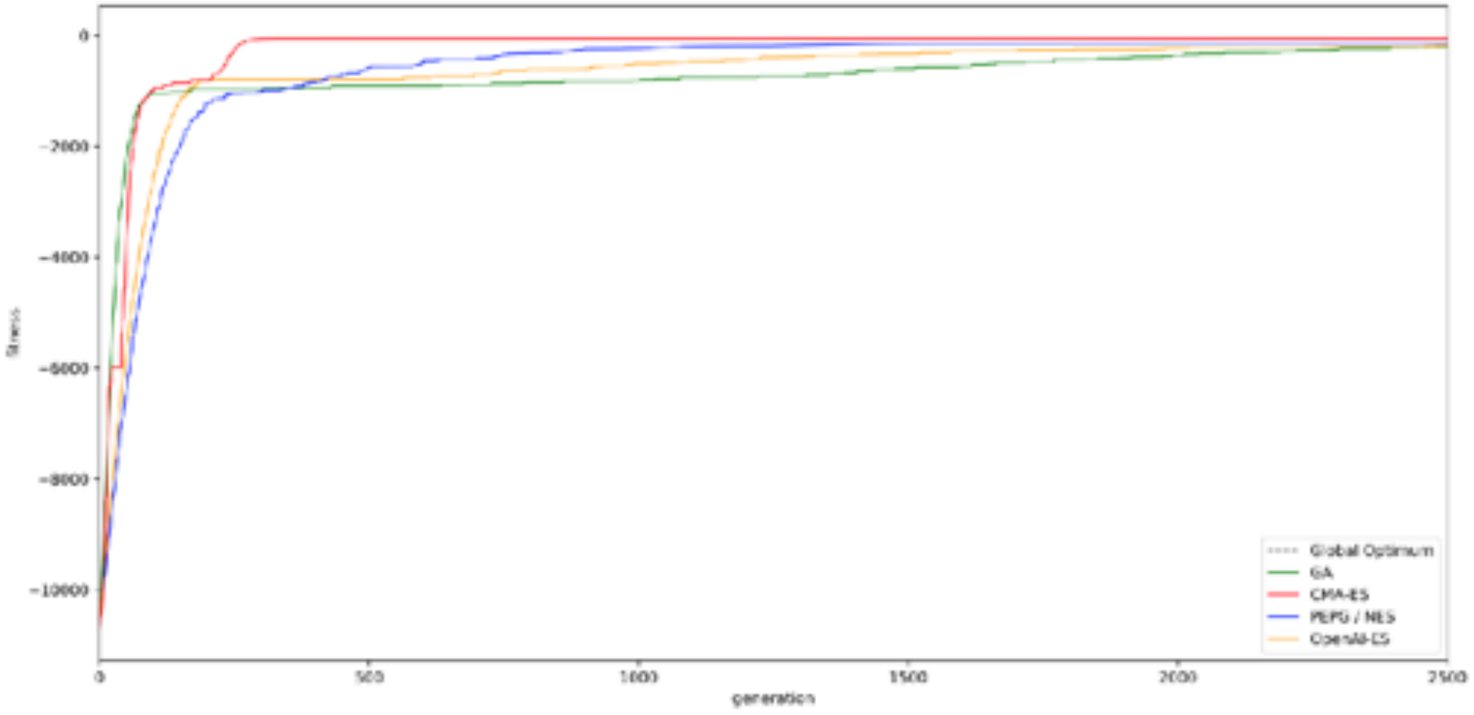

```
solver = OurNeuroevolutionAlgo()

POPULATION_SIZE = 100

# solver = ES(POPULATION_SIZE)
# solver = GA(POPULATION_SIZE)
# solver = CMAES(POPULATION_SIZE)

while True:

  solutions = solver.ask()
  fitness_list = np.zeros(POPULATION_SIZE)

  for i in range(POPULATION_SIZE):
    fitness_list[i] = evaluate(solutions[i])

  solver.tell(fitness_list)
  bestsol, bestfit = solver.result()

  if bestfit > MY_REQUIREMENT:
    break
```

100-D Rastrigin Function



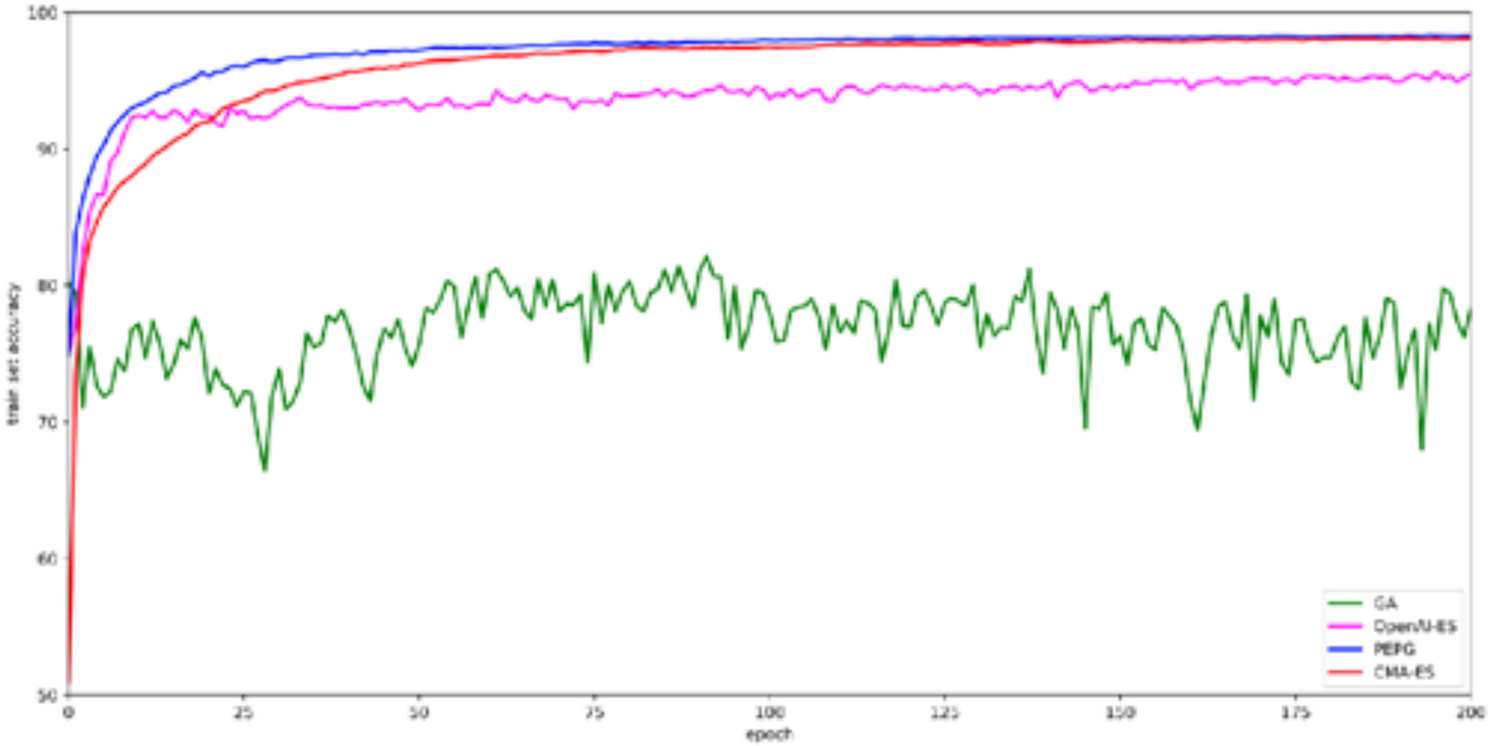
```
[ 10.          10.          10.          10.          10.
  10.99495864  10.99495864   8.01008777  10.99495864   9.00504136
  10.          10.          10.           9.00504136  10.
   9.00504136  10.          10.99495864   8.01008776  10.99495864
  10.99495864  10.           9.00504137  10.99495863  10.99495864
   9.00504136  10.           9.00504136  10.99495863  10.
  10.99495863   8.01008776  10.99495863  10.          10.
  10.99495863   9.00504137  10.99495863   9.99999999  10.
  10.          10.          10.          10.          10.
  10.99495864  10.          10.           9.00504137   9.00504137
  10.           9.00504136  10.99495863  10.99495864  10.00000001
   9.00504136  10.           9.00504136  10.          10.
  10.           9.99999999  10.          10.           9.99999999
  10.           9.00504137  10.          10.           9.00504136
   9.00504137  10.           9.00504137   9.99999999  10.
   9.00504136   9.00504136  10.00000001  10.          10.
   9.00504136  10.99495864  10.          10.99495864  10.
  10.00000001  10.00000001  10.           9.00504136  10.99495865
   9.00504136  10.99495865  10.          10.          10.00000001
   9.99999999  10.          10.99495864   9.99999999  10.        ]
```

- Final solution that CMA-ES discovered for 100-D Rastrigin function.
- Global optimal solution is a 100-dimensional vector of exactly 10.



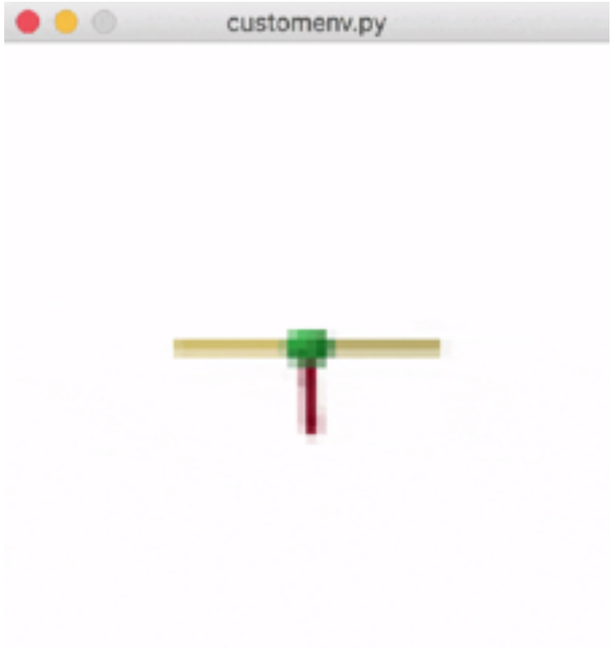
MNIST the "Hard Way"

Method	Train Set	Test Set
Adam (SGD)	99.8	98.9
Simple GA	82.1	82.4
CMA-ES	98.4	98.1
OpenAI-ES	96.0	96.2
PEPG	98.5	98.0



OpenAI Gym Environments


```
import gym
env = gym.make(“Pendulum-v1”)
observation = env.reset()
for _ in range(1000):
  env.render()
  # your agent here (just random actions)
  action = env.action_space.sample()
  obs, reward, done, info = env.step(action)
```



- Standard interface for single-agent RL environment.
- Interface for partially observable Markov decision process (POMDP)
- The simple API has been adopted by RL community.

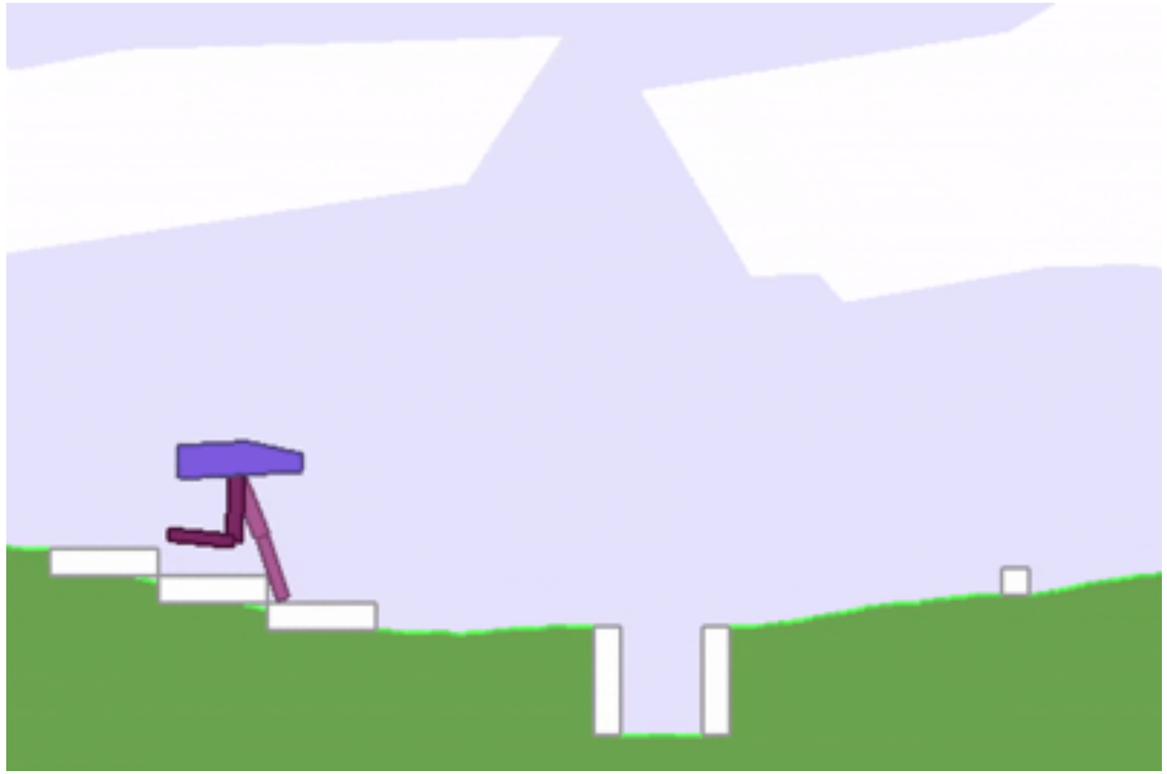
Neuroevolution with OpenAI Gym


```
env = gym.make('Our_Environment')

solver = OurNeuroevolutionAlgo()

while True:

  solutions = solver.ask()
  fitlist = np.zeros(POPULATION_SIZE)

  for i in range(POPULATION_SIZE):
    agent = Agent(solutions[i])
    fitlist[i] = rollout(agent, env)

  solver.tell(fitlist)
  bestsol, bestfit = solver.result()

  if bestfit > MY_REQUIREMENT:
    break
```

```
def rollout(agent, env):
  obs = env.reset()
  done = False
  total_reward = 0
  while not done:
    a = agent.get_action(obs)
    obs, reward, done = env.step(a)
    total_reward += reward
  return total_reward
```

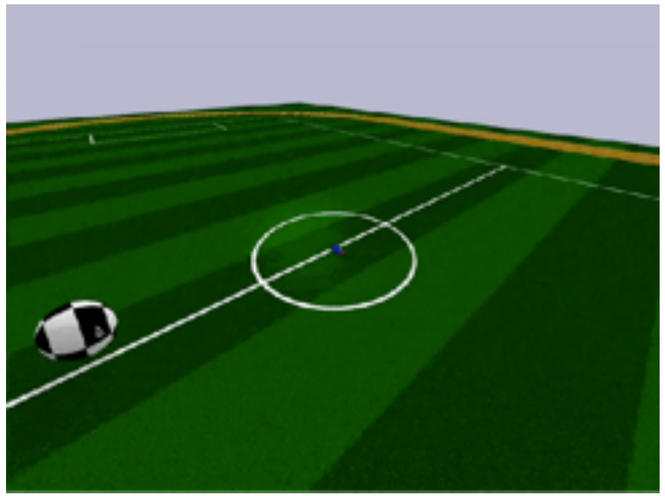
- We only care about the terminal, cumulative reward.
- Simple implementation at http://github.com/hardmaru/estool

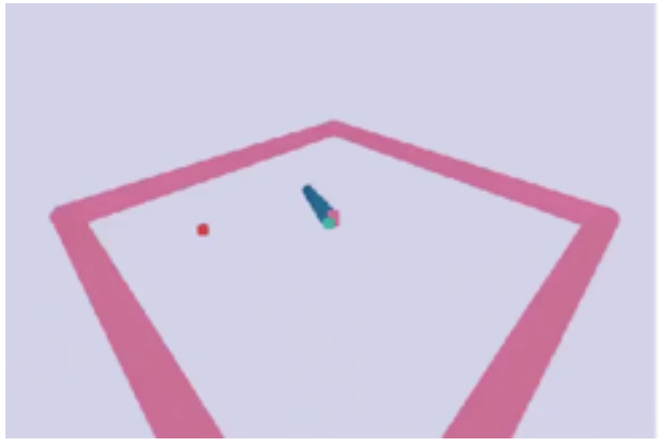
Why Evolution for RL problems?

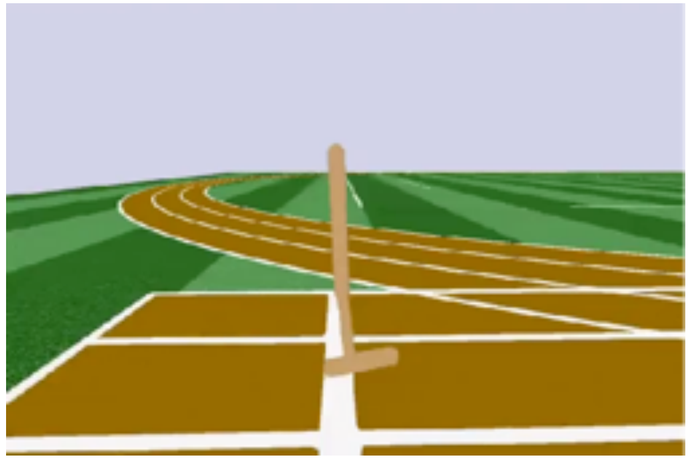


- Credit assignment, especially long term rewards, is hard!
- Easy to get stuck in local optima.
- RL works really well for dense reward signals, but not for sparse or cumulative reward.

ESTool with PyBullet, Roboschool

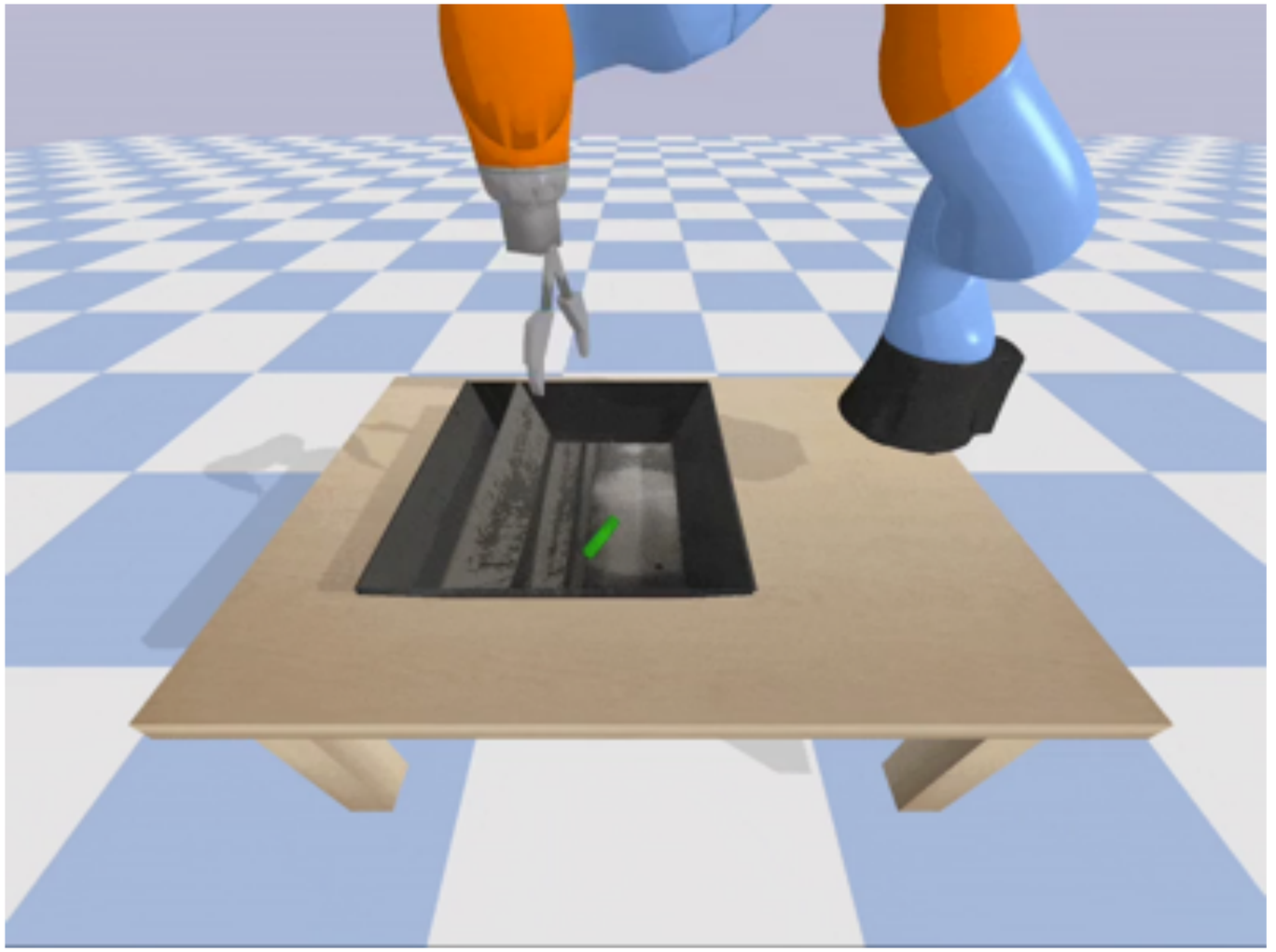






- We can solve a large set of standard continuous control tasks.

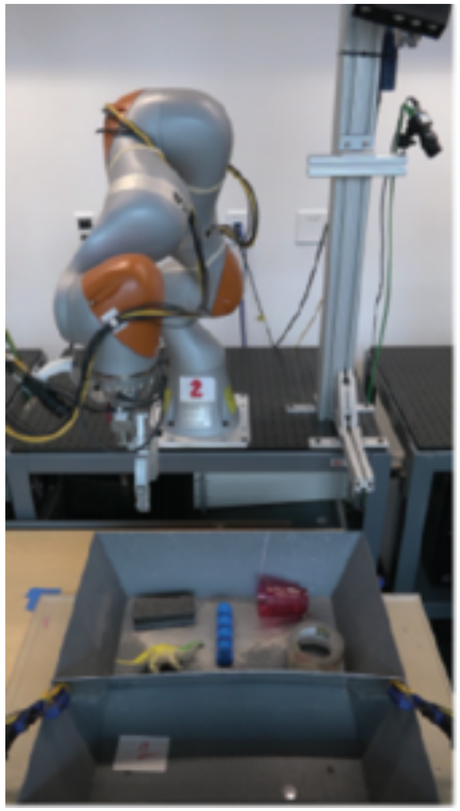
ESTool with PyBullet Kuka Arm

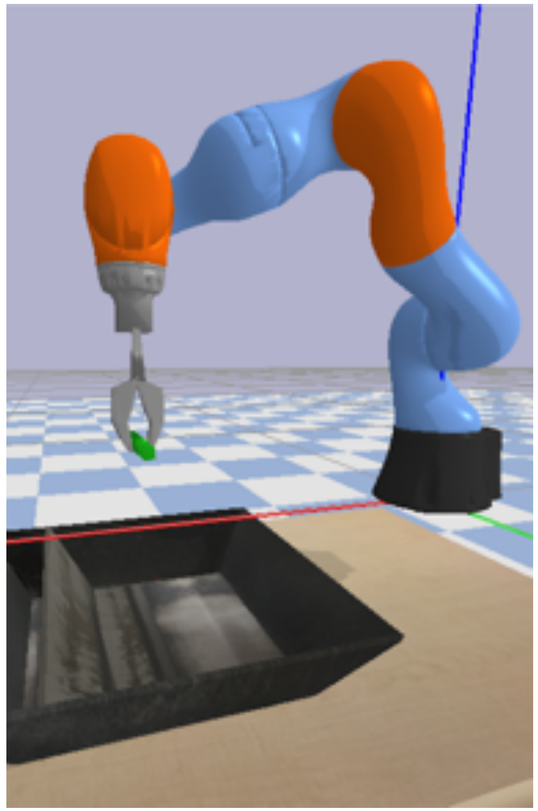


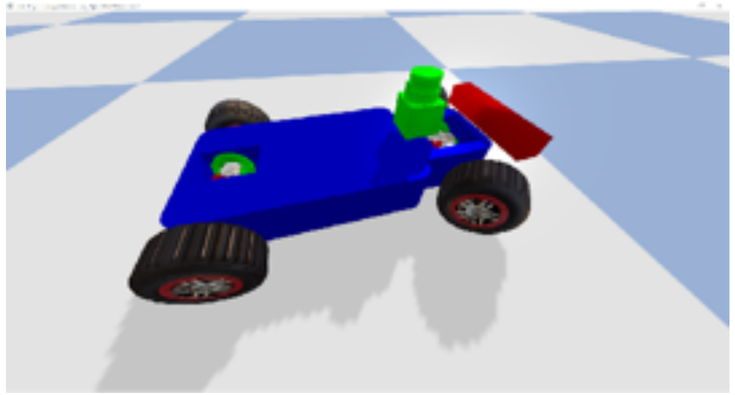
- Kuka grasping tasks easily solvable. Can incorporate vision.

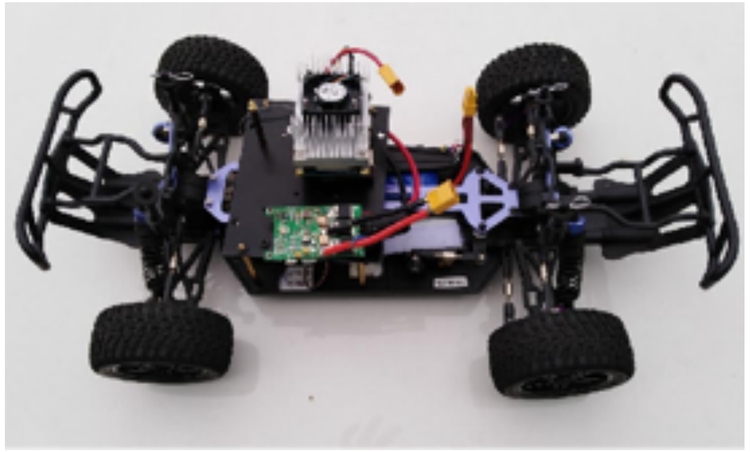
Transfer Learning PyBullet Models

- MIT Racecar, Minitaur, Kuka Arm
- Erwin Coumans, Brain Robotics



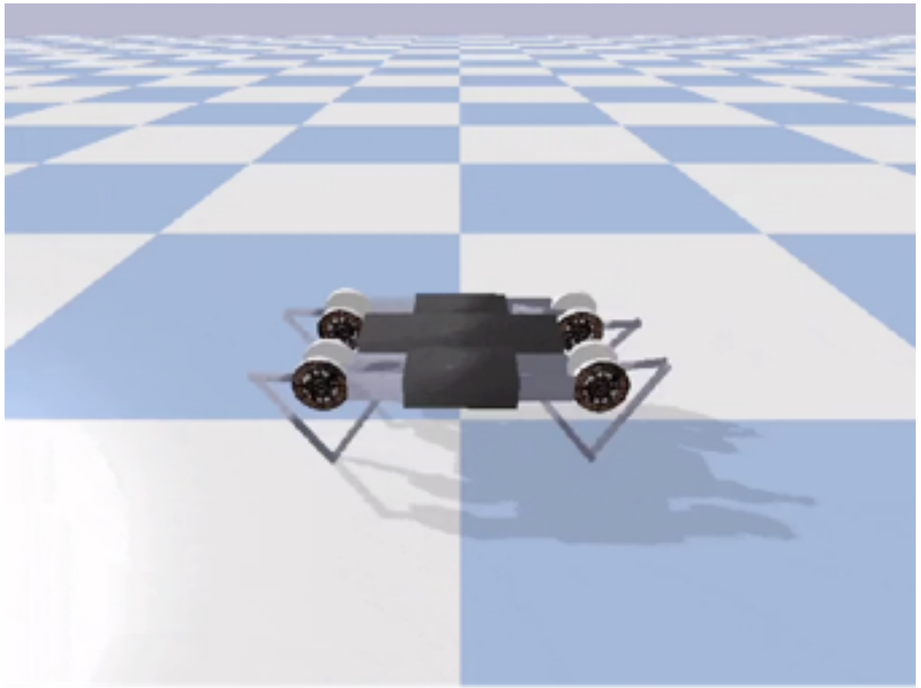






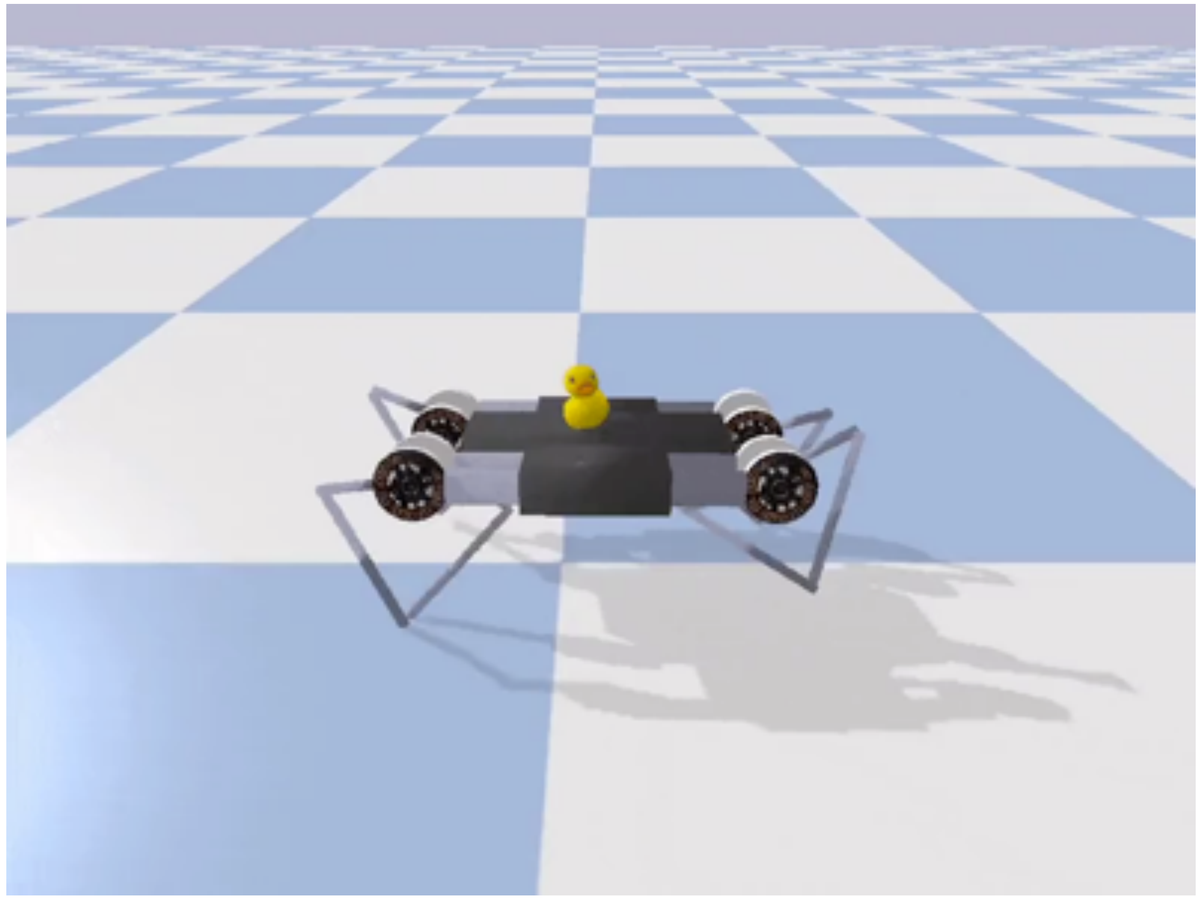
Check out “Evolutionary Robotics” Tutorial by Doncieux, Bredeche, Mouret at GECCO tomorrow!

PyBullet Minitaur Task (Erwin Coumans)



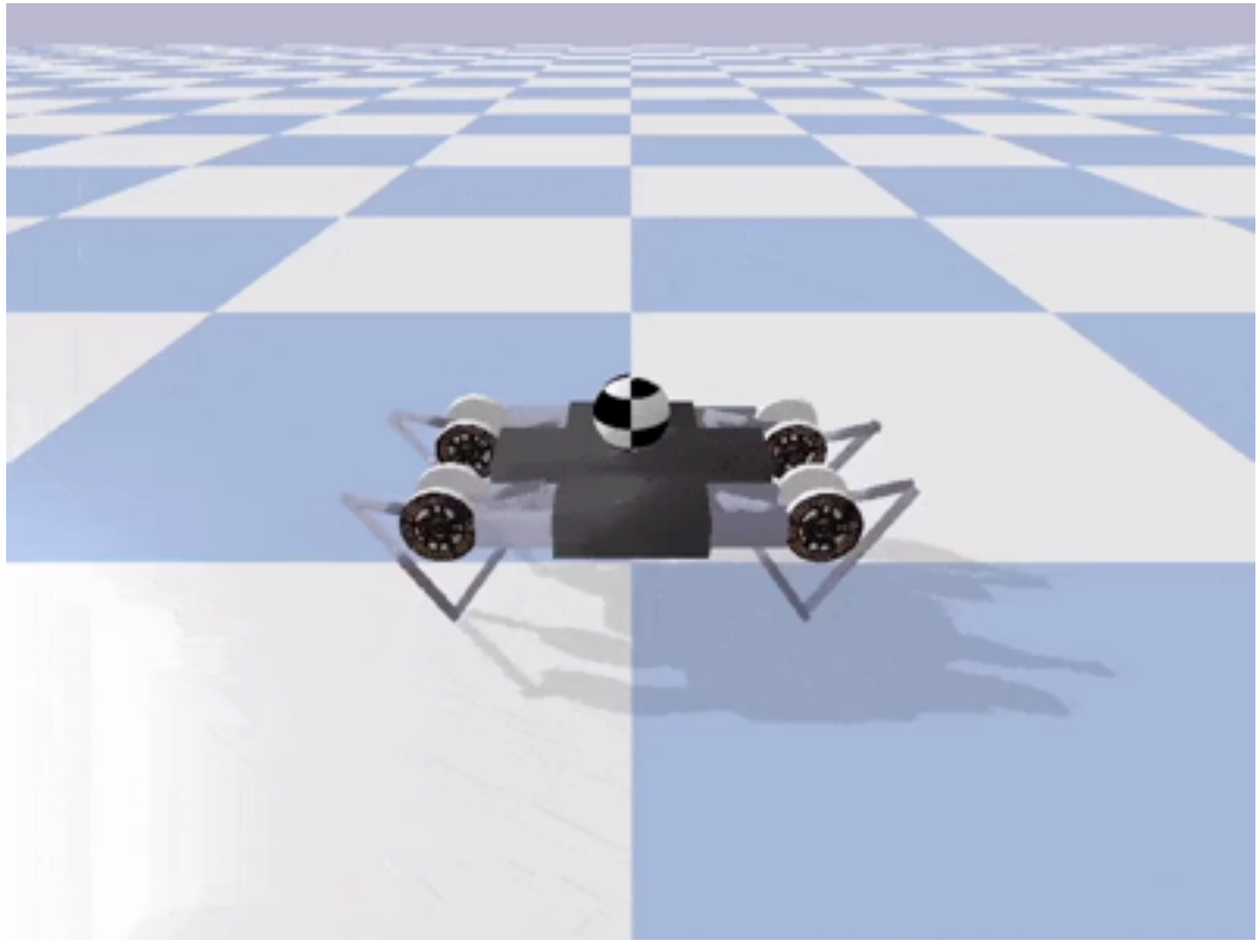
- PyBullet includes realistic models of actual robots.
- Useful to experiment with transfer learning.

Robust Minitaur Environments



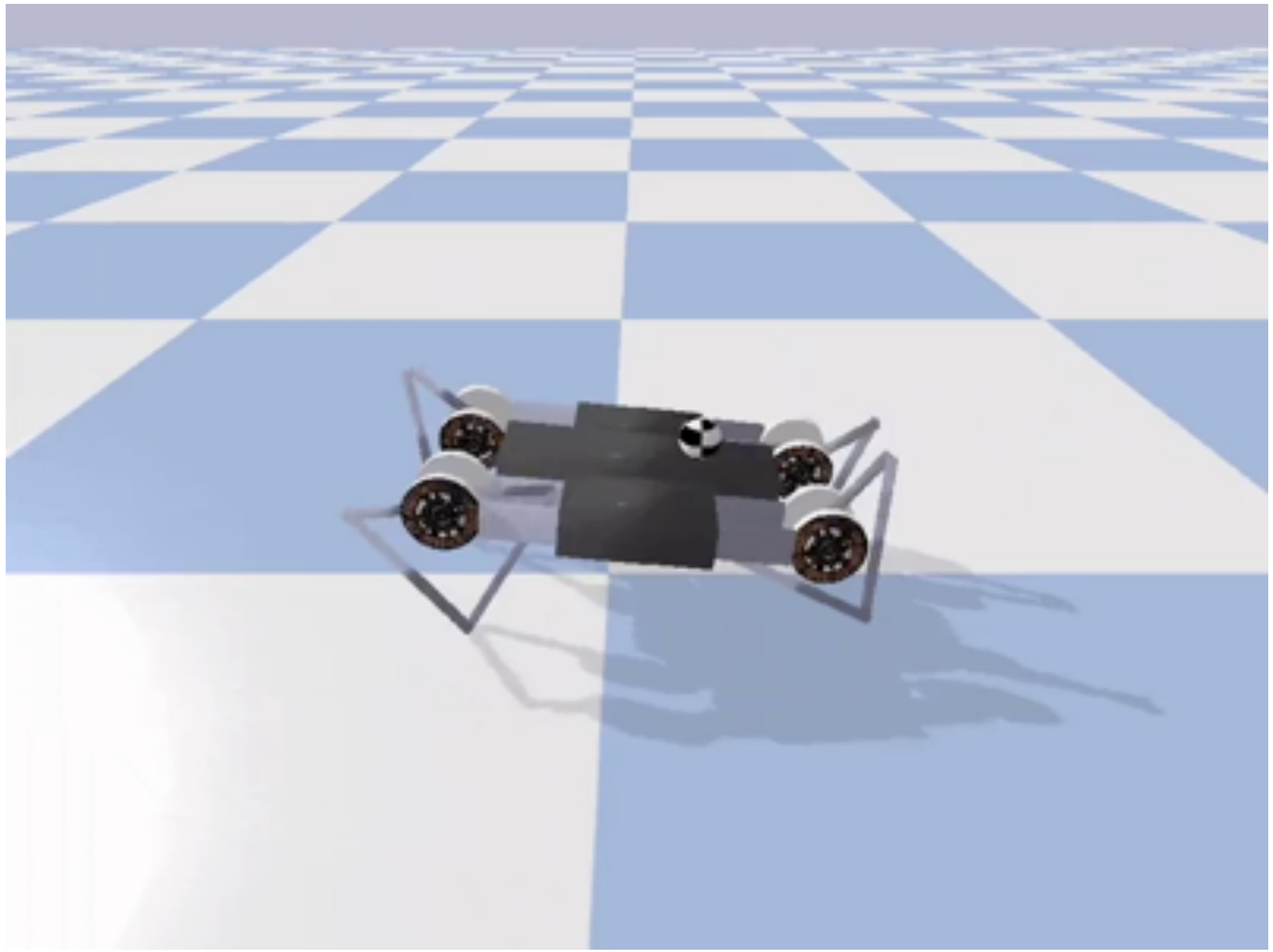
- Add more difficult task to the existing environment.
- End rollout once either objective failed.

Robust Minitaur Environments



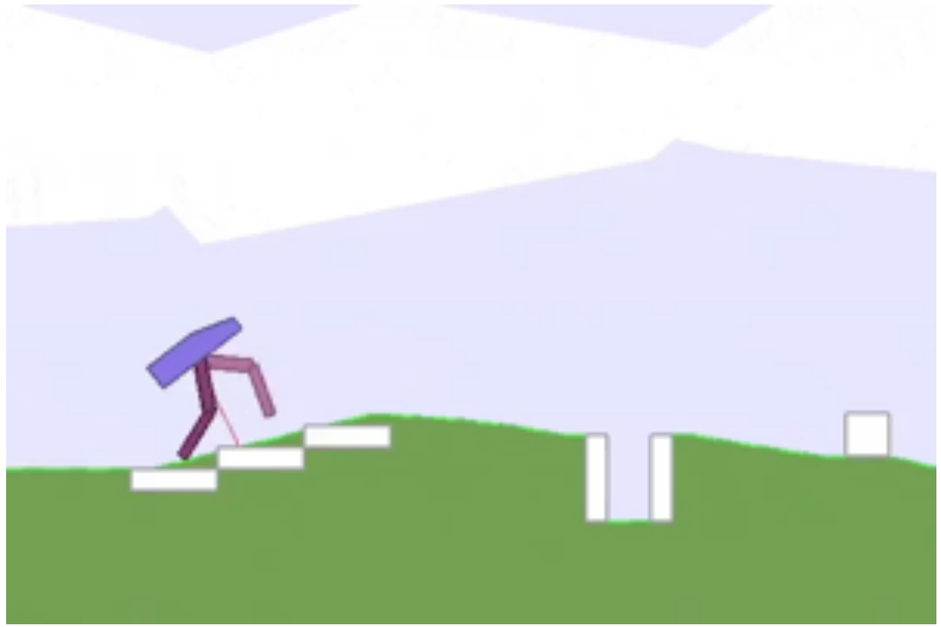
- It learned to cheat.

Robust Minitaur Environments

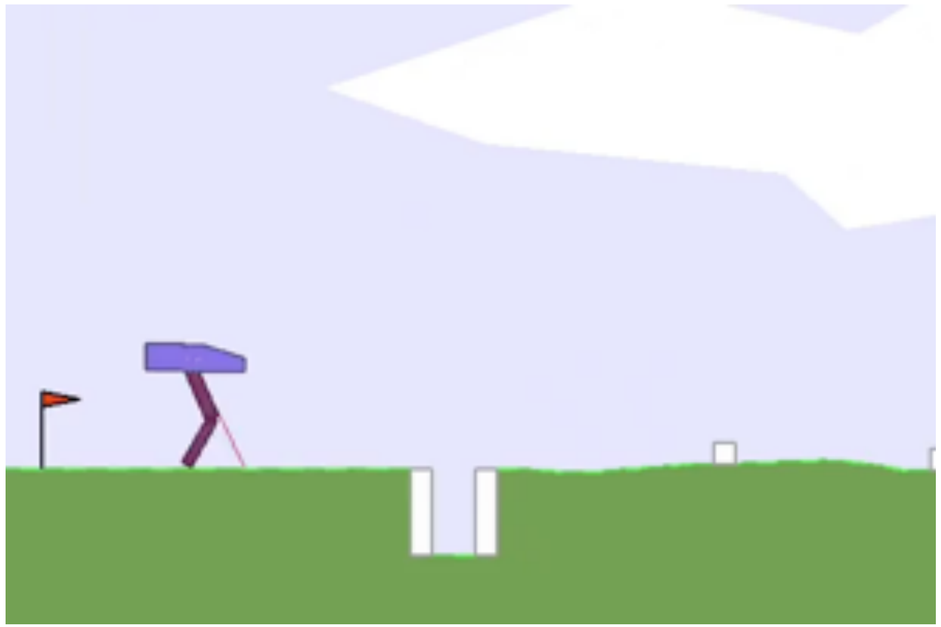


- Showed it who is boss.

ES Solved BipedalWalkerHardcore-v2



CMA-ES



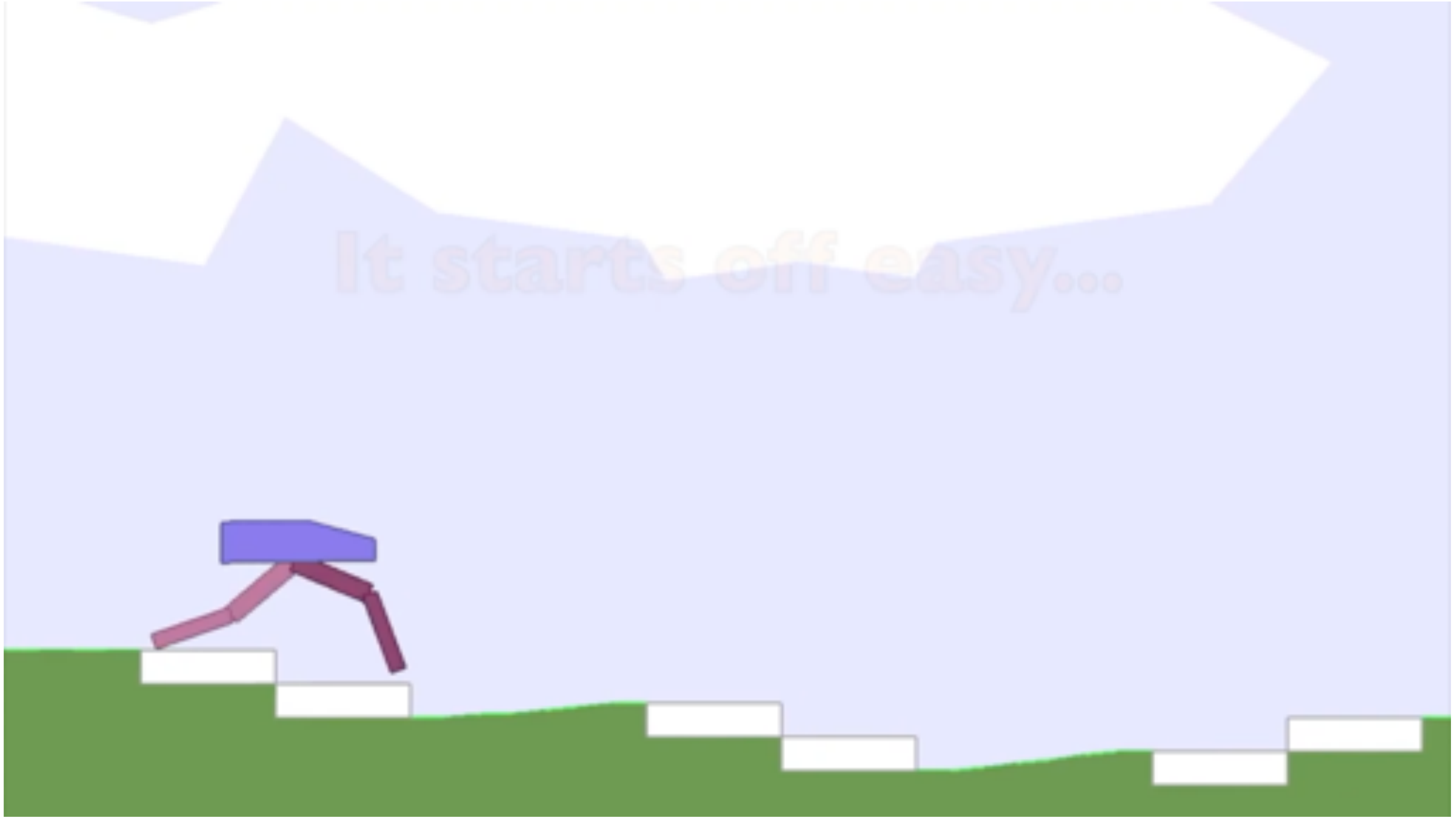
PEPG

- First solution to achieve score > 300 over 100 random trials.
- Subsequently solved with A3C, but needs ConvLSTM network.

Some humans come up with creative ways to walk too . . .

(Source: Chuck Berry, Stable Baselines Project, by Antonin Raffin 2018)

Open-endedness: Let environment adapt to the agent's capability

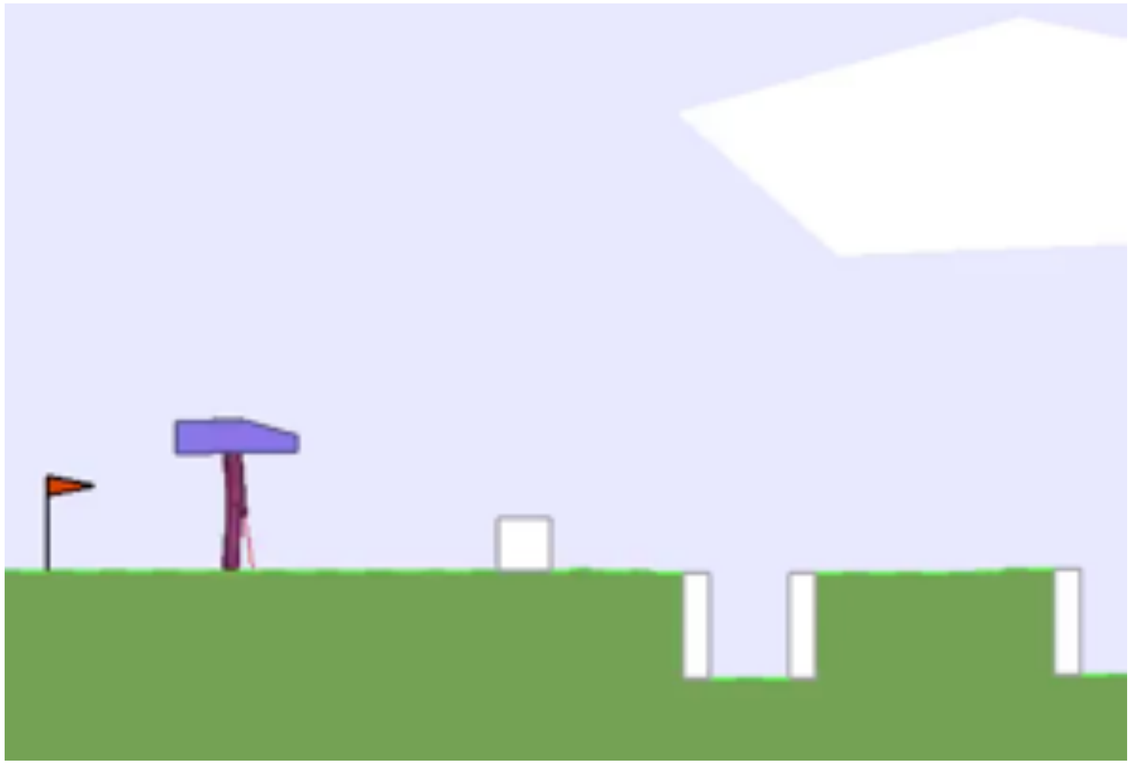


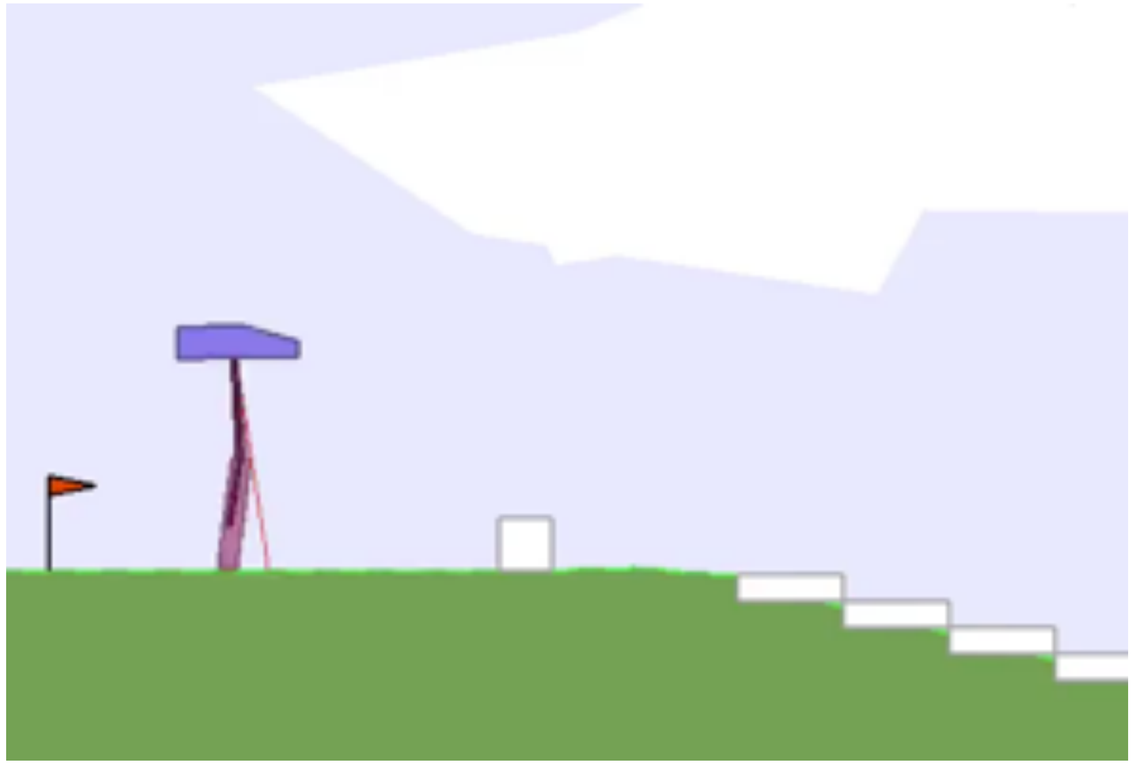
Wang, Lehman, Clune and Stanley, POET: Open-Ended Coevolution of Environments and their Optimized Solutions (GECCO 2019)

Self Modifying Agents

designrl.github.io

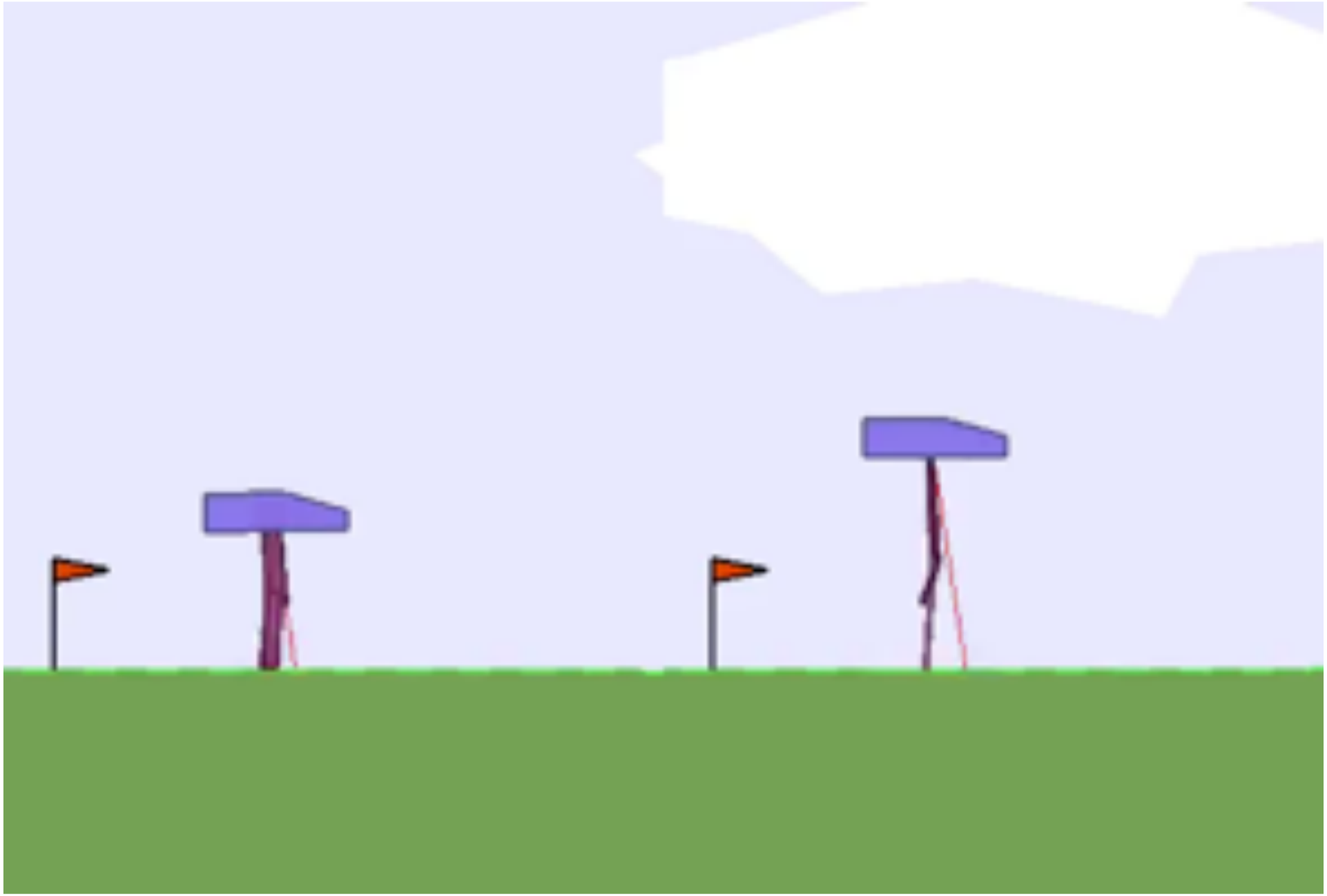
Agents that learns a better body design jointly with the navigation



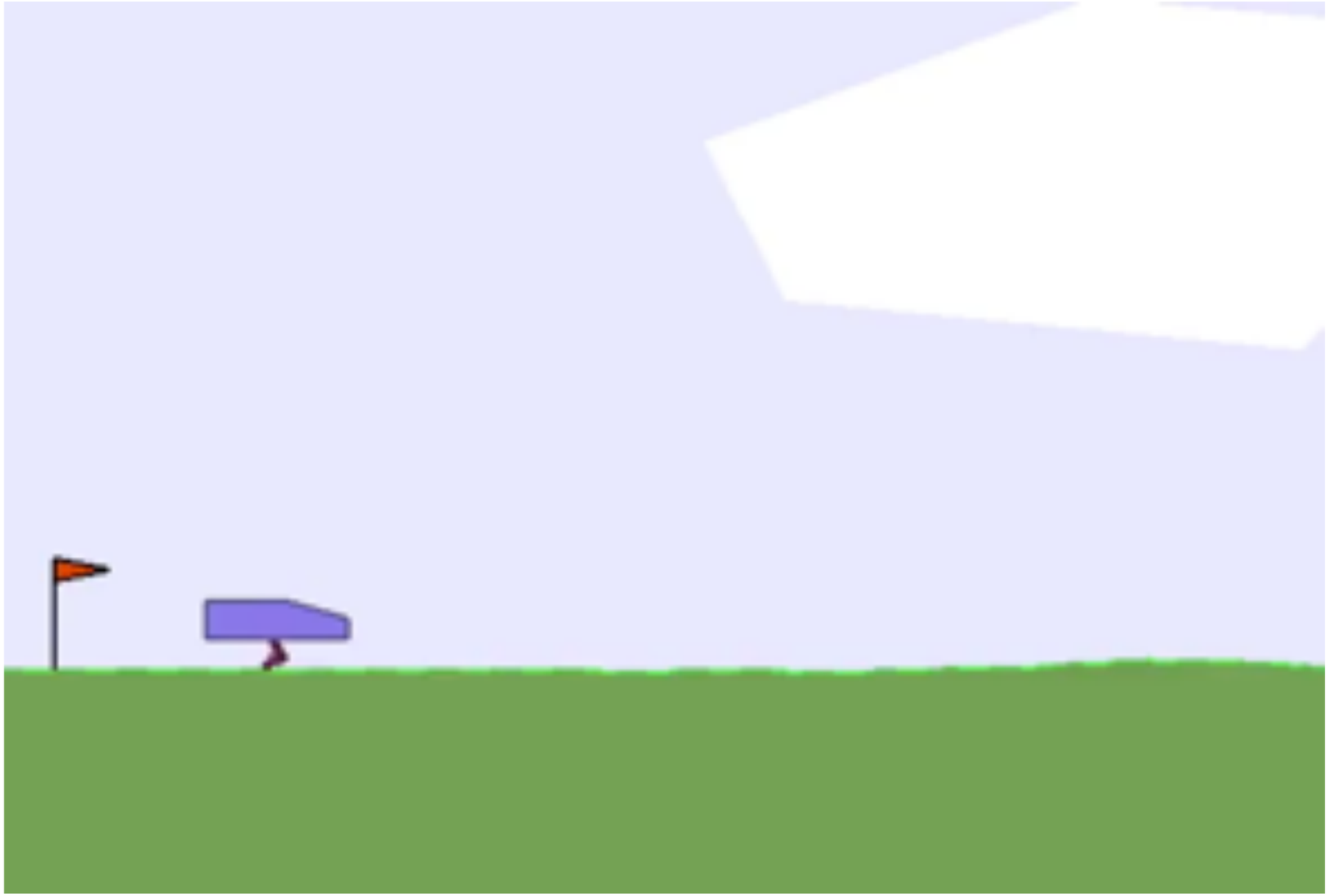


Reinforcement Learning for Improving Agent Design (Presented at NeurIPS 2018 RL Workshop, To appear in Artificial Life, 2019)

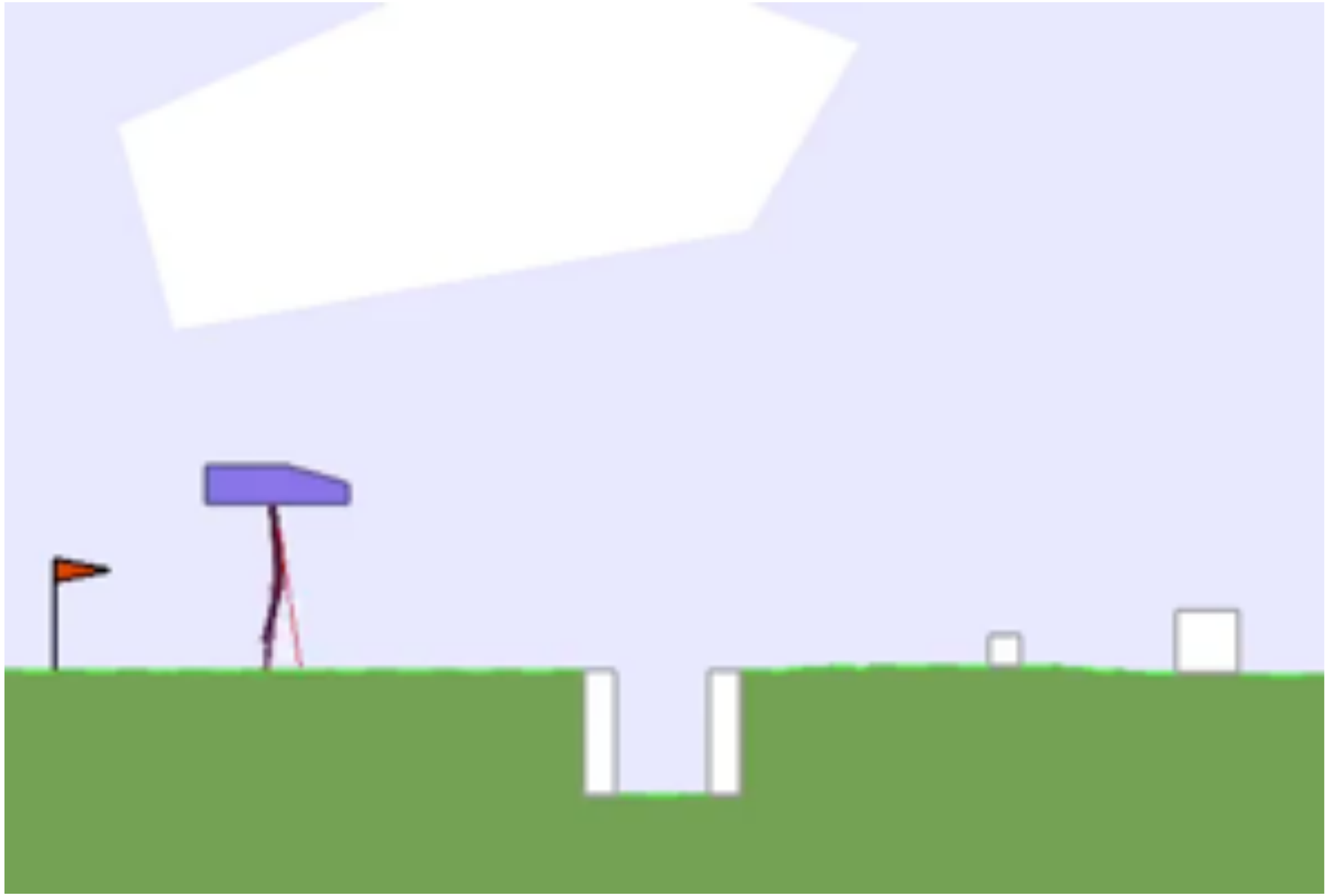
Agent learns a body to allow it to bounce forward efficiently.



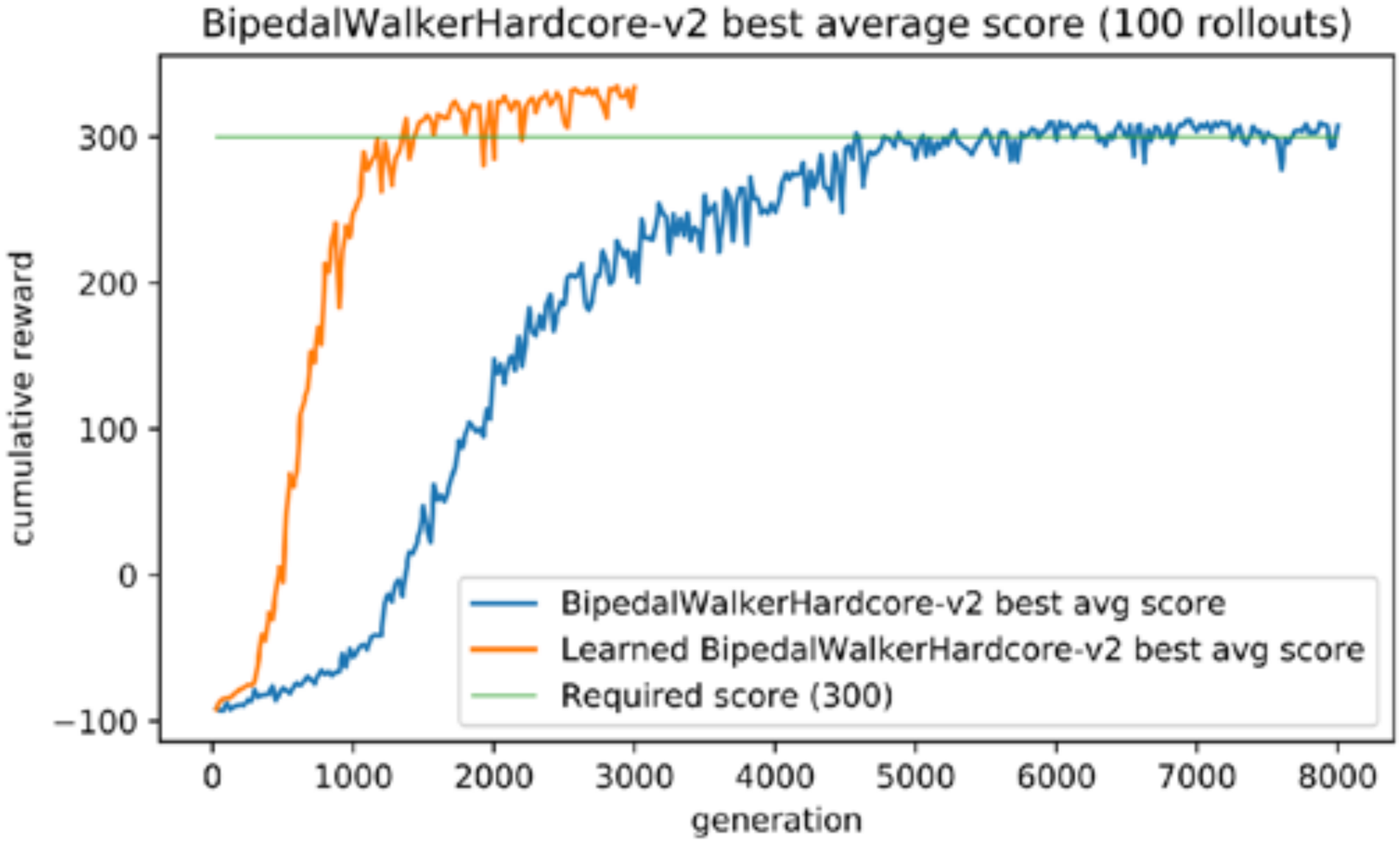
Optimizing for small legs



Optimizing for small legs on Hardcore level

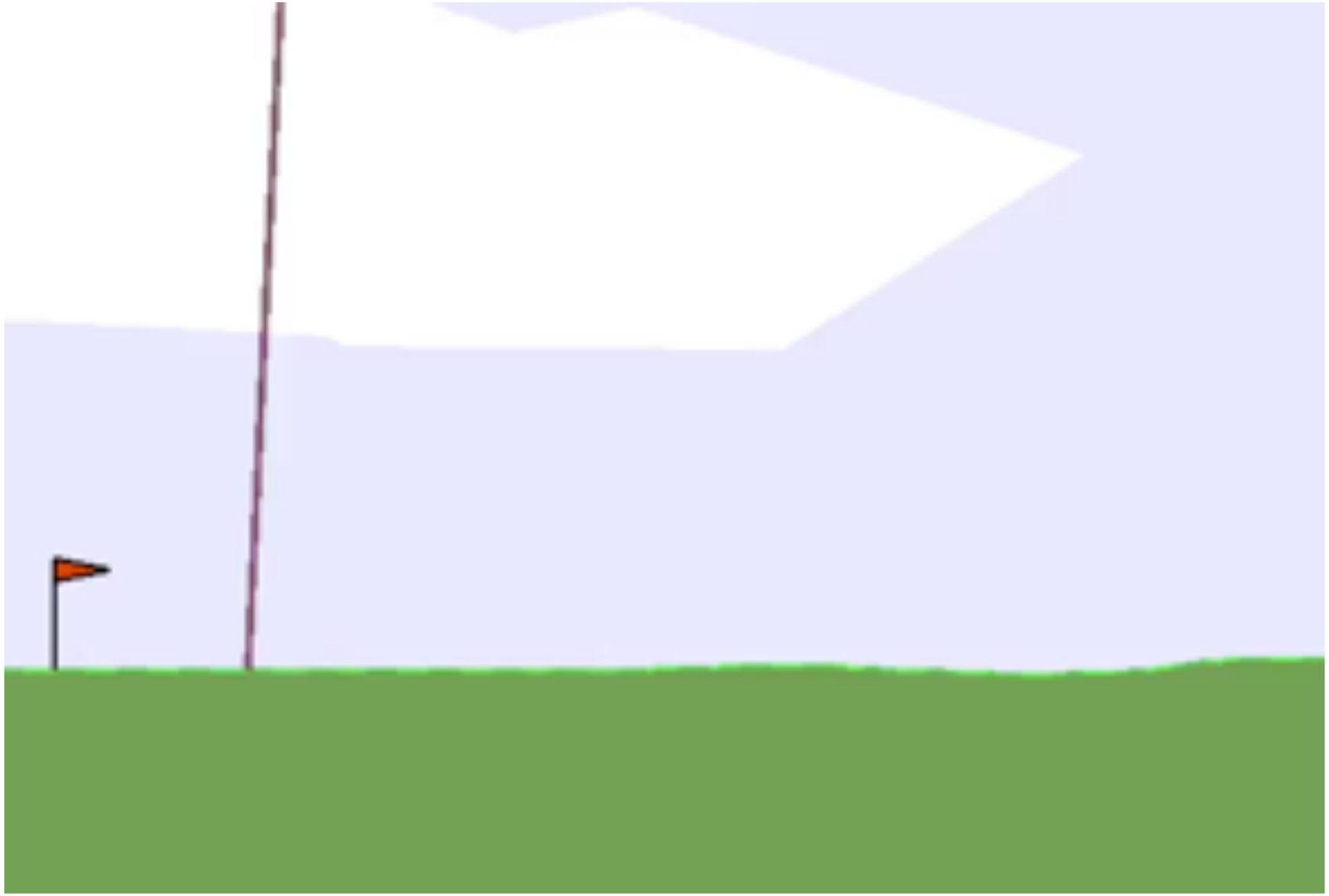


Faster policy learning when agent can modify body.

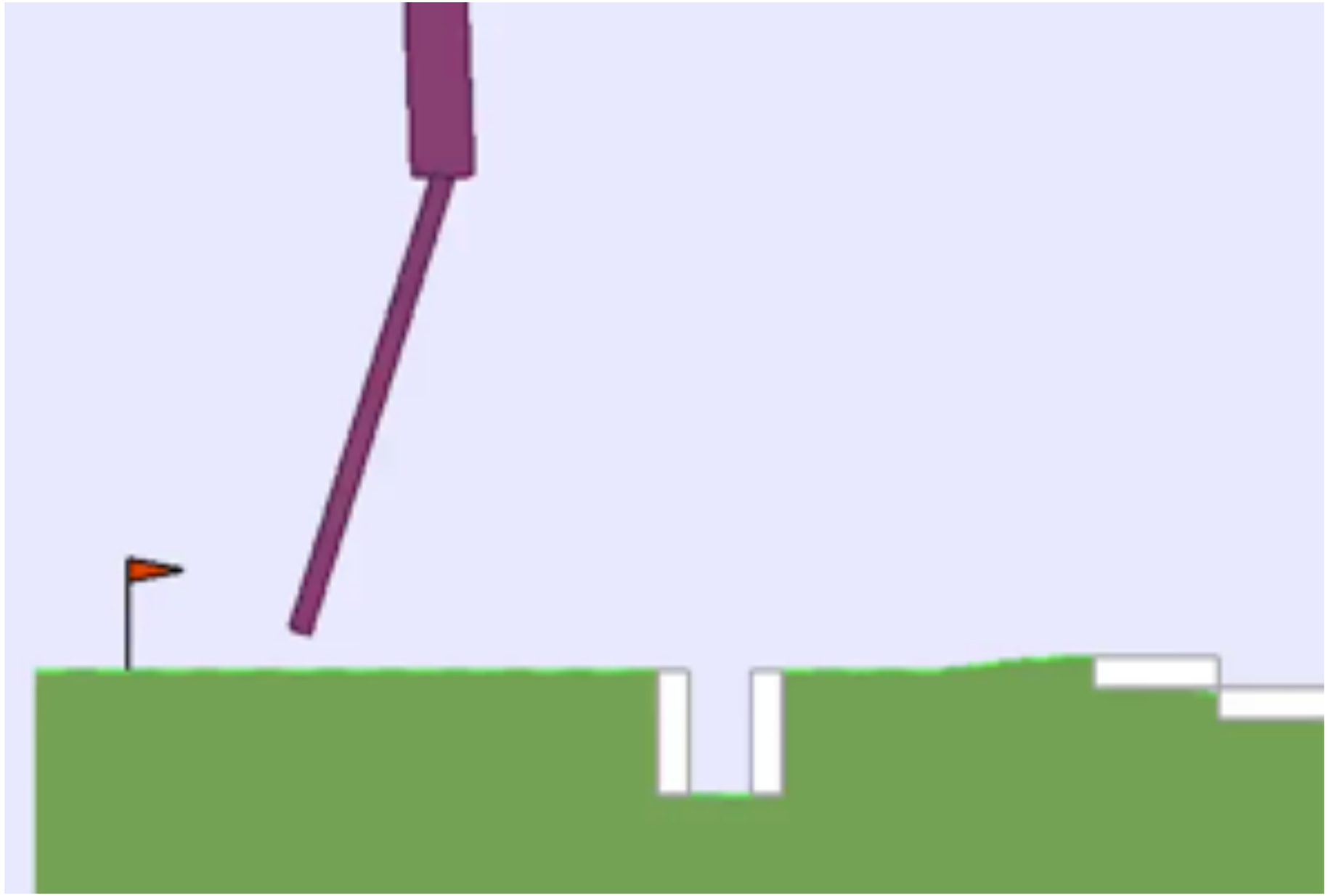


Plot of performance of best agent in the population over 100 random trials.
Original version solved under 4600 generations (40 hours), while learnable one solved under 1400 generations (12 hours).

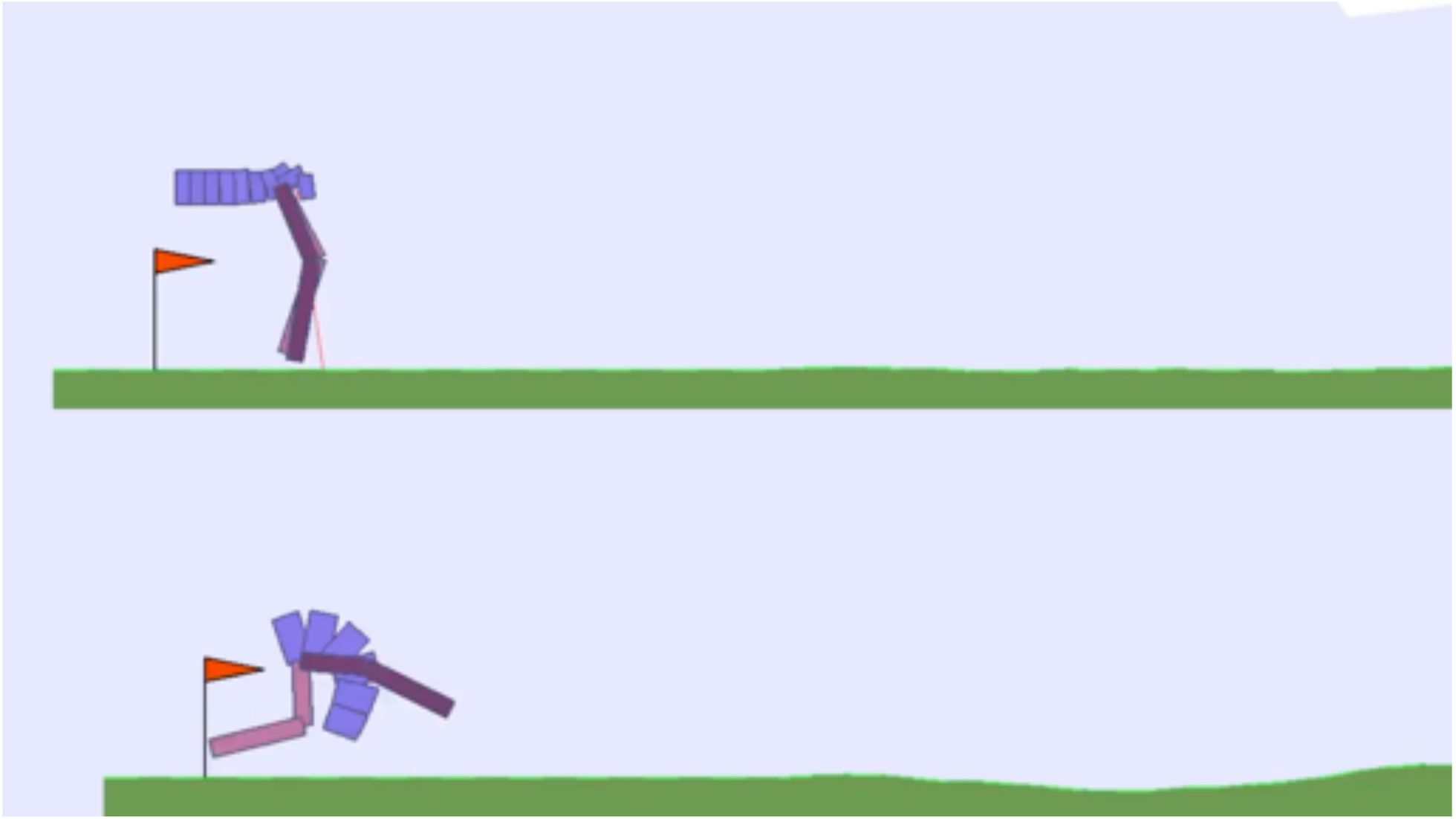
Optimizing Design with no constraints.



Optimizing Design with no constraints.

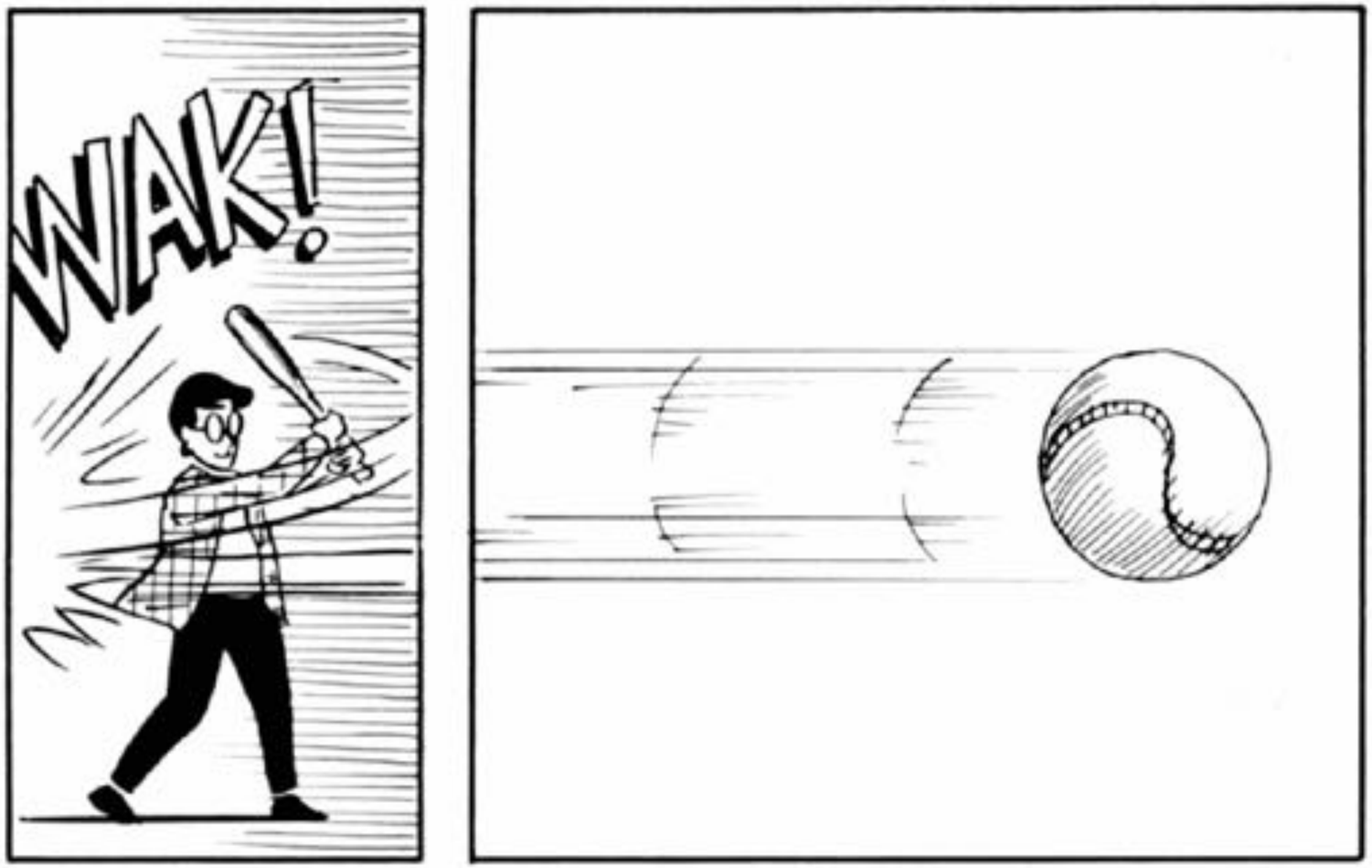


Learn a walking policy transferable across many randomized body configurations



Zhanson, Parisotto and Salakhutdinov, Proprioceptive Spatial Representations for Generalized Control, (ICLR Workshop 2019)

Representation Learning for Reinforcement Learning



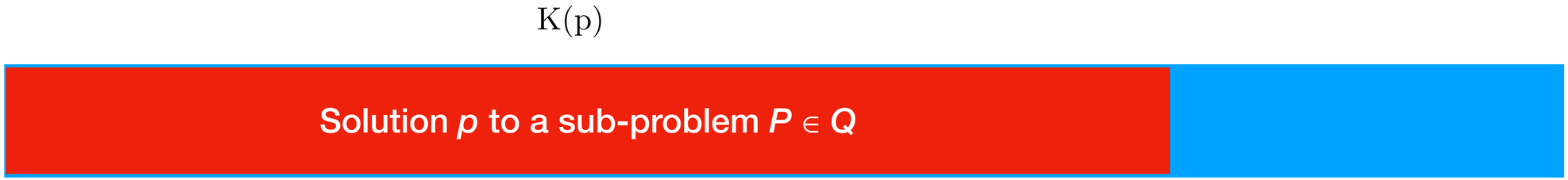
“In the world of comics, time and space are one and the same.”
— Scott McCloud, Understanding Comics

Basic Algorithmic Information Theory Argument

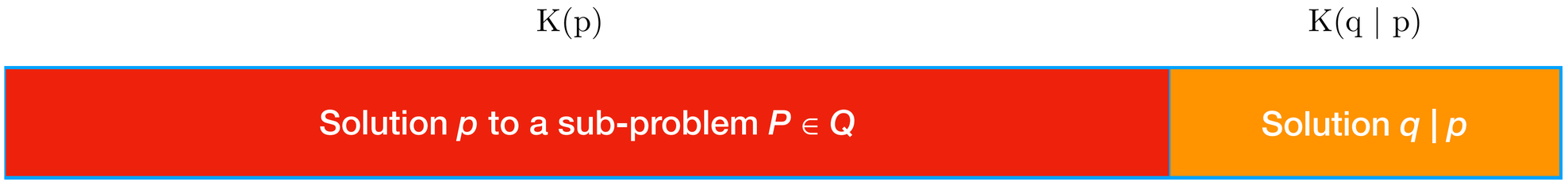
K(q): Kolmogorov complexity

Find a solution, q, to a difficult problem, Q

Basic Algorithmic Information Theory Argument



Basic Algorithmic Information Theory Argument



If p is a solution to problem P and if $K(q \mid p)$ is much shorter than $K(q)$, then finding p first, and searching for $q \mid p$ to compute and thus solve Q is much faster than searching for q from scratch.

Schmidhuber, On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models, (Technical Report, 2015)

Basic Algorithmic Information Theory Argument

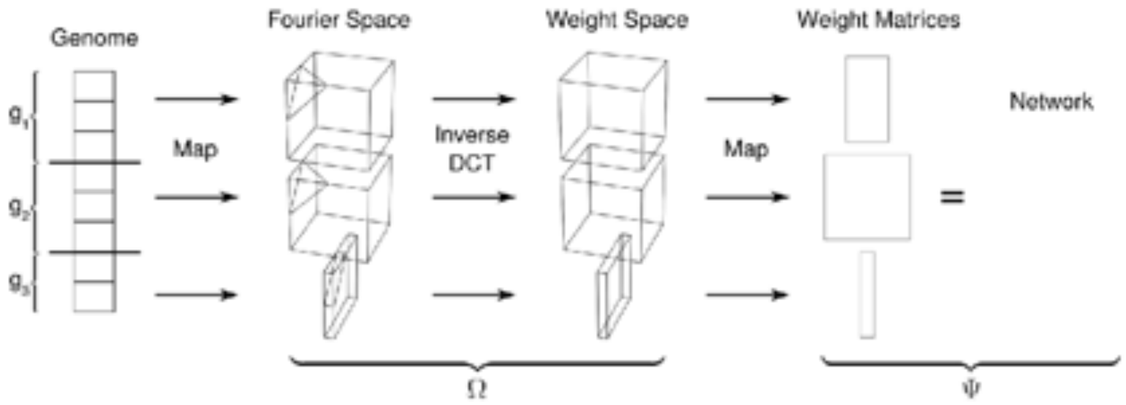
Species

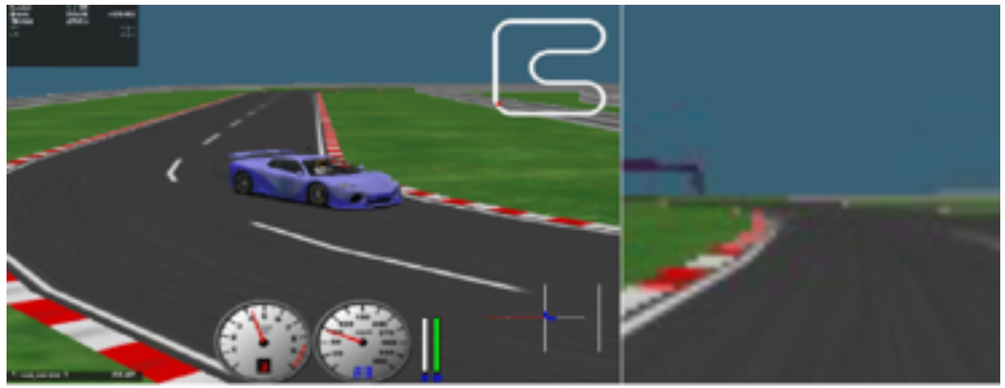
phenotype + learning from experiences in natural environment “nurture”	genotype “nature”

Basic Algorithmic Information Theory Argument

Agent in artificial environment (Indirect Encoding)

Weights produced using HyperNEAT algorithm Weights decompressed using Discrete Cosine Transform algorithm	NEAT parameters DCT coefficients





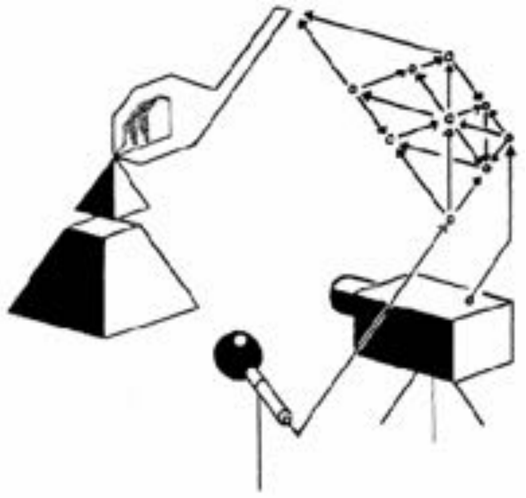
Stanley et al., A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks (HyperNEAT) (Artificial Life, 2009)

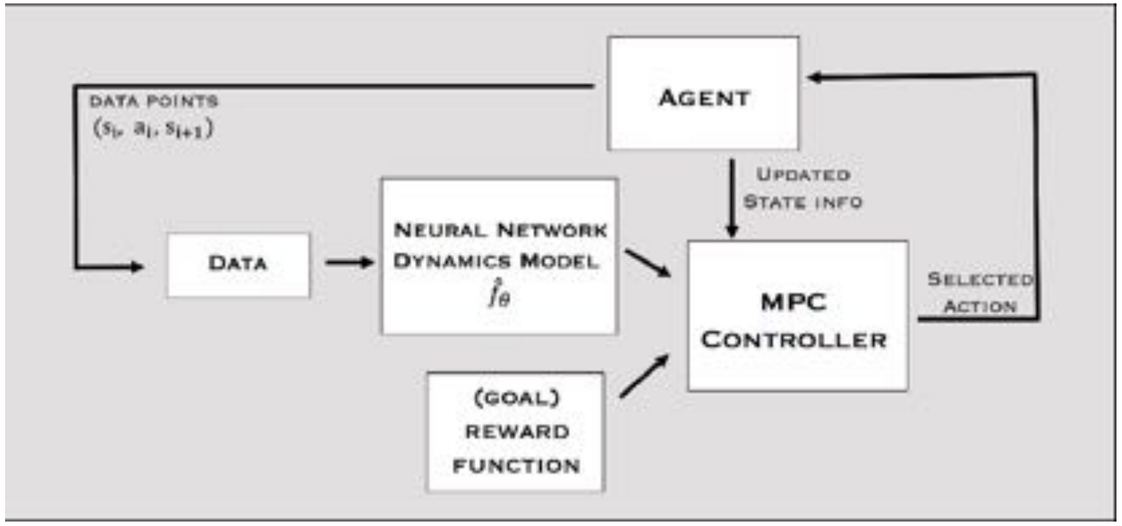
Koutník, Evolving Large-Scale Neural Networks for Vision-Based Reinforcement Learning, (GECCO 2013)

Basic Algorithmic Information Theory Argument

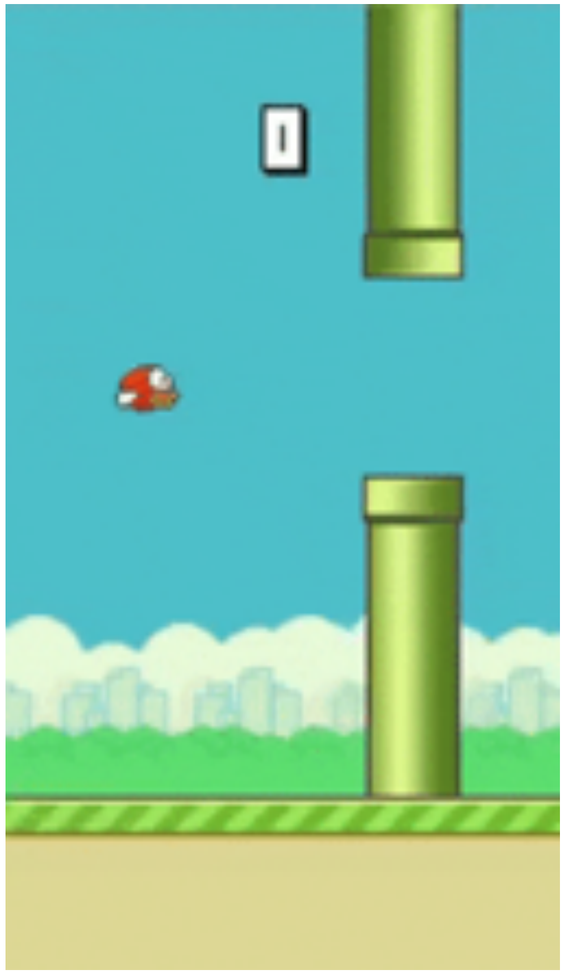
Agent in artificial environment (Model-based Reinforcement Learning)

Internal Representation of Time and Space (Model) (Hand-engineered, realistic physics model, or learned from data)	Controller (Policy Search)





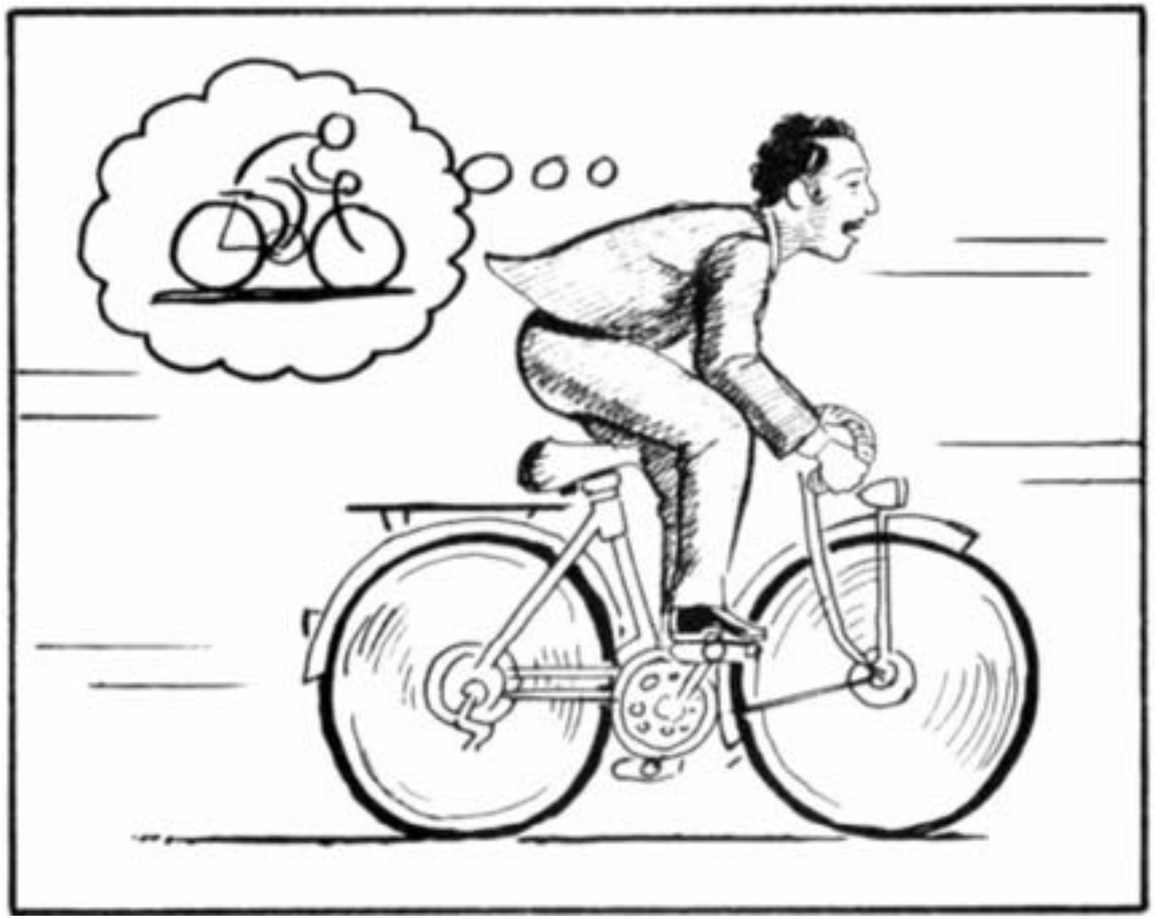
Basic Algorithmic Information Theory Argument for Representation Learning

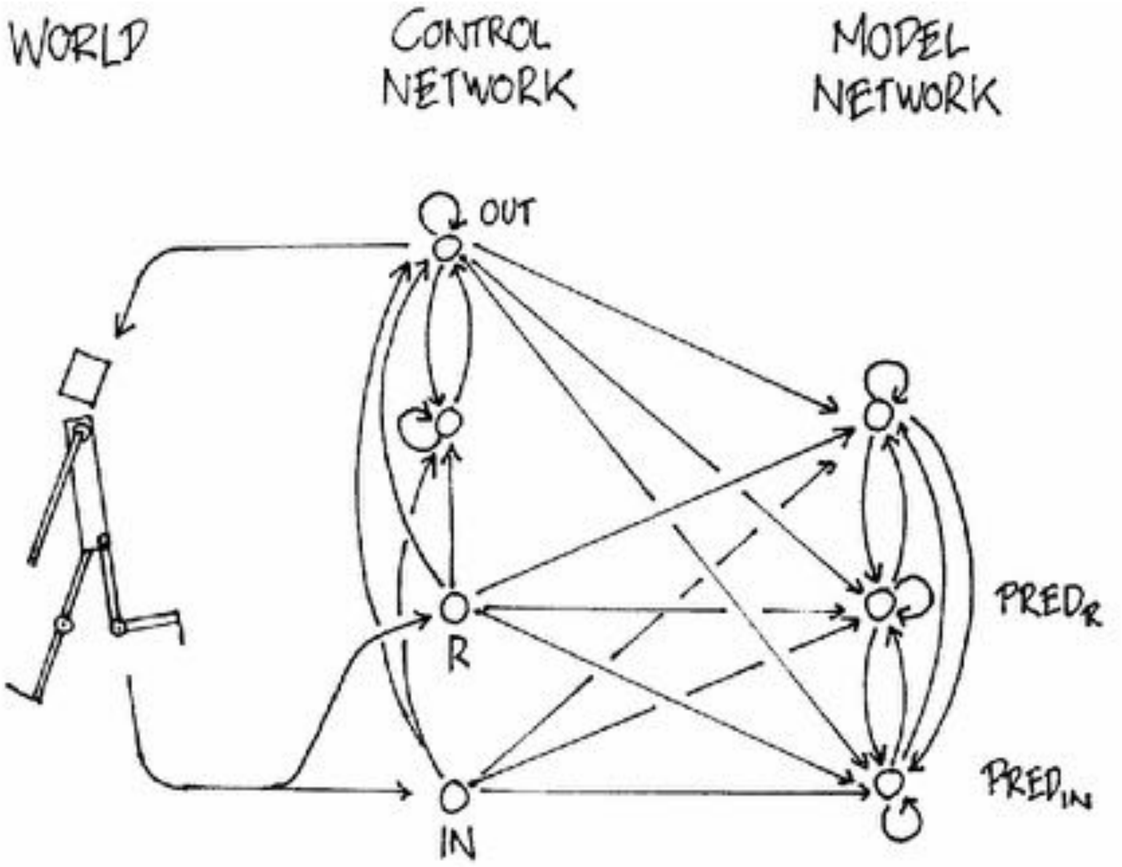


- Many RL tasks requires representation learning and predicting the future.
- We can efficiently train highly expressive models to learn representations of space and time with backprop.
- Use these representations as features to learn a compact policy, using Neuroevolution, to achieve the task.

World Models

worldmodels.github.io

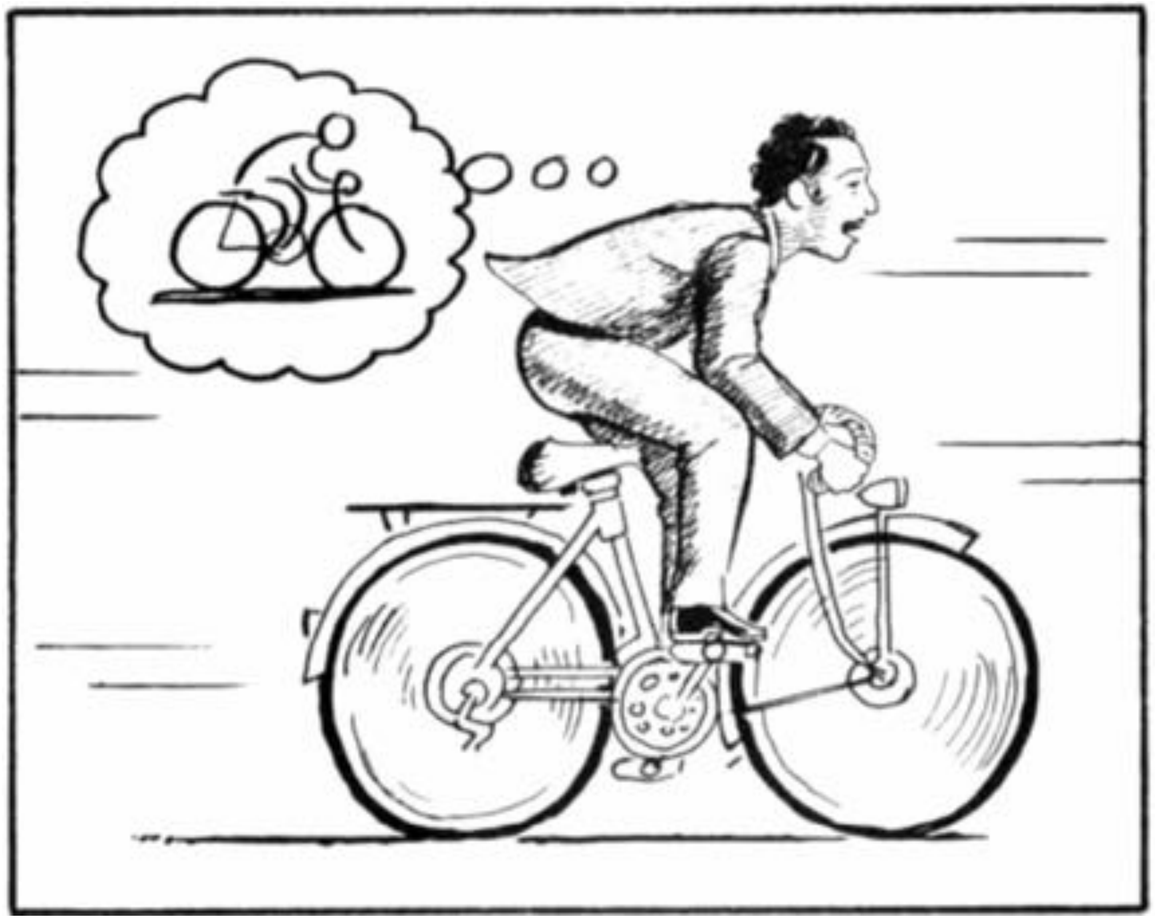




Art © Scott McCloud, Understanding Comics

Ha and Schmidhuber, Recurrent World Models Facilitate Policy Evolution (NeurIPS 2018)

Mental World Models



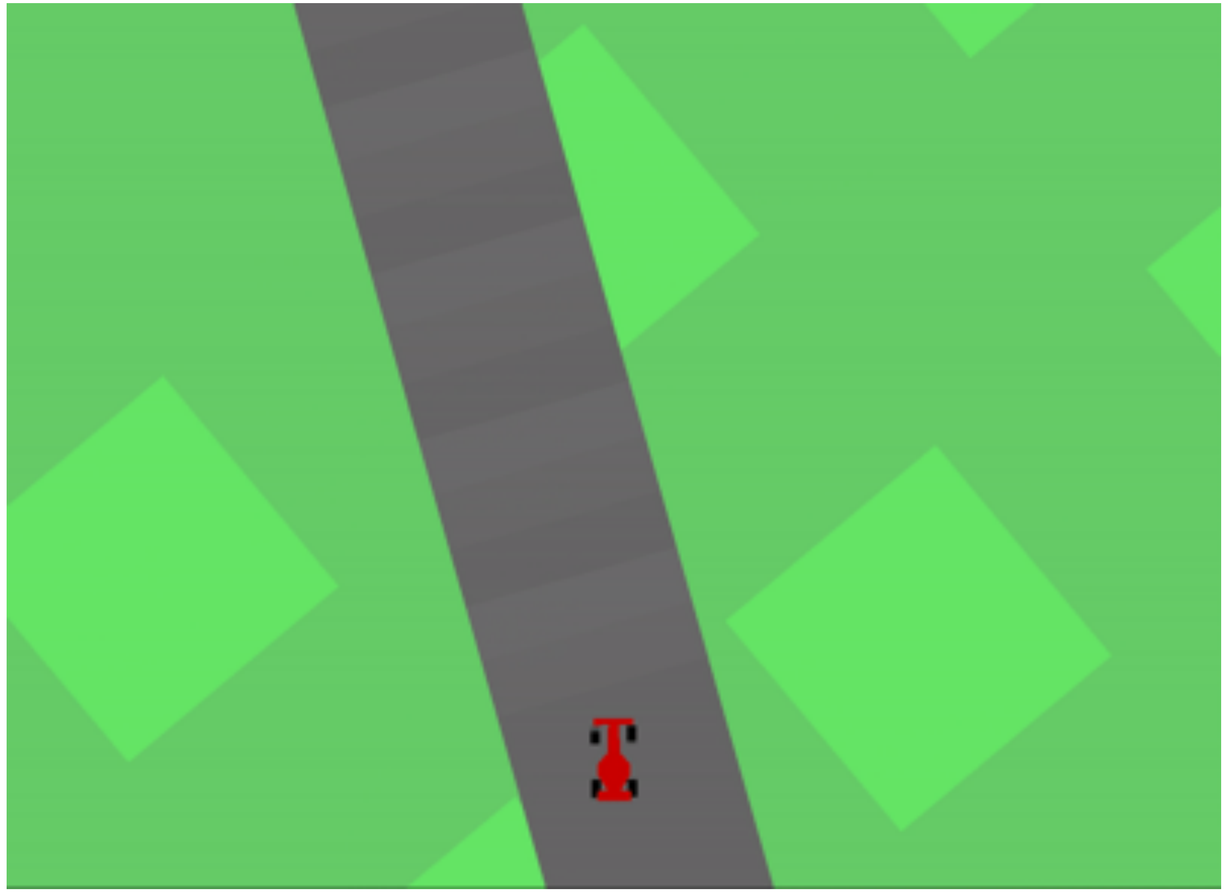
Art © Scott McCloud, Understanding Comics

"The image of the world around us, which we carry in our head, is just a model."

– Jay Write Forrester (1918 – 2016)
Father of system dynamics.

Game Environments

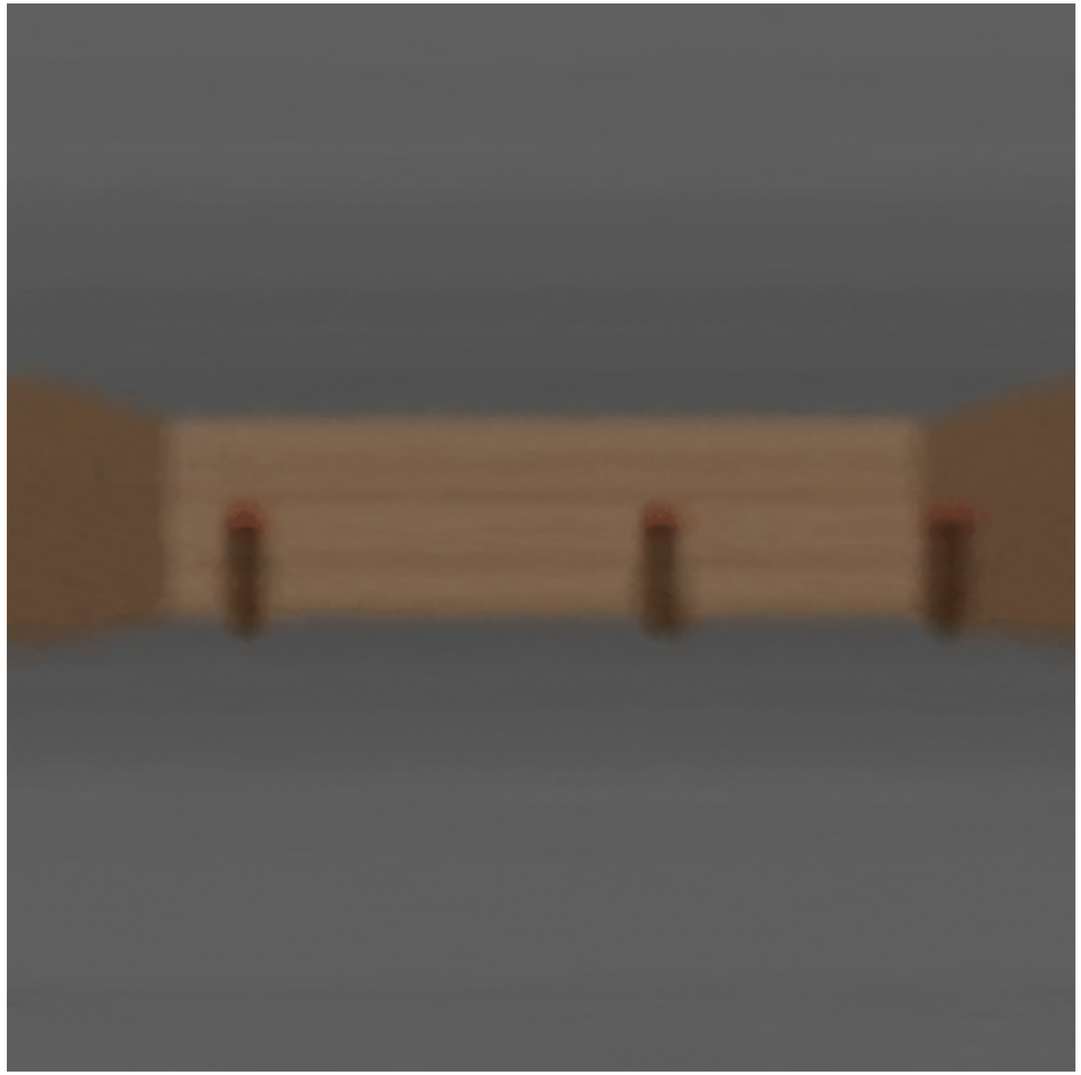
Doom TakeCover



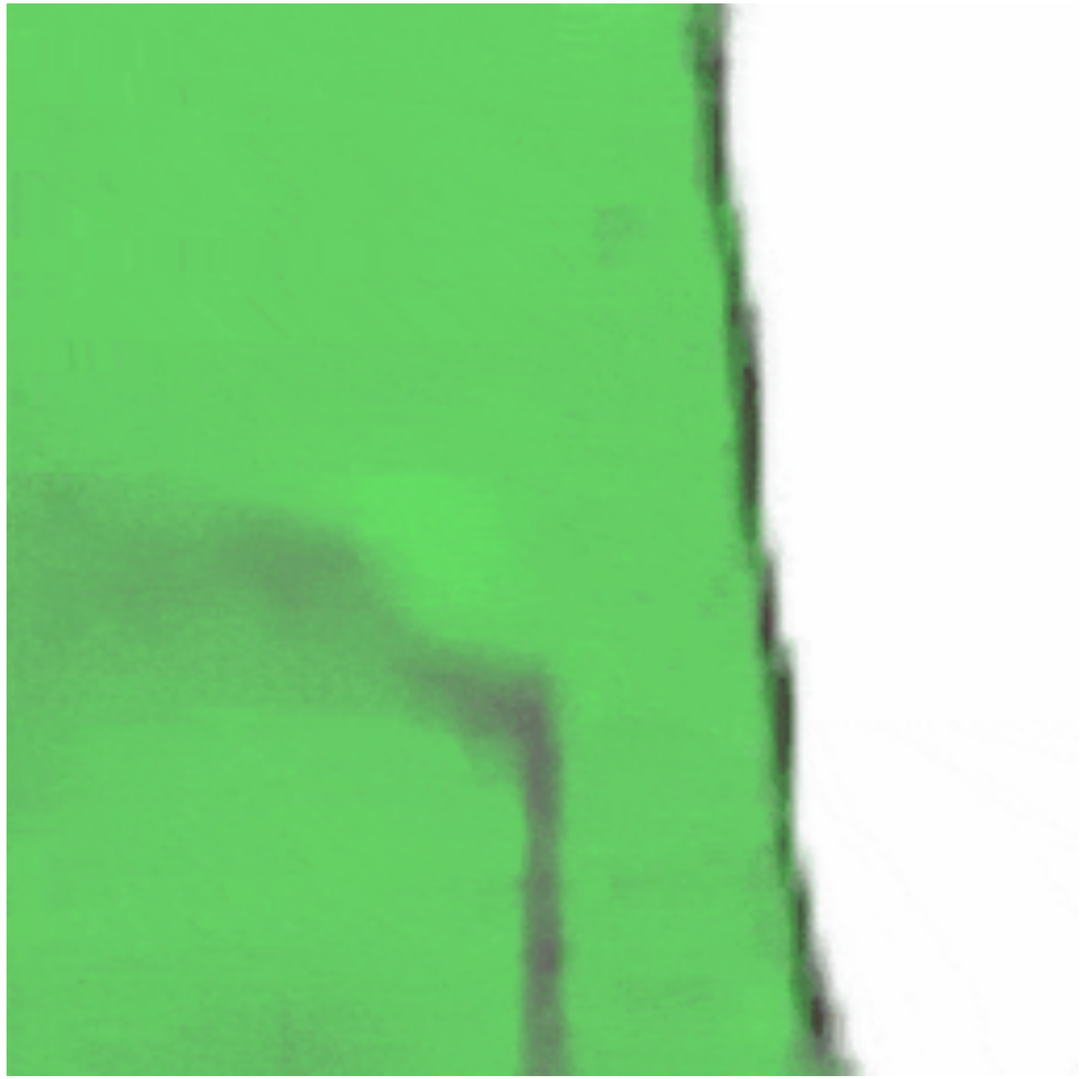
Box2D CarRacing

Source: OpenAI Gym

Generative Game Environments



Doom TakeCover



Box2D CarRacing

Variational Autoencoder

Input Image

Low Dimensional
Latent Space (Z)

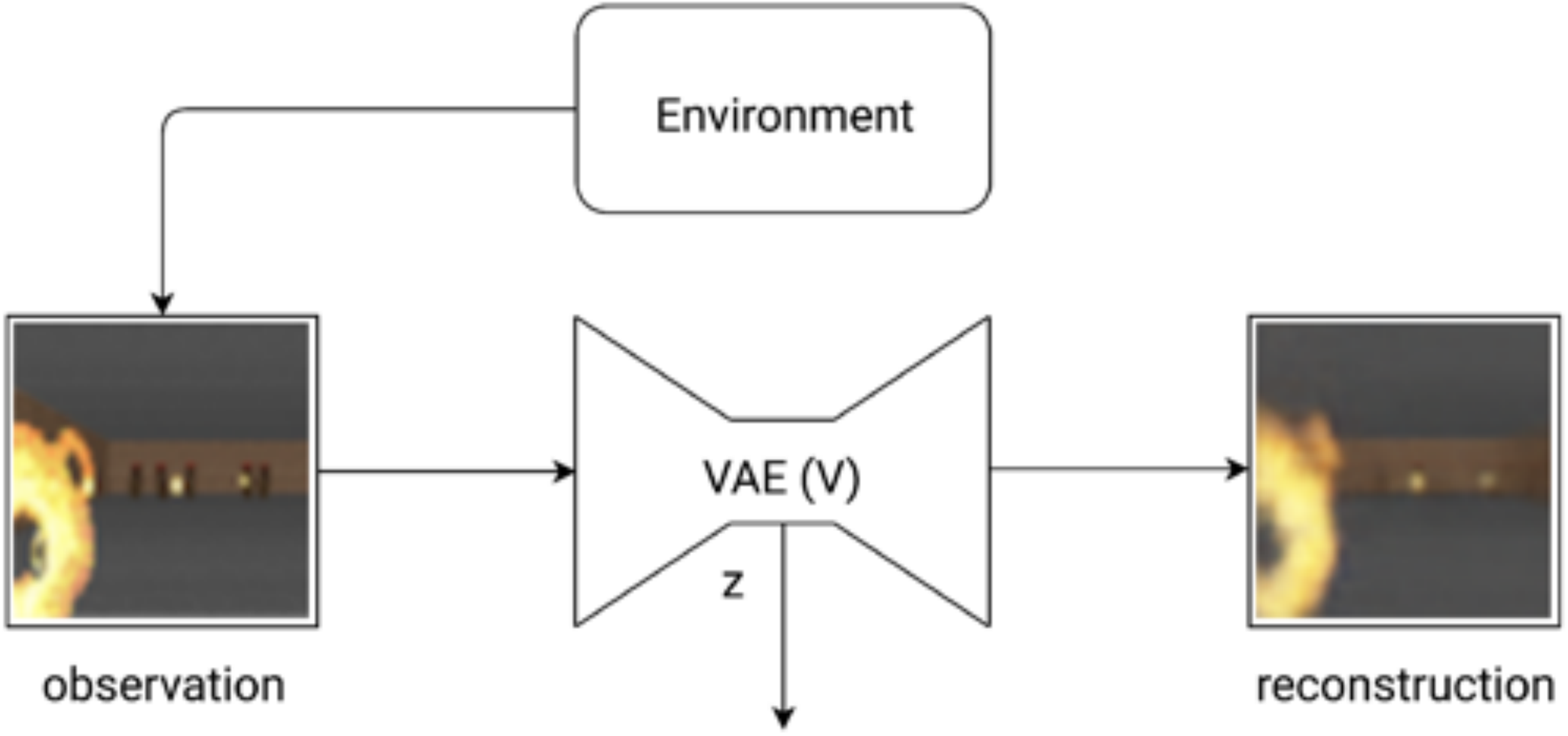
Reconstruction

Source: “Sampling Generative Networks” (Tom White, 2016)

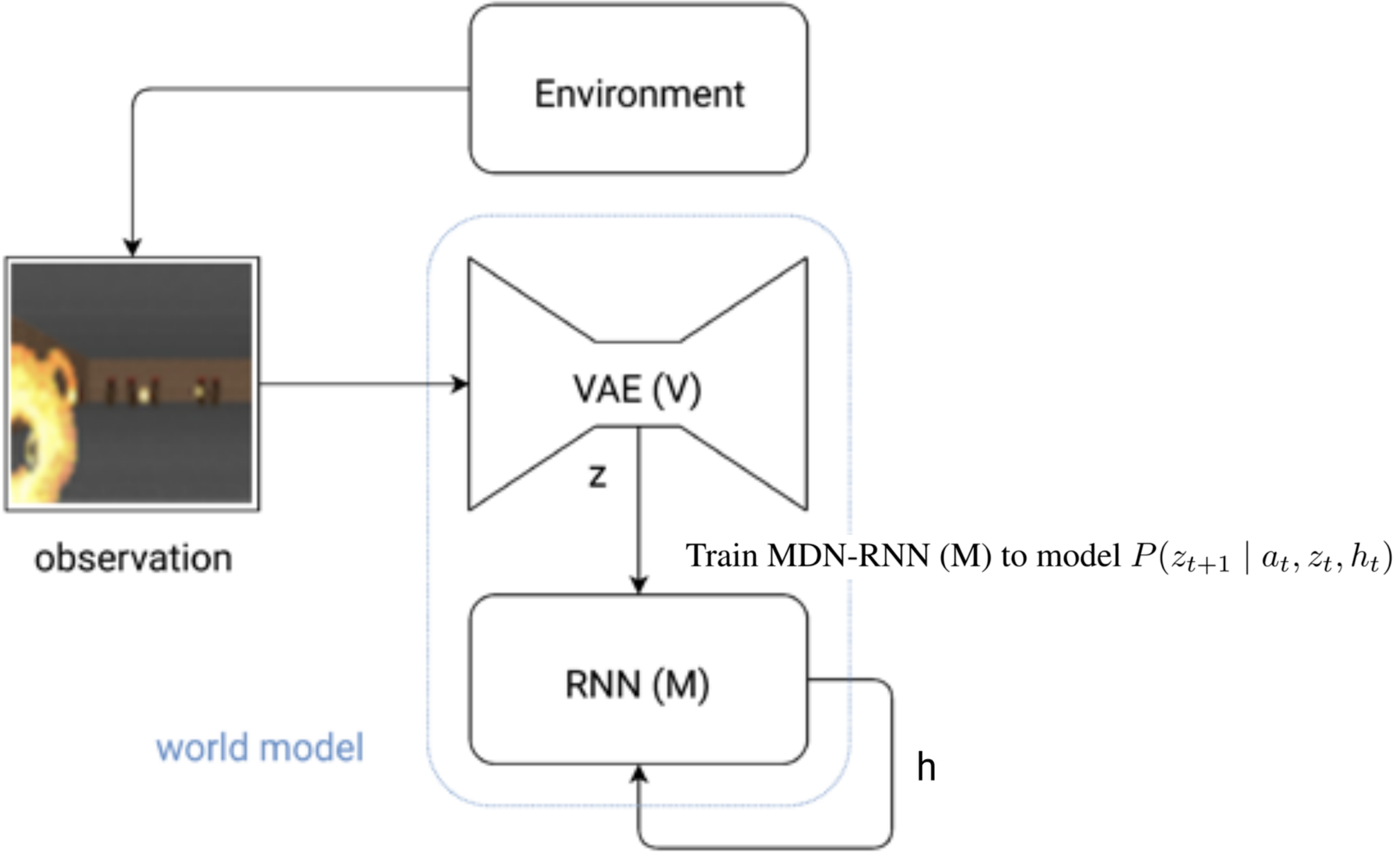
Density Estimation with Recurrent Neural Networks

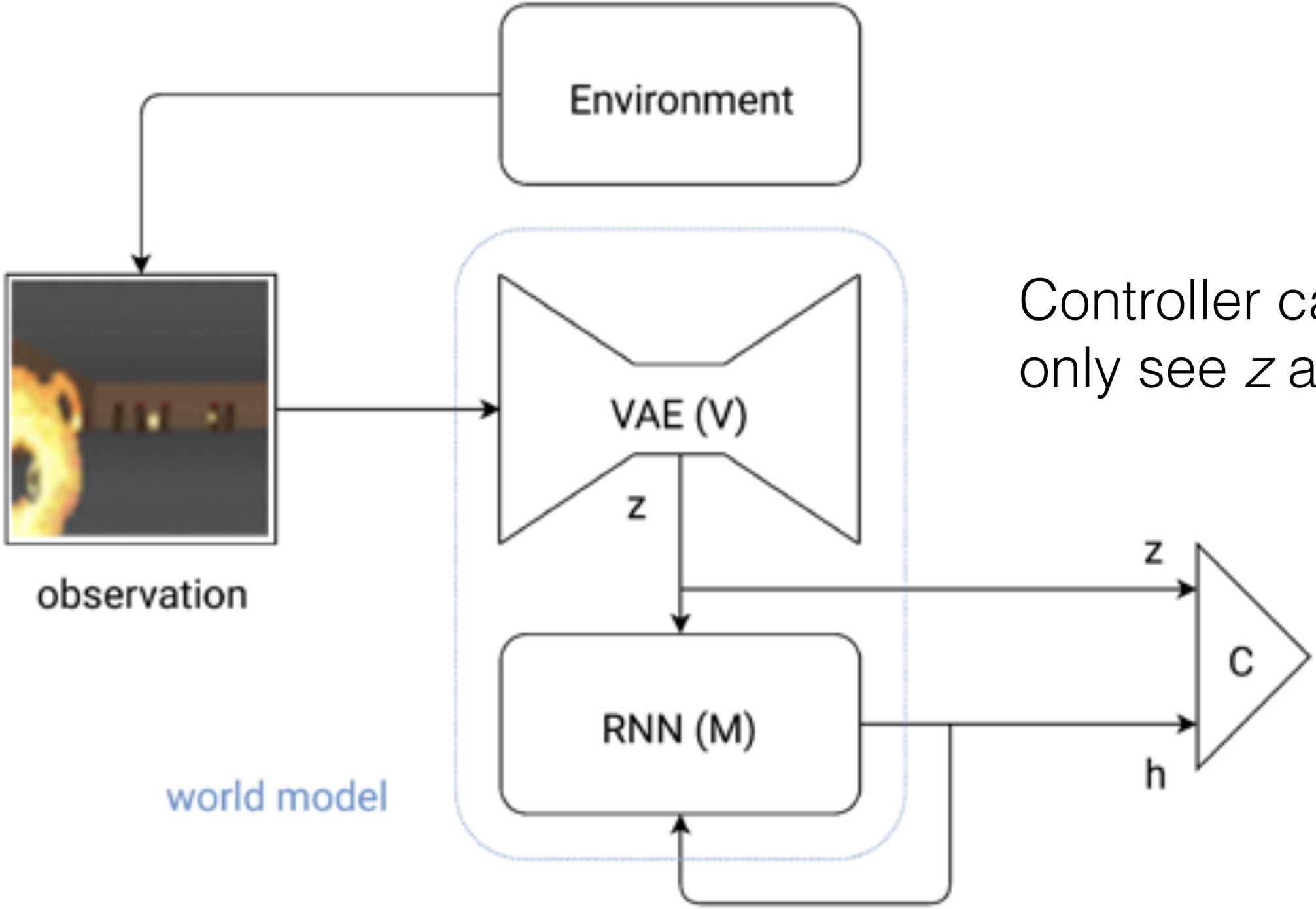
Source:“A Neural Representation of Sketch Drawings” (Ha & Eck, 2017)


Environment
observation

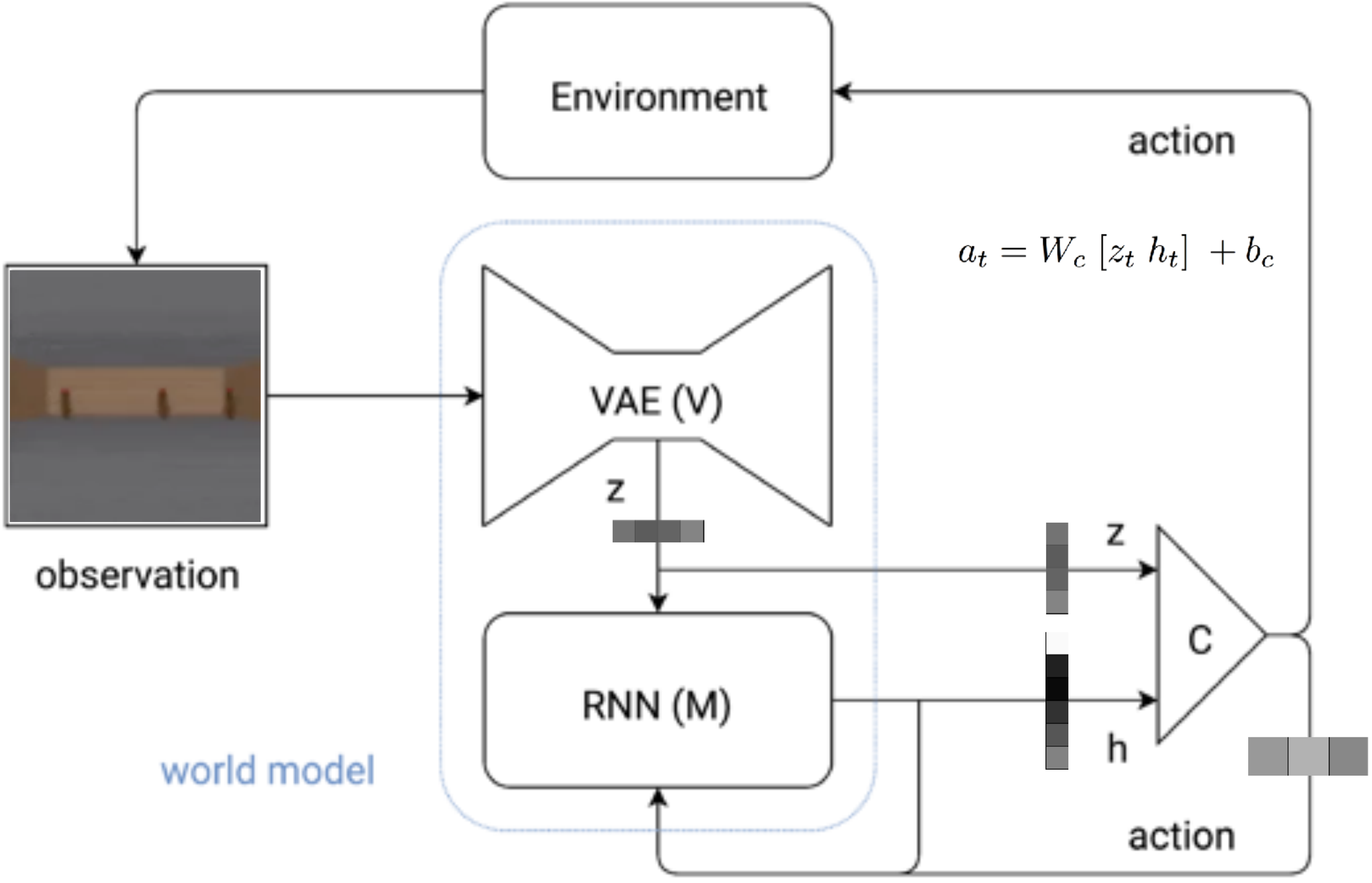


Train VAE (V) to encode frames into z

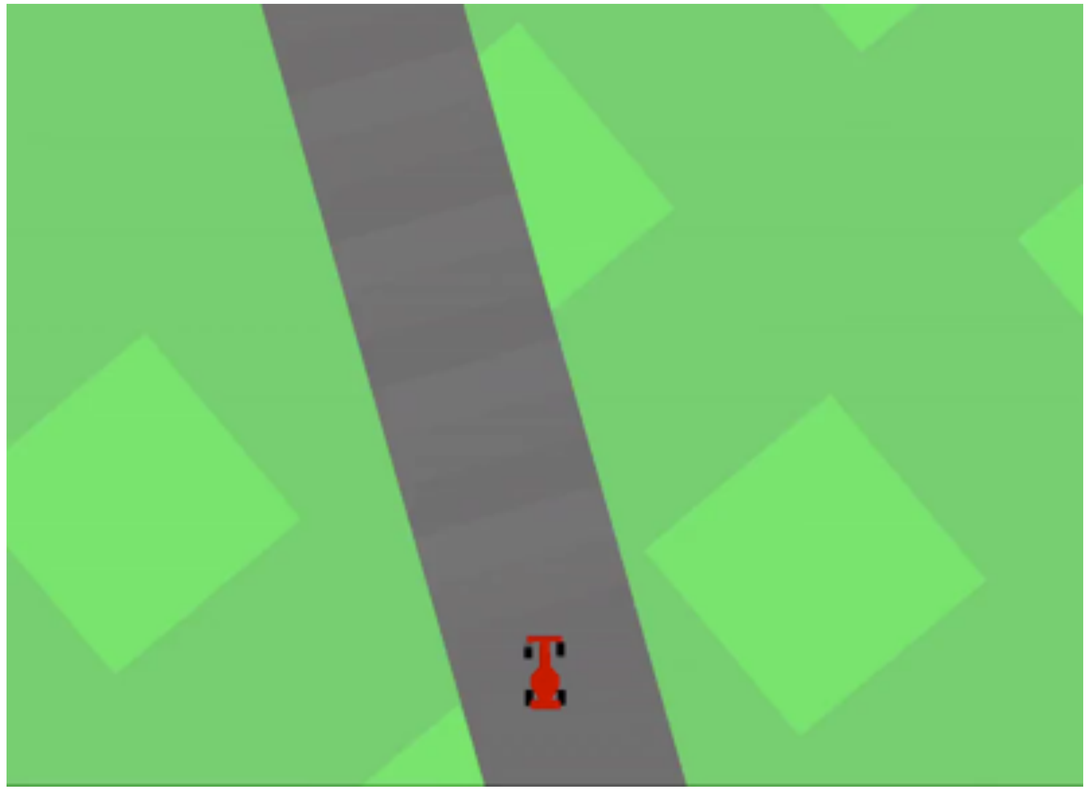

Environment
VAE (V)
z
observation
Train MDN-RNN (M) to model $P(z_{t+1} \mid a_t, z_t, h_t)$
RNN (M)
world model
h



Controller can only see *z* and *h*.


Environment
action
$a_t = W_c \, [z_t \; h_t] \; + b_c$
VAE (V)
z
observation
RNN (M)
world model
z
C
h
action

CarRacing-v0



Procedure:

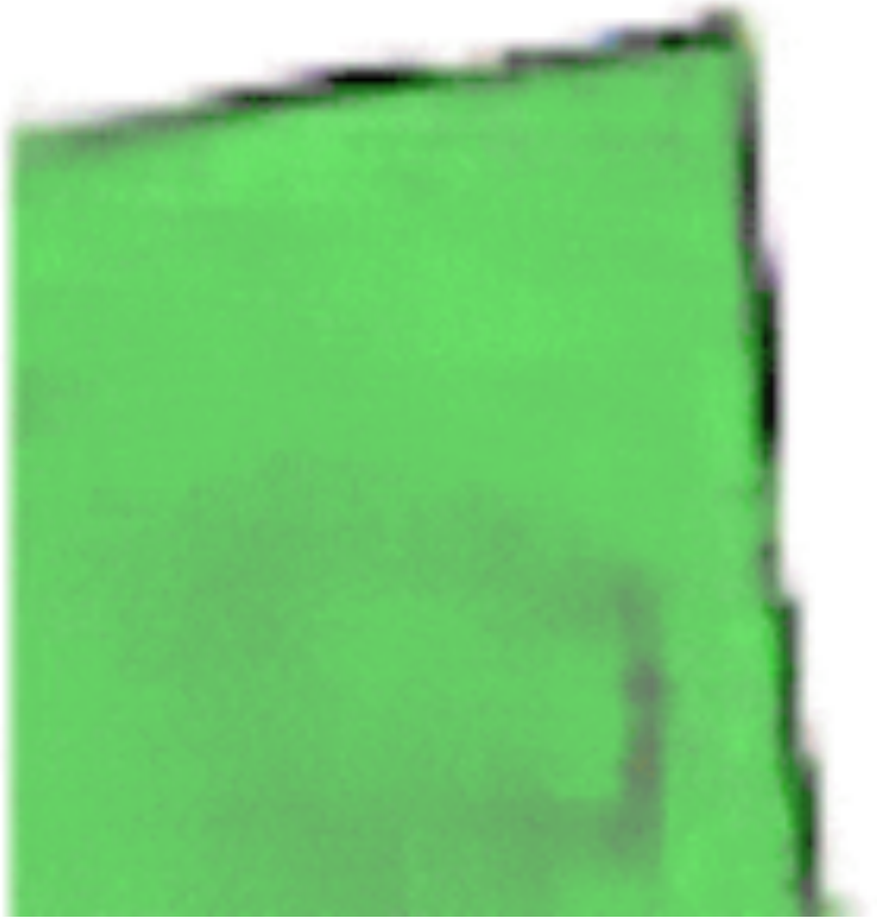
1. Collect 10,000 rollouts from a random policy.
2. Train VAE (V) to encode frames into $z \in \mathcal{R}^{32}$.
3. Train MDN-RNN (M) to model $P(z_{t+1} \mid a_t, z_t, h_t)$.
4. Evolve linear controller (C) to maximize the expected cumulative reward of a rollout. $a_t = W_c \left[z_t \; h_t\right] + b_c$

Model	Parameter Count
VAE	4,348,547
MDN-RNN	422,368
Controller	867

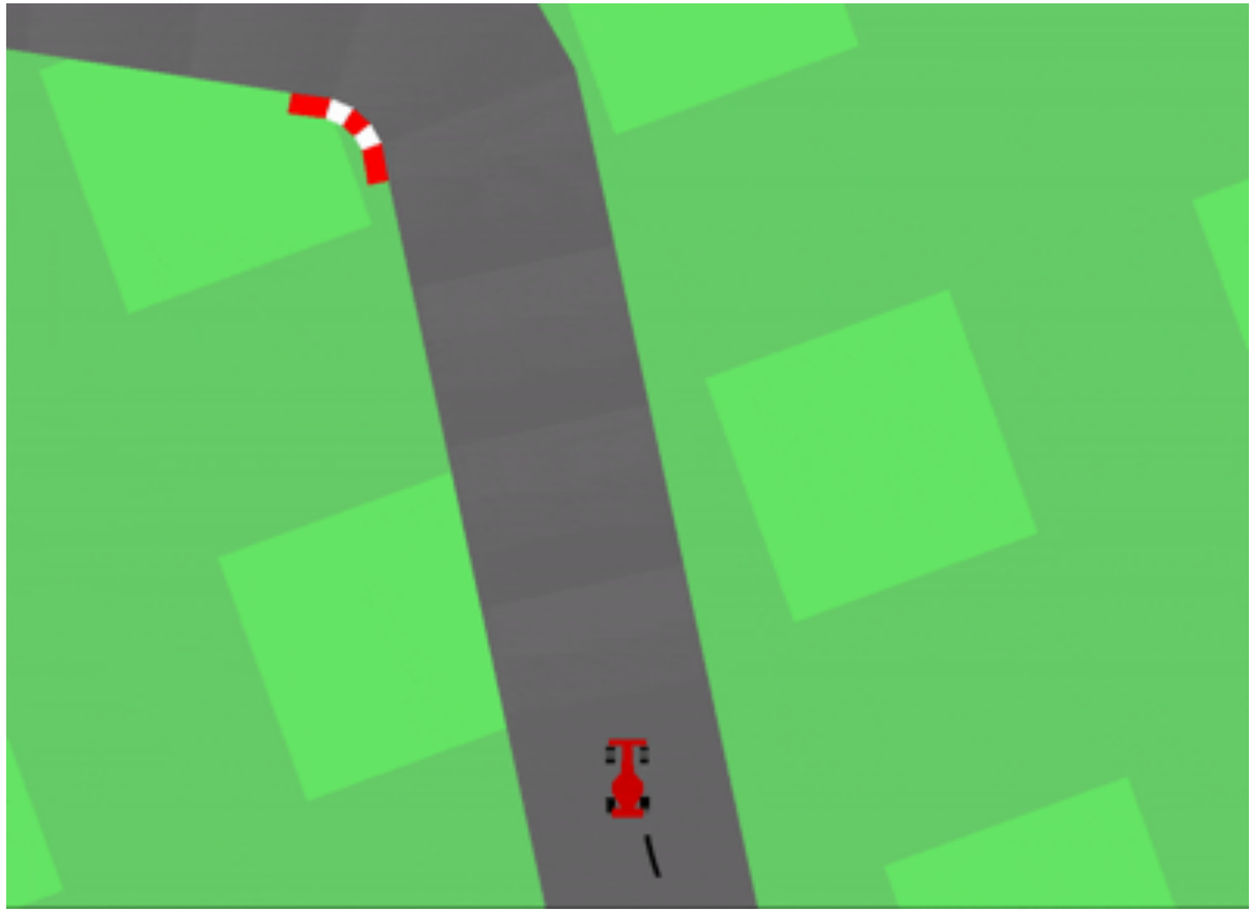
- Randomly generated tracks
- Stay on tiles, travel around track
- Average Score > 900 (100 trials) to “solve” task

Input Frame (64x64px)

Frame Reconstruction using z



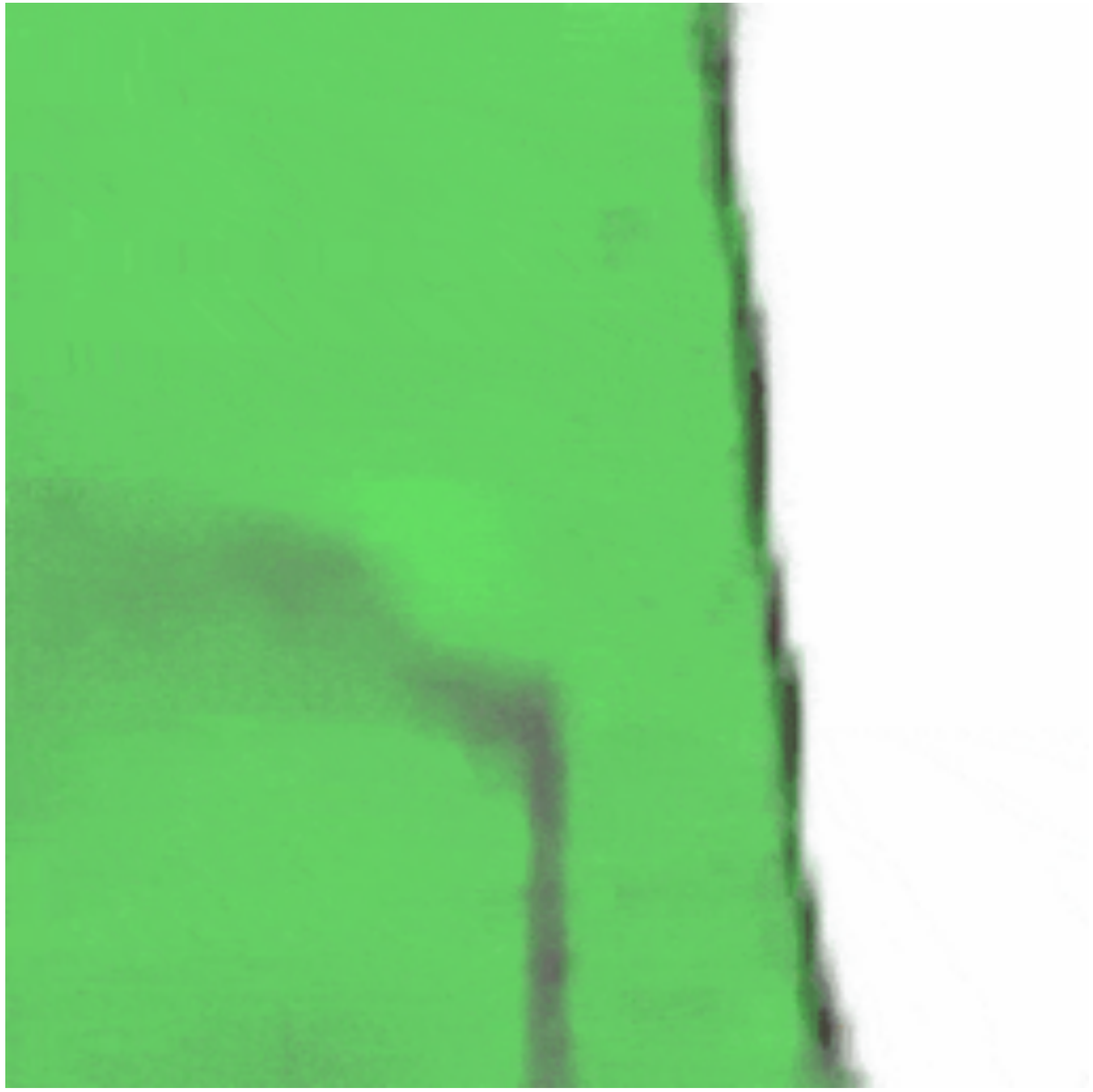
Car Racing: Spatial Inputs vs Spatial and Temporal Inputs



z (spatial) only input

z and h (RNN's hidden state) as inputs

CarRacing-v0 Results

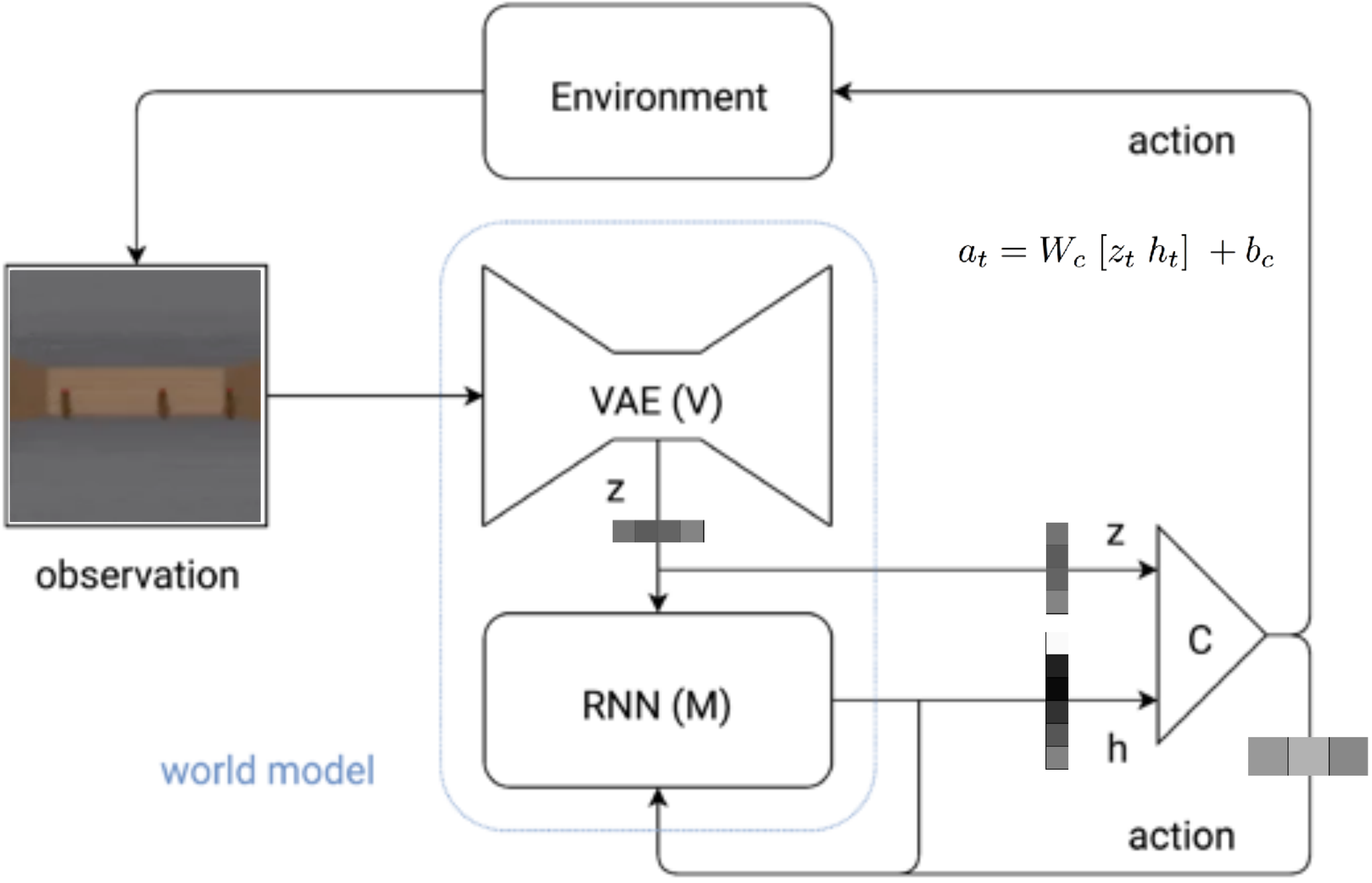


METHOD	AVG. SCORE
DQN (PRIEUR, 2017)	343 ± 18
A3C (CONTINUOUS) (JANG ET AL., 2017)	591 ± 45
A3C (DISCRETE) (KHAN & ELIBOL, 2016)	652 ± 10
CEOBILLIONAIRE (GYM LEADERBOARD)	838 ± 11
V MODEL	632 ± 251
V MODEL WITH HIDDEN LAYER	788 ± 141
FULL WORLD MODEL	$\mathbf{906 \pm 21}$

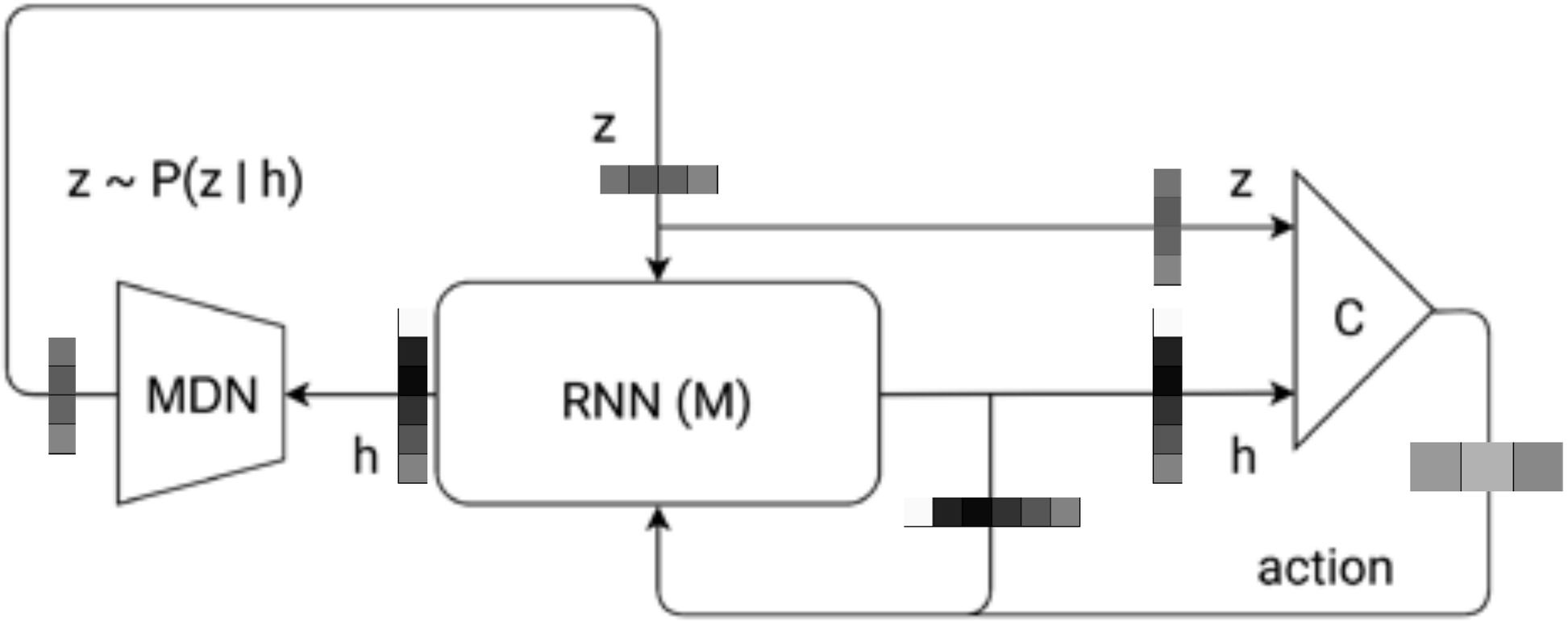
`CarRacing-v0` scores achieved using various methods.

METHOD	AVG. SCORE
Random Weights for RNN M (Tallec et al., 2018)	870 ± 120
3-Layer DQN w/ Dropout, Curriculum Learning (Gerber et al., 2018)	893 ± 42
Train V and M also with Neuroevolution (Risi and Stanley, GECCO 2019)	903 ± 72

`CarRacing-v0` scores achieved using newer methods.


Environment
action
$a_t = W_c \, [z_t \; h_t] \; + b_c$
VAE (V)
z
observation
RNN (M)
world model
z
C
h
action

Train controller inside latent space environment.



DoomTakeCover-v0

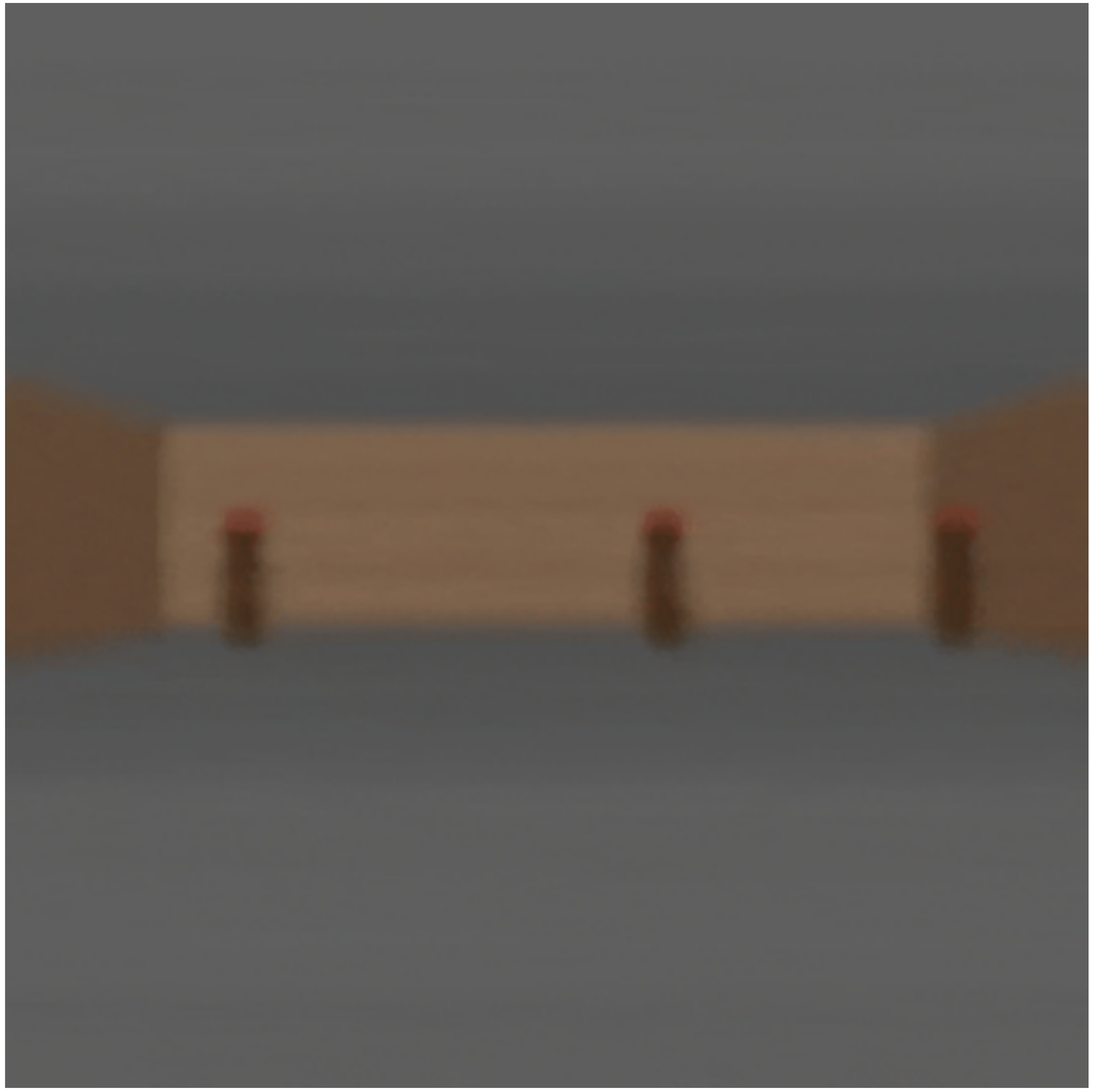
Procedure:

1. Collect 10,000 rollouts from a random policy.
2. Train VAE (V) to encode frames into $z \in \mathcal{R}^{64}$.
3. Train MDN-RNN (M) to model $P(z_{t+1}, d_{t+1} \mid a_t, z_t, h_t)$.
4. Evolve linear controller (C) to maximize the expected cumulative reward of a rollout.
5. Deploy learned policy from (4) on actual environment.

MODEL	PARAMETER COUNT
VAE	4,446,915
MDN-RNN	1,678,785
CONTROLLER	1,088

- Avoid fireballs from monsters
- Stay alive for as long as possible
- Average Score > 750 time steps (100 trials) to “solve” task

Doom TakeCover: Cheating the World Model



Normal Temperature

Higher Temperature

DoomTakeCover-v0 Results

Temperature τ	Virtual Score	Actual Score
0.10	2086 ± 140	193 ± 58
0.50	2060 ± 277	196 ± 50
1.00	1145 ± 690	868 ± 511
1.15	918 ± 546	1092 ± 556
1.30	732 ± 269	753 ± 139
Random Policy	N/A	210 ± 108
Gym Leader	N/A	820 ± 58

`DoomTakeCover-v0` scores at various settings of τ.

- Agent learned actions to take advantage of flaws of virtual environment.
- Adjust temperature parameter in the sampling to control uncertainty.

Iterative Training Policy

Procedure:

1. Initialize M, C with random model parameters.
2. Rollout to actual environment N times. Save all actions a_t and observations x_t during rollouts to storage.
3. Train M to model $P(x_{t+1}, r_{t+1}, a_{t+1}, d_{t+1}|x_t, a_t, h_t)$ and train C to optimize expected rewards inside of M.
4. Go back to (2) if task has not been completed.

Iteration #1

Iterative Training Policy

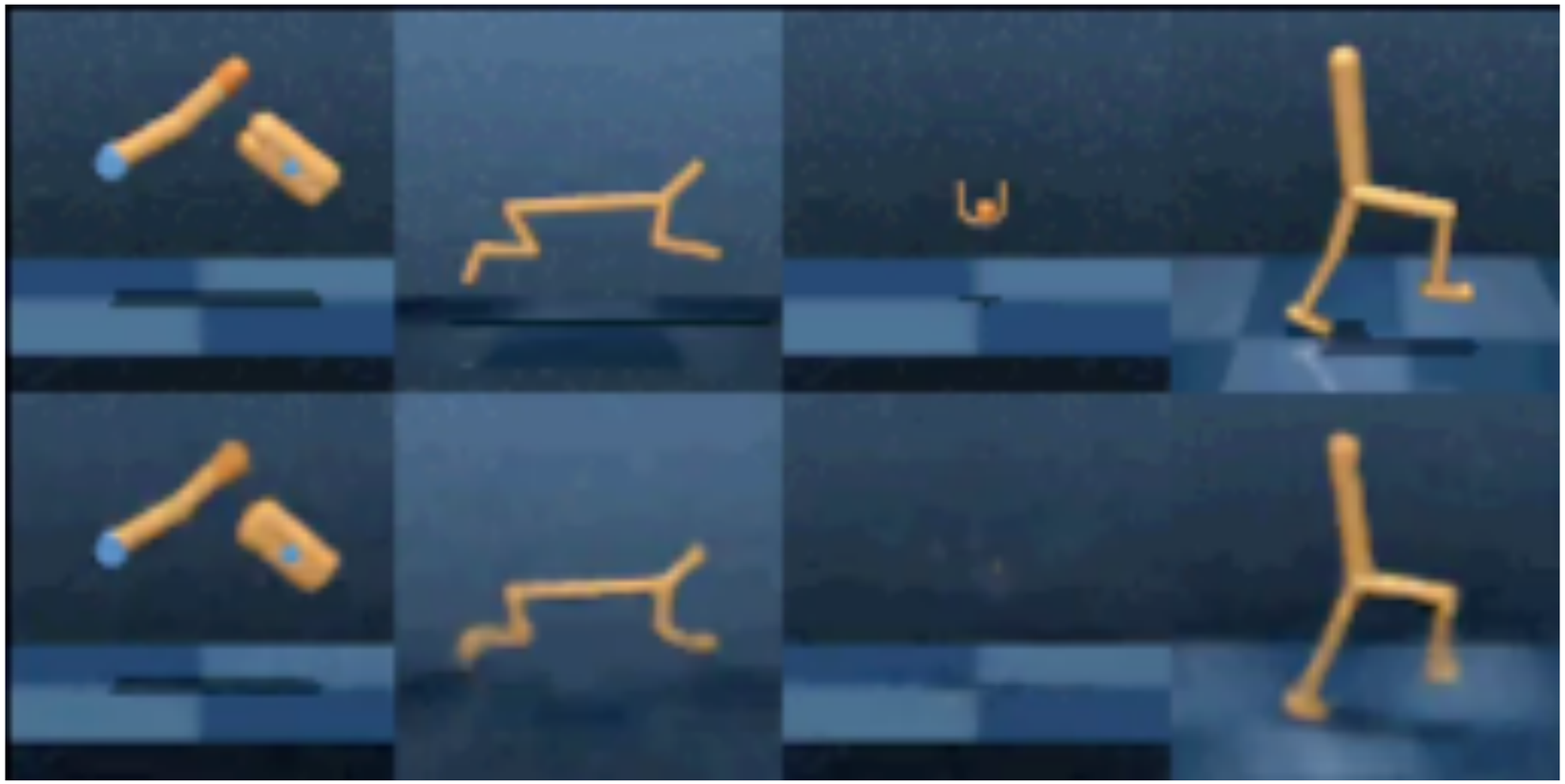
Procedure:

1. Initialize M, C with random model parameters.
2. Rollout to actual environment N times. Save all actions a_t and observations x_t during rollouts to storage.
3. Train M to model $P(x_{t+1}, r_{t+1}, a_{t+1}, d_{t+1}|x_t, a_t, h_t)$ and train C to optimize expected rewards inside of M.
4. Go back to (2) if task has not been completed.

Iteration #20

Learning Latent Dynamics for Planning from PixelsI (ICML 2019)

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee and James Davidson



PlaNet learns a world model from image inputs only and successfully leverages it for planning in latent space. Solves variety of image-based control tasks, competing with SOTA model-free algorithms in terms of final performance while being 50x more sample efficient.

Model-Based Reinforcement Learning for Atari (2019)

Błażej Osiński, Łukasz Kaiser, Mohammad Babaeizadeh, George Tucker, Dumitru Erhan, Ryan Sepassi, Chelsea Finn, Sergey Levine, Piotr Kozakowski, Konrad Czechowski,Piotr Miłos and Henryk Michalewski

Simple iterative training and learning “in dream” achieves state-of-the-art results in sample efficiency for many Atari games.

Neural Network Architectures for Reinforcement Learning Environments

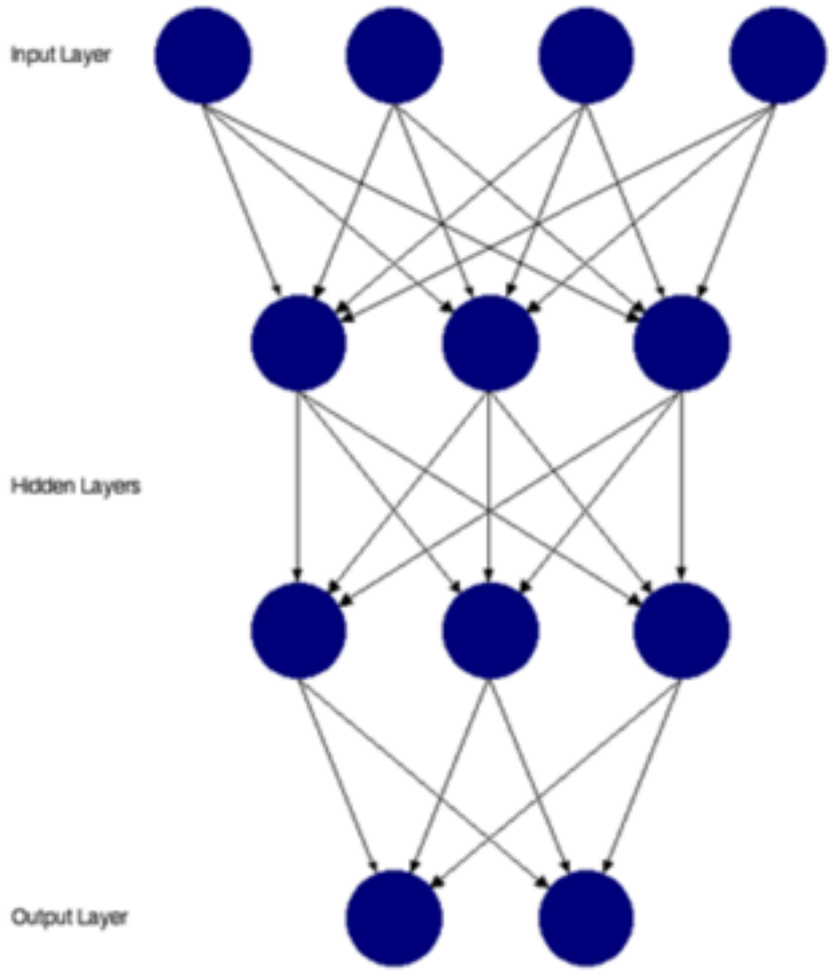
Neural Network evolved to play Mario Kart 64, requiring only small computing power.

Implementation by Nick Nelson (2016)
github.com/nicknlsn/MarioKart64NEAT

Stanley and Miikkulainen,
Efficient Reinforcement Learning Through Evolving Neural Network Topologies (NEAT) (GECCO 2002, Best Paper Award in GA)

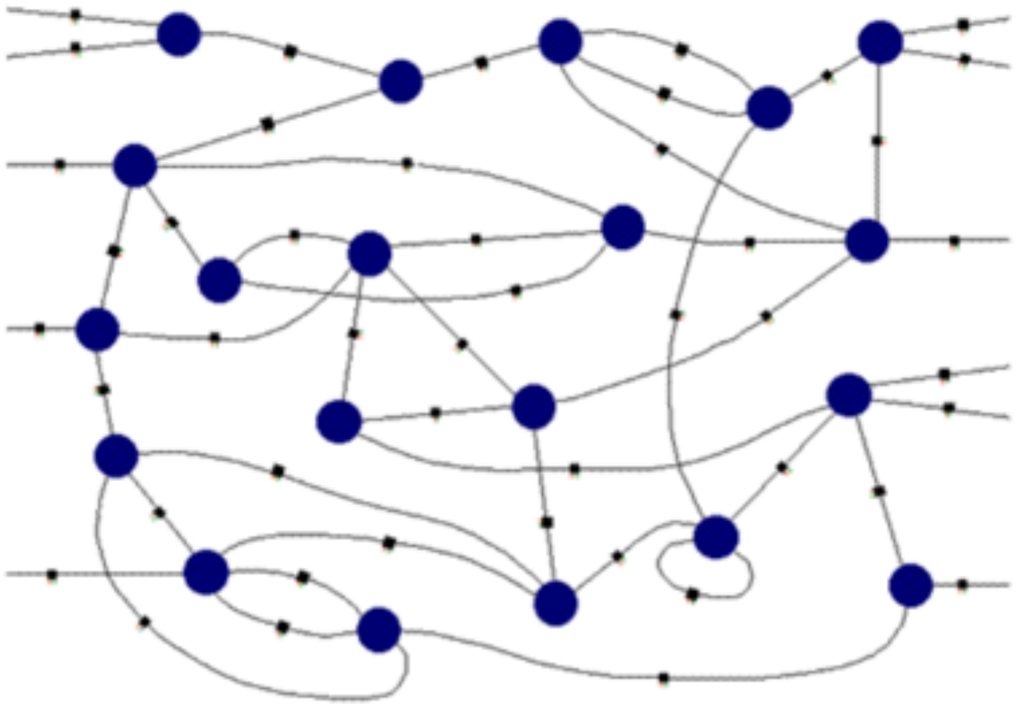
Alan Turing's Unorganized Machines (1948)

alanturing.net



A conventional neural network

In contrast, the neurons in a B-type neural network interconnect freely and a large B-type may be awash with feedback:



Part of a large initially random B-type network

Perhaps Turing's B-types contain lessons for modern connectionists.

Alan Turing, Intelligent Machinery, 1948 (published posthumously a few decades later)

A paper with “NeuroGENESYS” actually got into NIPS back then …

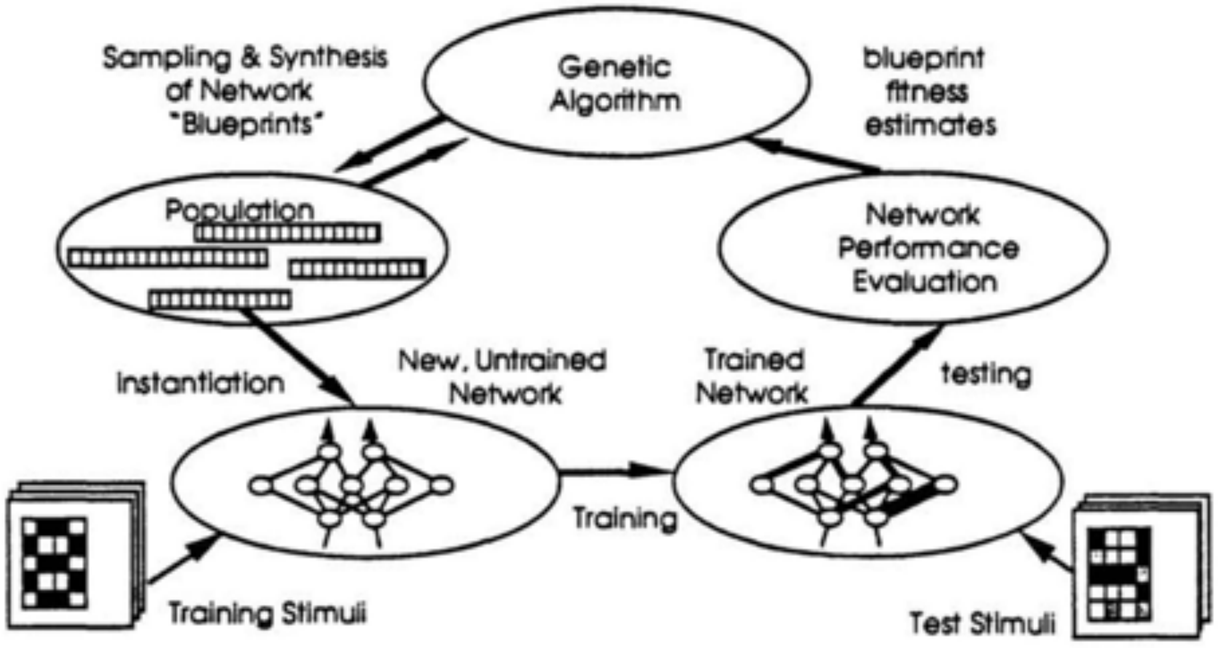
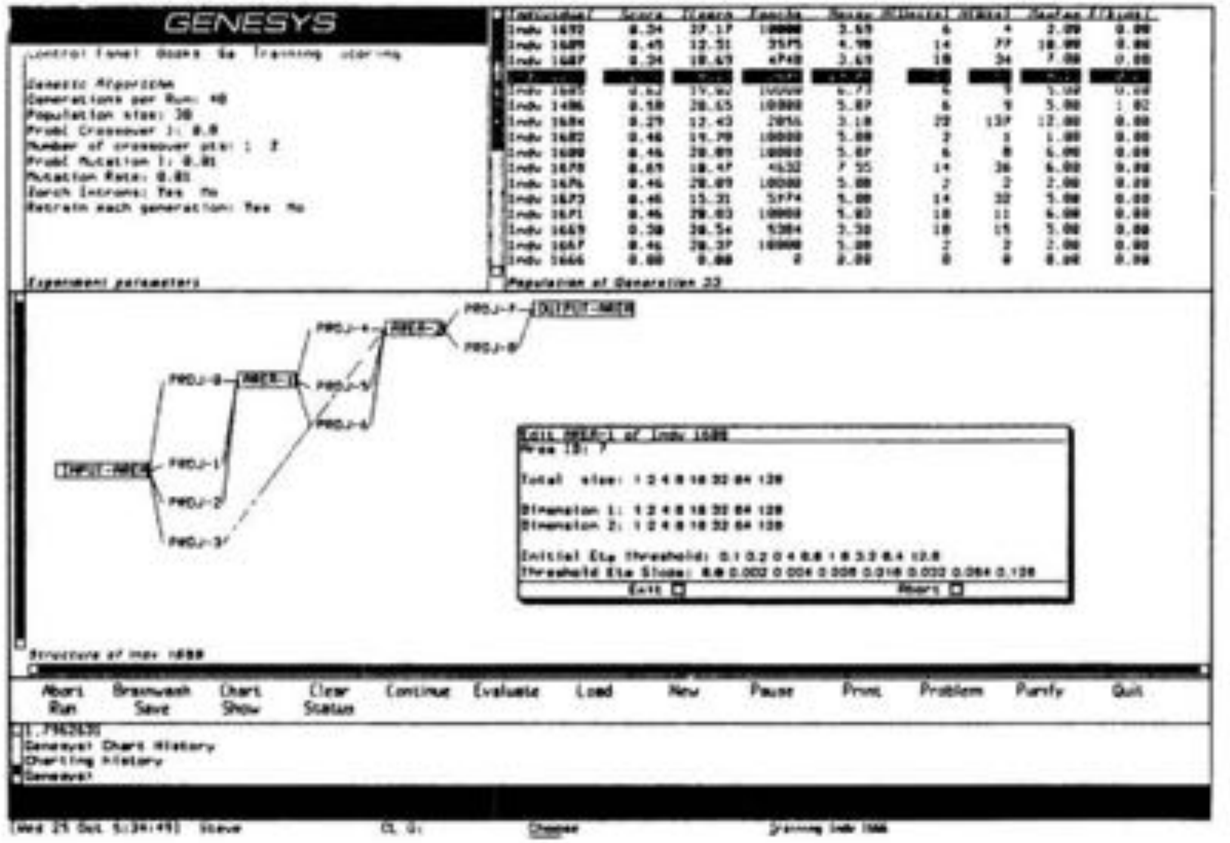
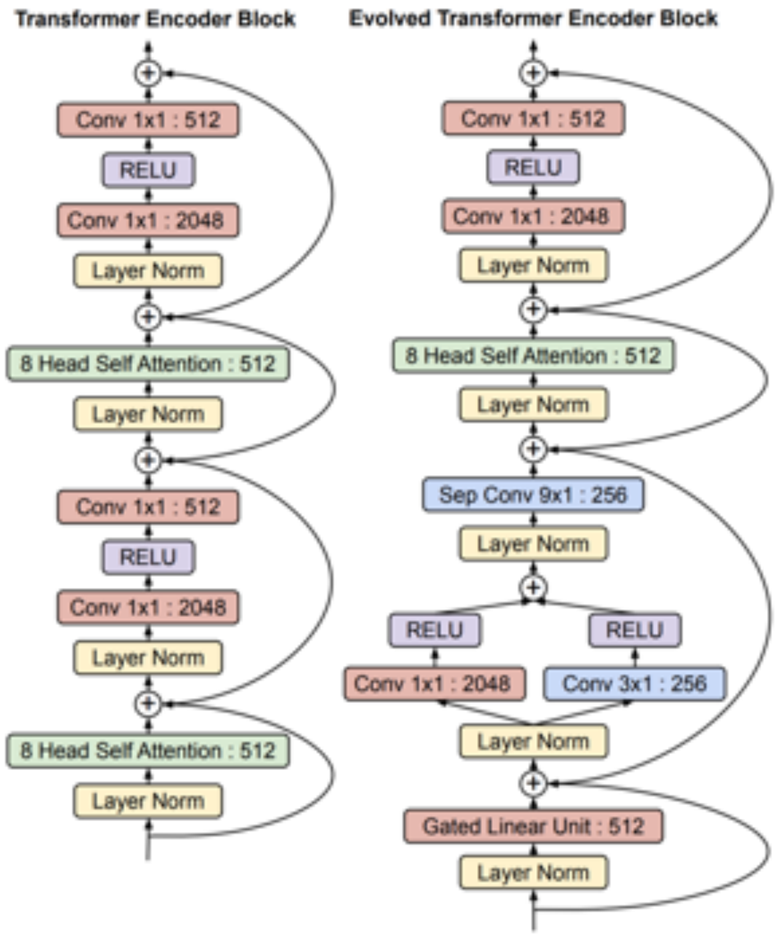


Figure 1: A population of network "blueprints" is cyclically updated by the genetic algorithm based on their fitness.



Harp et al., Designing Application-Specific Neural Networks Using the Genetic Algorithm (NIPS 1989)

Neural Architecture Search (2016-now) use hand-designed components



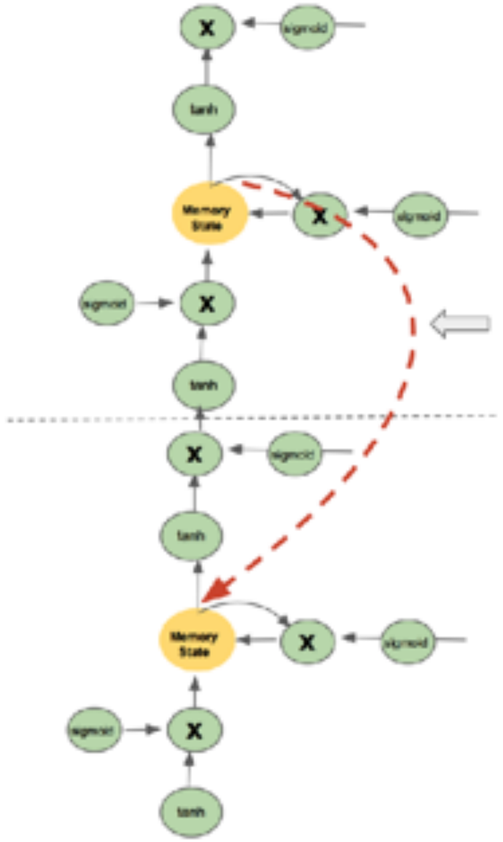


Figure 3: The best-performing LSTM variant after 25 generations of neuroevolution. It includes a novel skip connection between the two memory cells, resulting in 5% improvement over the vanilla LSTM baseline. Such improvements are difficult to discover by hand; CoDeepNEAT with LSTM-specific mutation searches for them automatically.

Zoph and Le, Neural Architecture Search with Reinforcement Learning (ICLR 2017)

So et al., The Evolved Transformer (ICML 2019)

Miikkulainen et al., Evolving Deep Neural Networks (2017)

Random Search (Single-Generation Evolution) gets near SOTA already

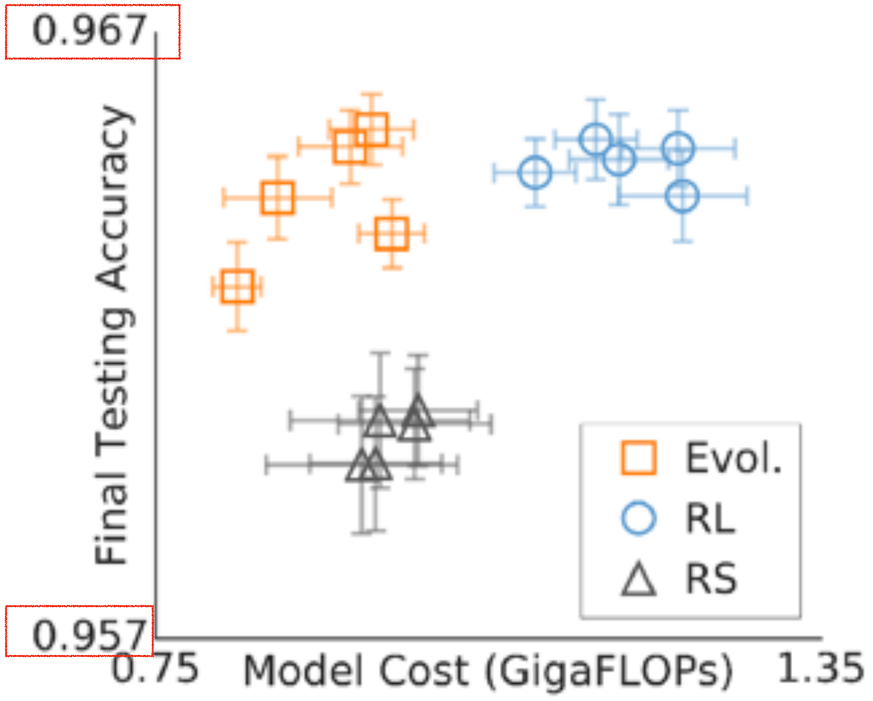


Figure 4: Final augmented models from 5 identical architecture-search experiments for each algorithm, on CIFAR-10. Each marker corresponds to the top models from one experiment.

Architecture	Source	Test Perplexity Valid	Test Perplexity Test	Params (M)	Search Cost Stage 1	Search Cost Stage 2	Search Cost Total	Comparable Search Space?	Search Method
LSTM + DropConnect	[38]	60.0	57.3	24	-	-	-	-	manual
ASHA + LSTM + DropConnect	[30]	58.1	56.3	24	-	-	13	N	HP-tuned
LSTM + MoS	[49]	56.5	54.4	22	-	-	-	-	manual
NAS*	[51]	N/A	64.0	25	-	-	1e4	N	RL
ENAS*†	[41]	N/A	56.3	24	0.5	N/A	N/A	Y	RL
ENAS†	[34]	60.8	58.6	24	0.5	N/A	N/A	Y	random
Random search baseline	[34]	61.8	59.4	23	-	-	2	Y	random
DARTS (first order)	[34]	60.2	57.6	23	0.5	1	1.5	Y	gradient-based
DARTS (second order)	[34]	58.1	55.7	23	1	1	2	Y	gradient-based
DARTS (second order)	Ours	58.2	55.9	23	1	1	2	Y	gradient-based
ASHA baseline	Ours	58.6	56.4	23	-	-	2	Y	random
Random search WS	Ours	57.8	55.5	23	0.25	1	1.25	Y	random

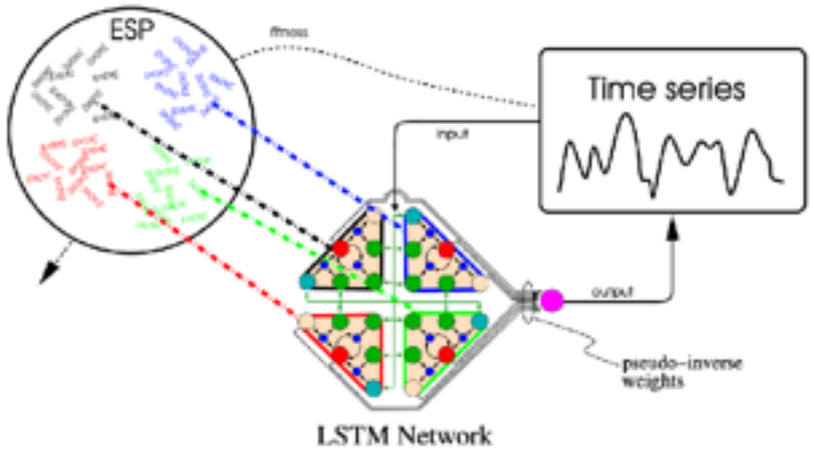
PTB Benchmark: Comparison with state-of-the-art NAS methods and manually designed networks.

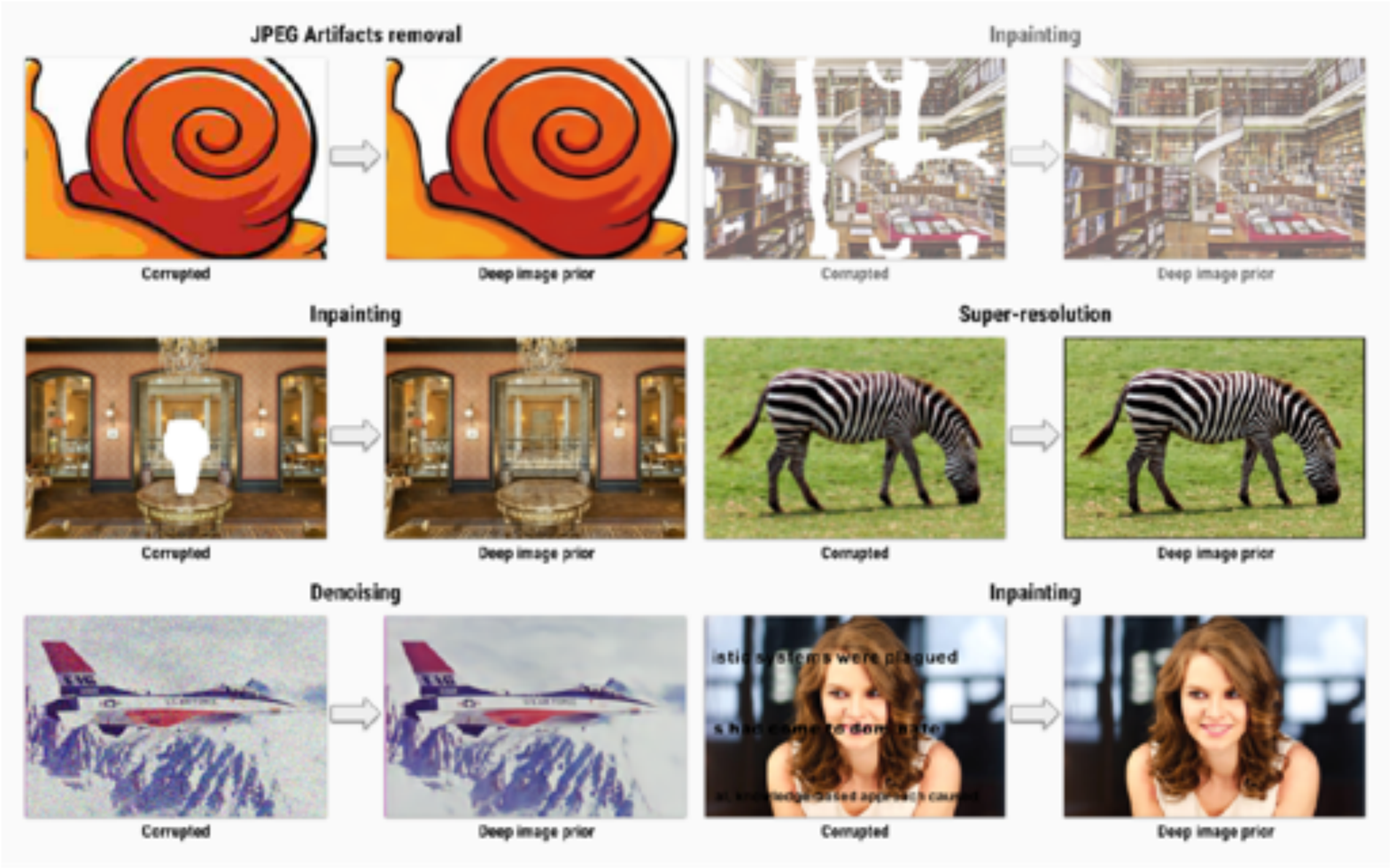
Architecture	Source	Test Error Best	Test Error Average	Params (M)	Search Cost Stage 1	Search Cost Stage 2	Search Cost Total	Comparable Search Space?	Search Method
Shake-Shake#	[9]	N/A	2.56	26.2	-	-	-	-	manual
PyramidNet	[48]	2.31	N/A	26	-	-	-	-	manual
NASNet-A#*	[52]	N/A	2.65	3.3	-	-	2000	N	RL
AmoebaNet-B*	[43]	N/A	2.55 ± 0.05	2.8	-	-	3150	N	evolution
ProxylessNAS†	[7]	2.08	N/A	5.7	4	N/A	N/A	N	gradient-based
GHN#†	[50]	N/A	2.84 ± 0.07	5.7	0.84	N/A	N/A	N	hypernetwork
SNAS†	[47]	N/A	2.85 ± 0.02	2.8	1.5	N/A	N/A	Y	gradient-based
ENAS†	[41]	2.89	N/A	4.6	0.5	N/A	N/A	Y	RL
ENAS	[34]	2.91	N/A	4.2	4	2	6	Y	RL
Random search baseline	[34]	N/A	3.29 ± 0.15	3.2	-	-	4	Y	random
DARTS (first order)	[34]	N/A	3.00 ± 0.14	3.3	1.5	1	2.5	Y	gradient-based
DARTS (second order)	[34]	N/A	2.76 ± 0.09	3.3	4	1	5	Y	gradient-based
DARTS (second order)‡	Ours	2.62	2.78 ± 0.12	3.3	4	6	10	Y	gradient-based
ASHA baseline	Ours	2.85	3.03 ± 0.13	2.2	-	-	9	Y	random
Random search WS‡	Ours	2.71	2.85 ± 0.08	4.3	2.7	6	9.7	Y	random

CIFAR-10 Benchmark: Comparison with state-of-the-art NAS methods and manually designed networks.

Real et al., Regularized Evolution for Image Classifier Architecture Search (2018),
Li and Talwalkar, Random Search and Reproducibility for Neural Architecture Search(2019), Sciuto et al., Evaluating the Search Phase of NAS (2019)

Hand-designed components have great inductive biases





"We show that Evolino-based LSTM can solve tasks that Echo State networks cannot, and achieves higher accuracy in certain continuous function generation tasks than conventional gradient descent RNNs, including gradient-based LSTM."

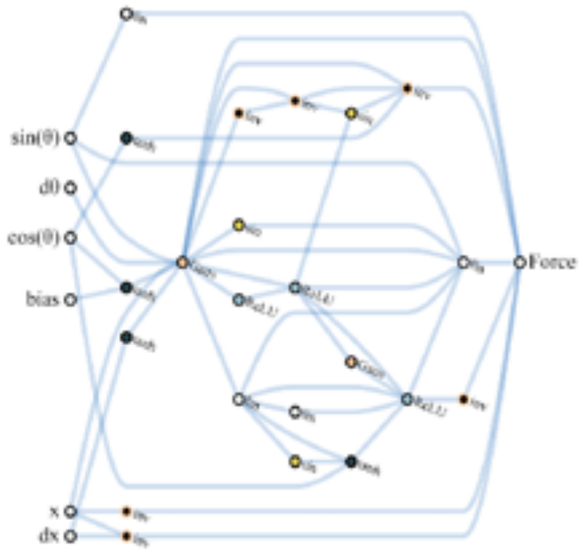
Schmidhuber et al., Training Recurrent Networks by Evolino (Neural Computation, 2007)

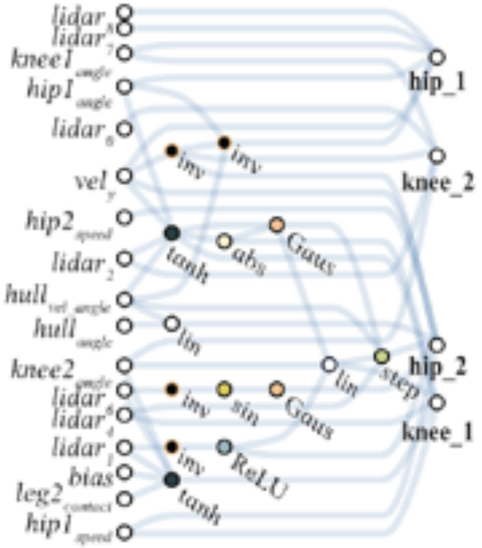
"We show that a randomly-initialized neural network can be used as a handcrafted prior with excellent results in standard inverse problems such as denoising, super-resolution, and inpainting."

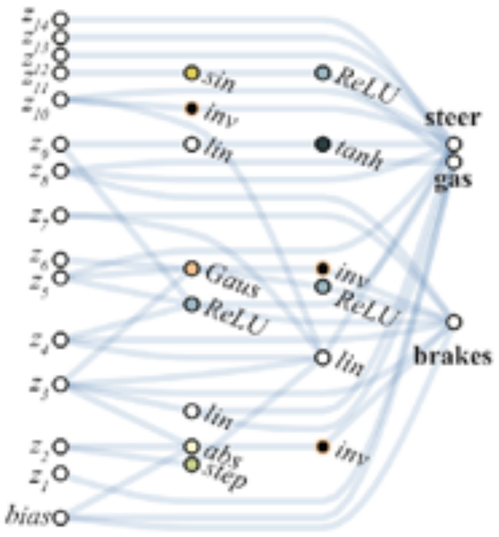
Ulyanov et al., Deep Image Prior (CVPR 2018)

In biology, precocial species are those whose young already possess certain abilities from the moment of birth. (Footage “Iguana vs Snakes”, Planet Earth II, BBC Earth)

Inspired by precocial species in biology, we set out to search for neural net architectures that can already (sort of) perform various tasks even when they use random weight values.



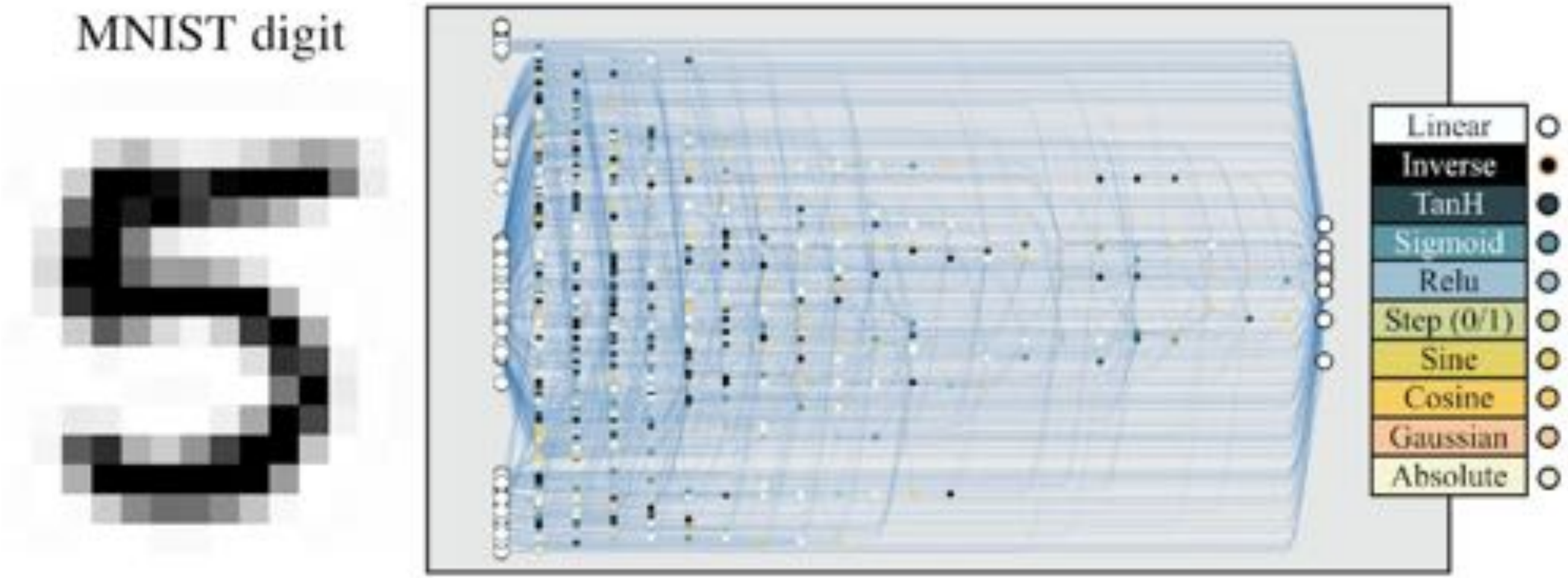




Gaier and Ha, Weight Agnostic Neural Networks (2019)

weightagnostic.github.io

An MNIST classifier evolved to work with random weights



While a conventional network with random initialization will get ~ 10% accuracy on MNIST, this particular network architecture achieves a much better than chance accuracy on MNIST (> 80%) with random weights. Without any weight training, the accuracy increases to > 90% when we use an ensemble of untrained weights.

weightagnostic.github.io

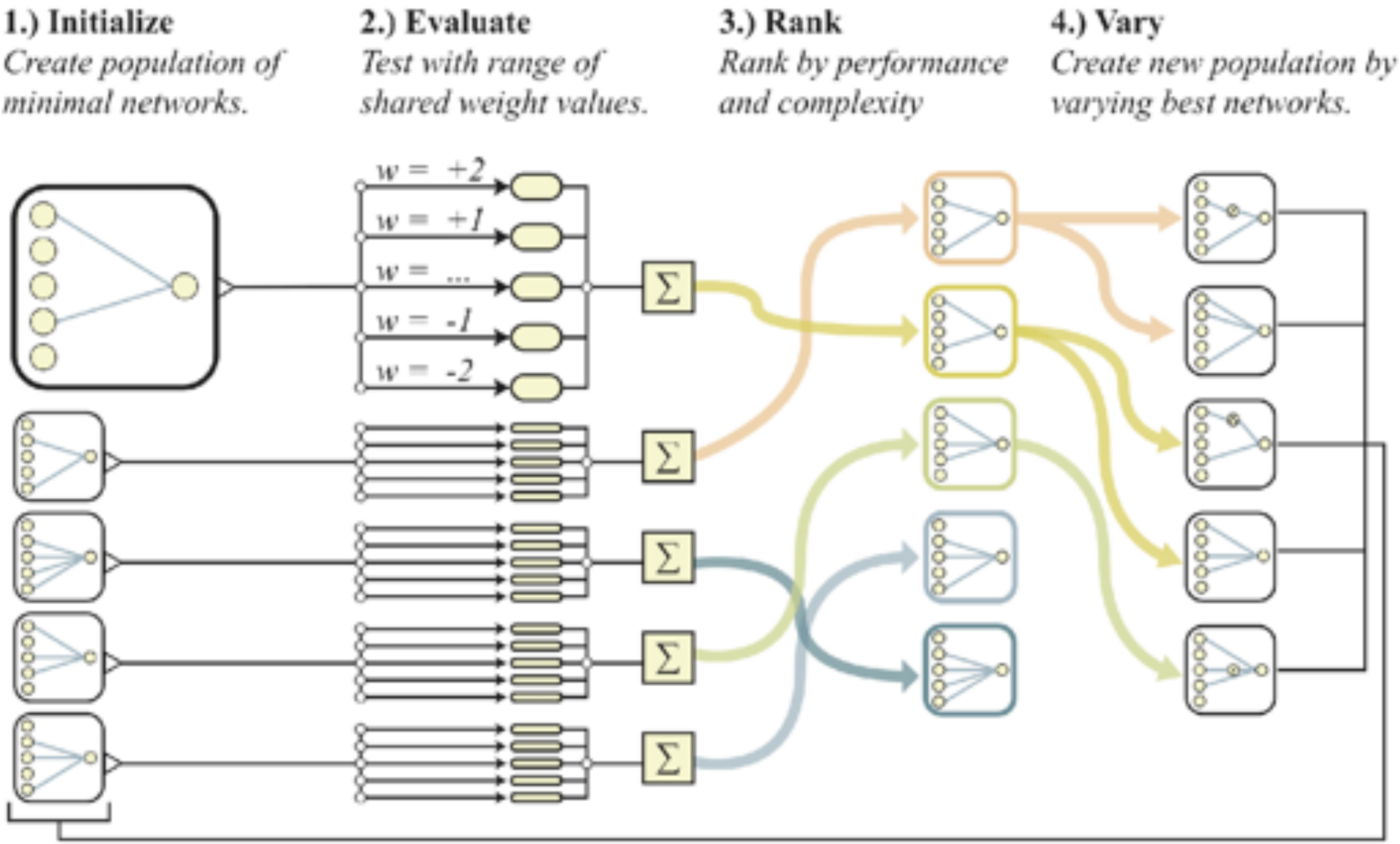
Key idea: search for architectures by de-emphasizing weights

During the search, networks are assigned a single shared weight value at each rollout, and optimized to perform well over a wide range of weight values.

As a bonus, we get to bypass the costly inner training loop!

weightagnostic.github.io

Use NEAT and NSGA-II to find effective, minimal networks that work on a wide range of weights

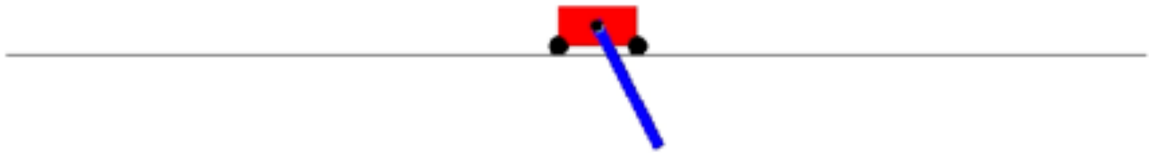


weightagnostic.github.io

Cartpole Swingup WANN outperforms fully-connected network

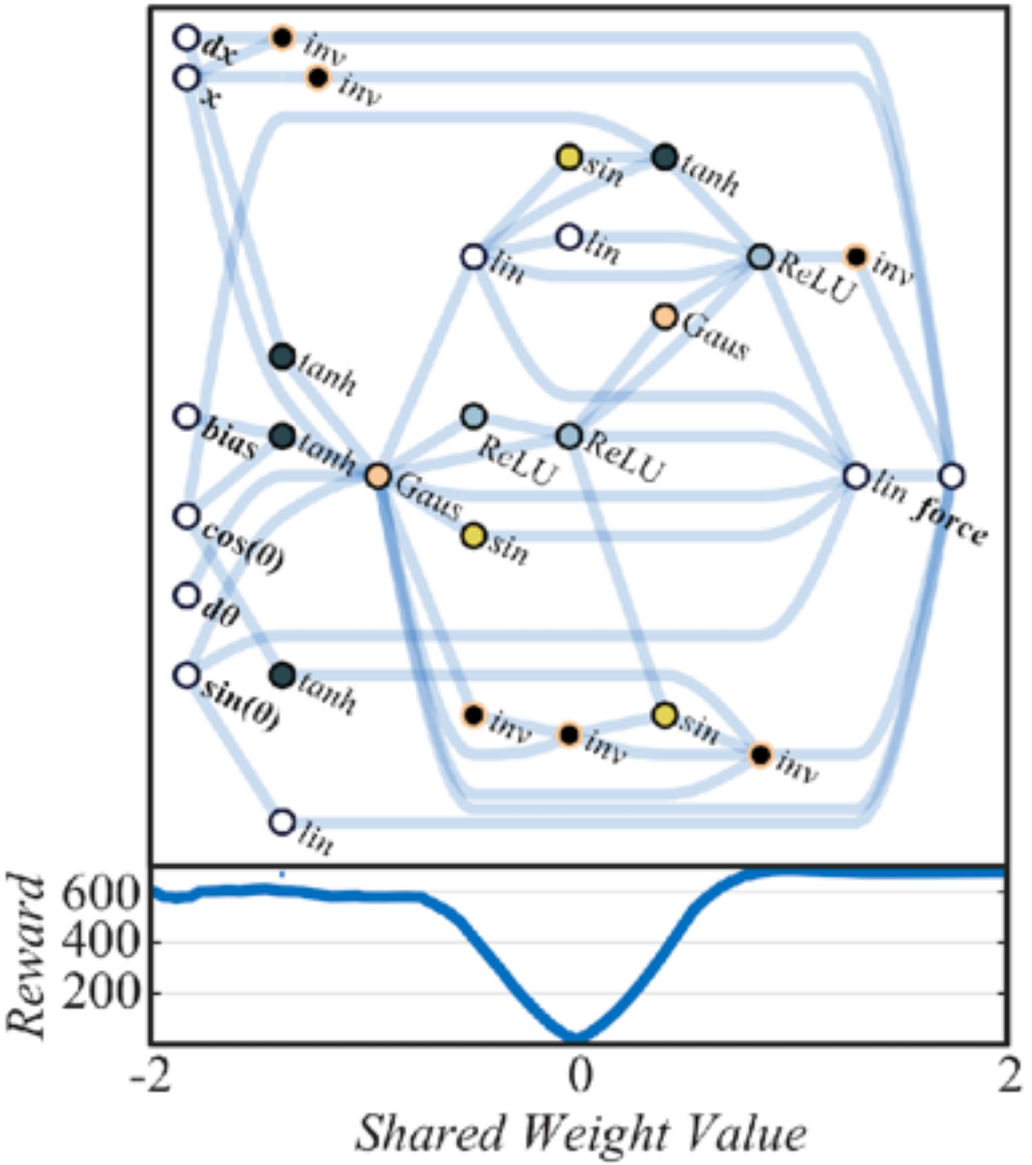
Weight set to -1.5

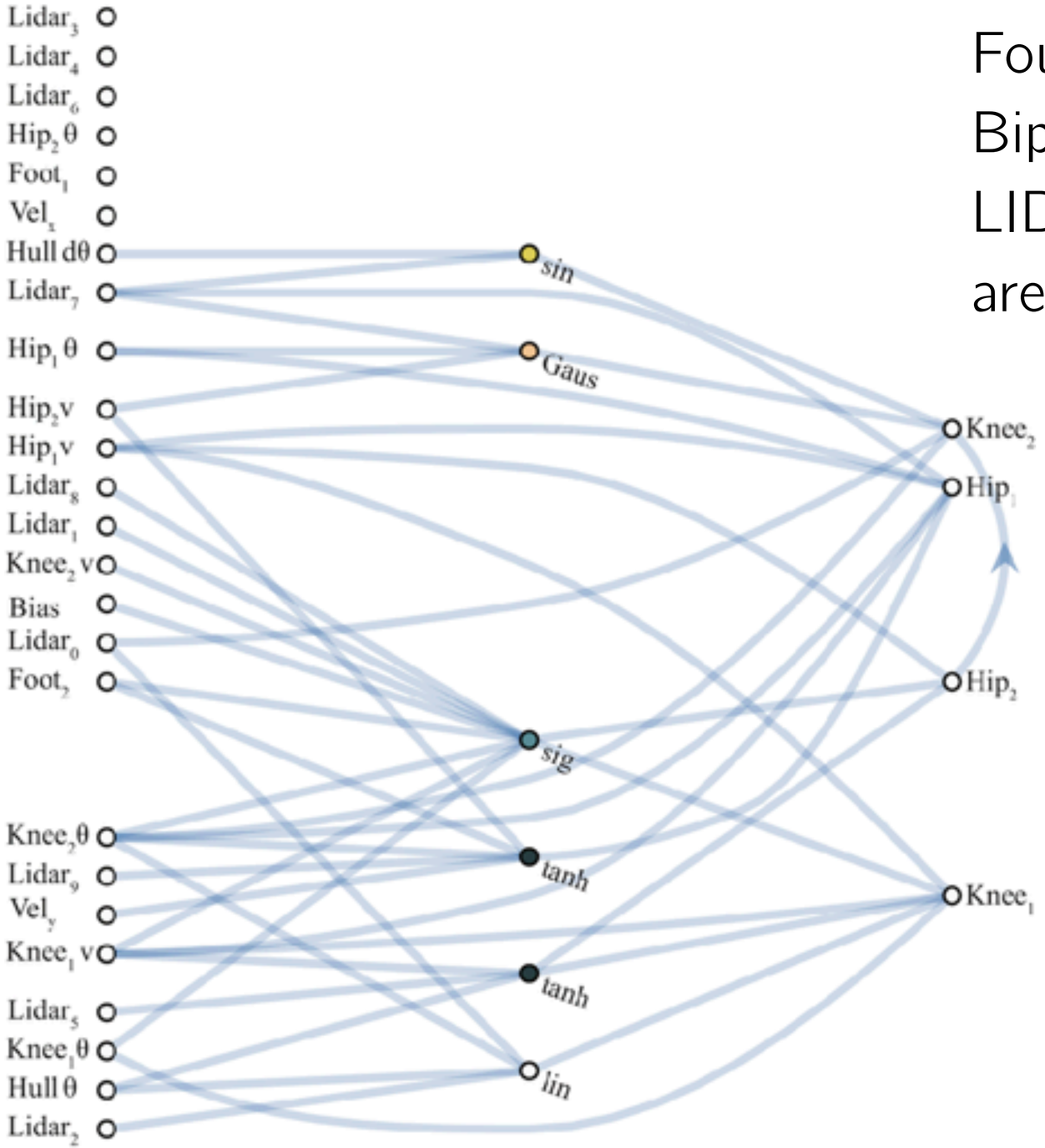
Weight set to +1.0



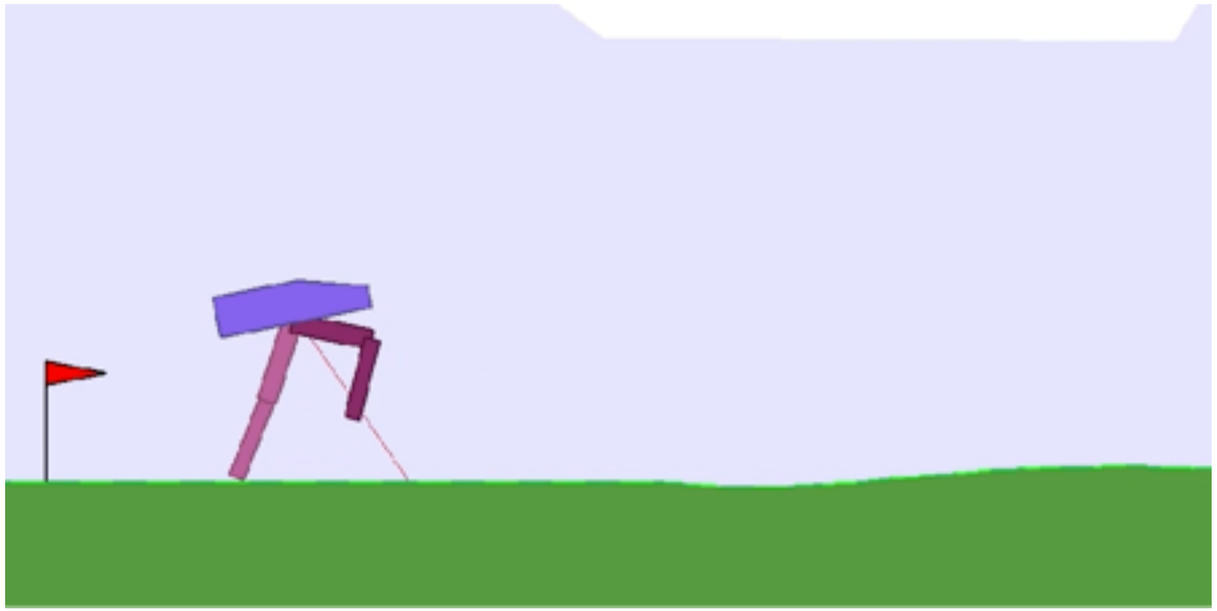
Fine-tuned Weights

Average score 932 ± 6 (917 ± 7 for baseline)



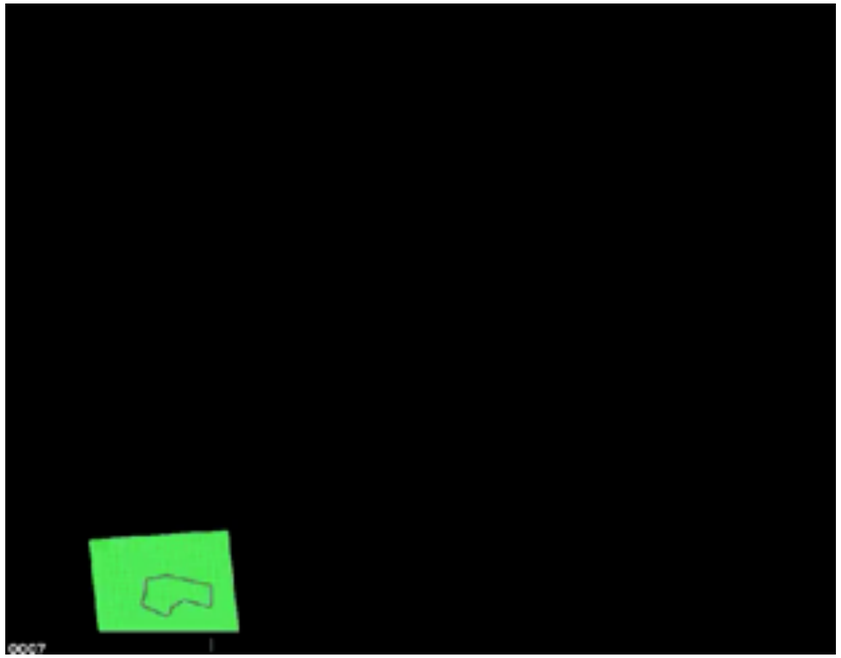


Found simple and elegant WANN for the Bipedal Walker that notably ignores many LIDAR, angle, and other input signals that are not required for the task.

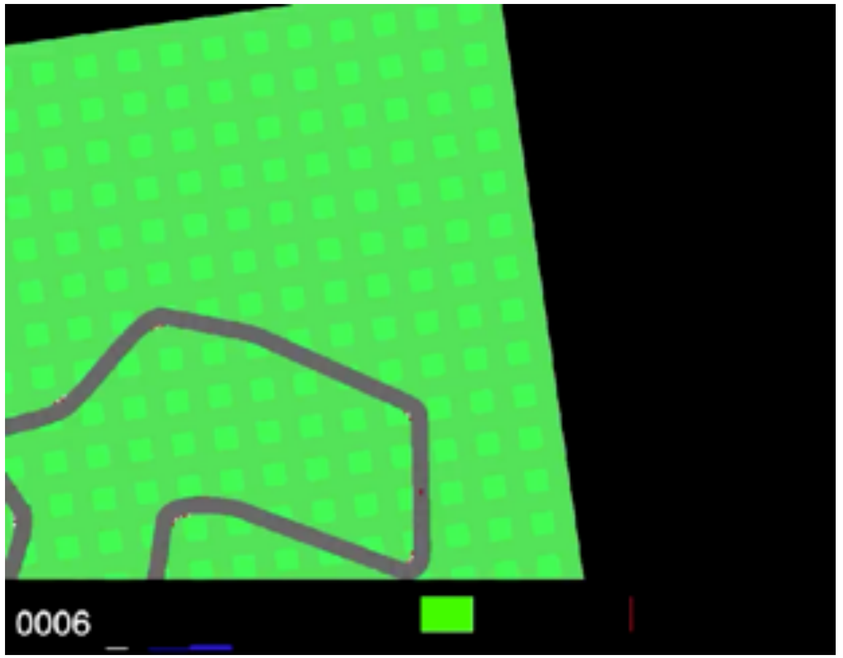


weightagnostic.github.io

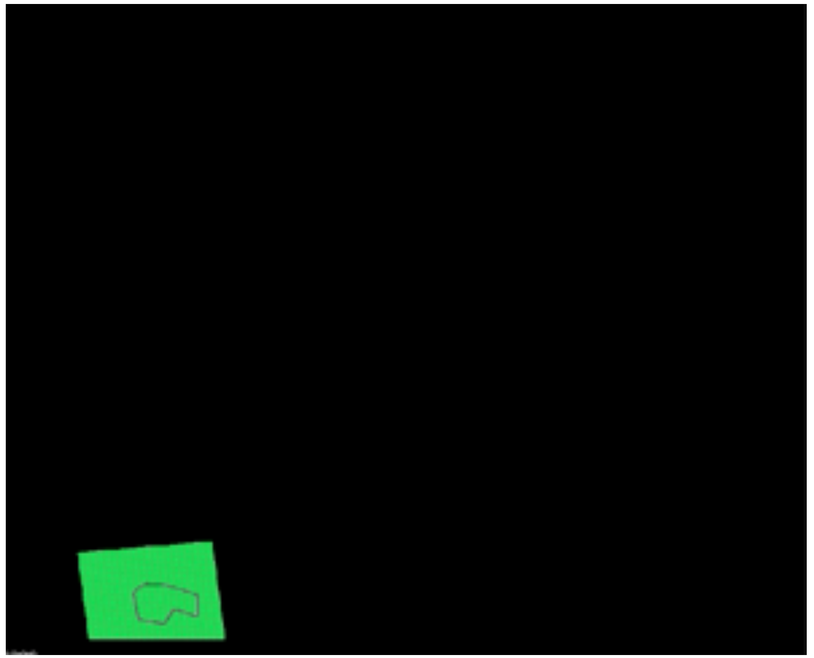
Found a feed-forward controller that almost solved CarRacing-v0



Weight set to -1.4



Weight set to +1.0



Fine-tuned Weights
(Average score 893 ± 74)

weightagnostic.github.io

Performance of Randomly Sampled and Trained Weights for Continuous Control Tasks

Swing Up	Random Shared Weight	Tuned Shared Weight	Tuned Weights
WANN	**515 ± 58**	**723 ± 16**	**932 ± 6**
Fixed Topology	7 ± 2	8 ± 1	918 ± 7

Biped	Random Shared Weight	Tuned Shared Weight	Tuned Weights
WANN	**51 ± 108**	**261 ± 58**	332 ± 1
Fixed Topology	-107 ± 12	-35 ± 23	**347 ± 1**

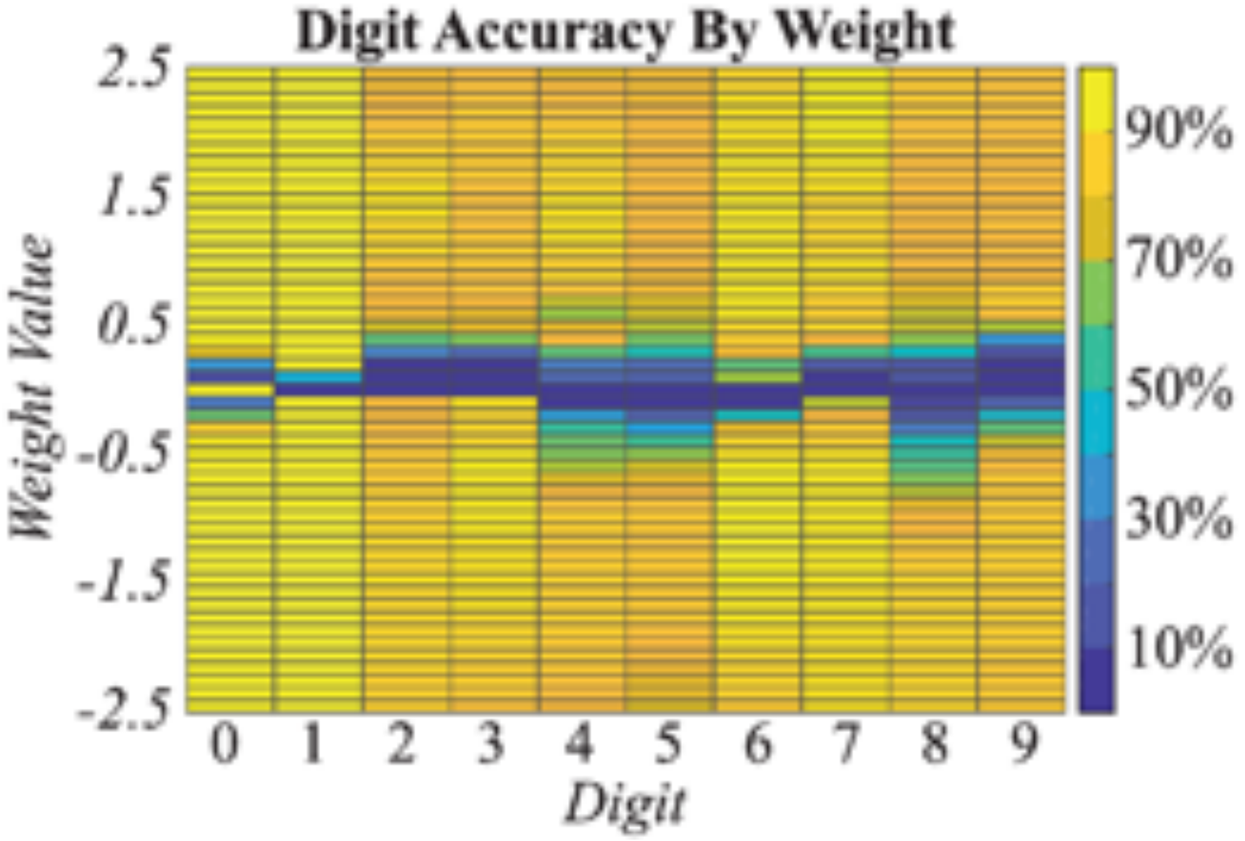
CarRacing	Random Shared Weight	Tuned Shared Weight	Tuned Weights
WANN	**375 ± 177**	**608 ± 161**	893 ± 74
Fixed Topology	-85 ± 27	-37 ± 36	**906 ± 21**

Since the networks are optimized to perform well using a single weight parameter over a range of values, this single parameter can easily be tuned to increase performance. Individual weight values can then be further tuned as offsets from the best shared weight.

WANN ensembles perform far better than when weights are sampled at random, and as well as a linear classifier with thousands of weights.

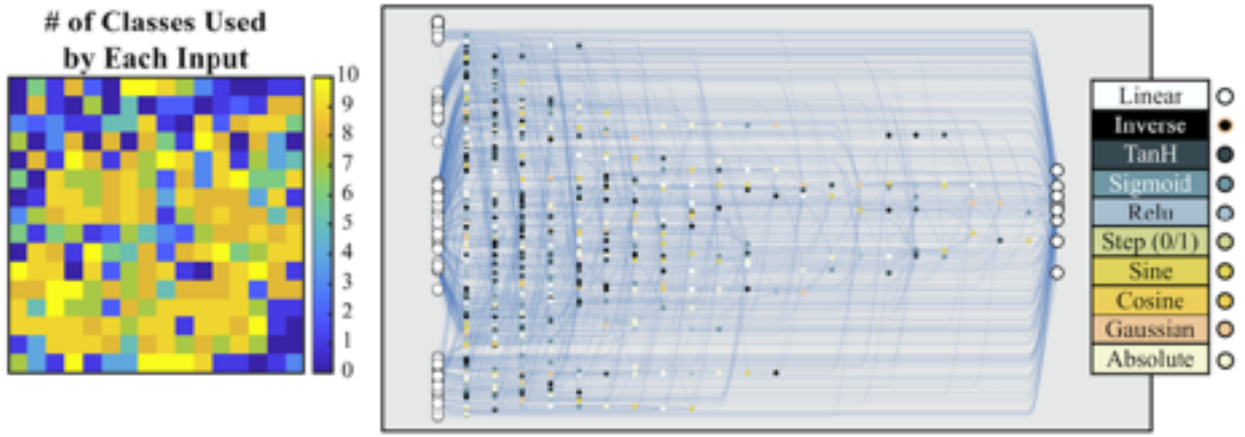
WANN	Test Accuracy
Random Weight	*82.0% ± 18.7%*
Ensemble Weights	*91.6%*
Tuned Weight	*91.9%*
Trained Weights	*94.2%*

ANN	Test Accuracy
Linear Regression	*91.6%* [57]
Two-Layer CNN	*99.3%* [14]

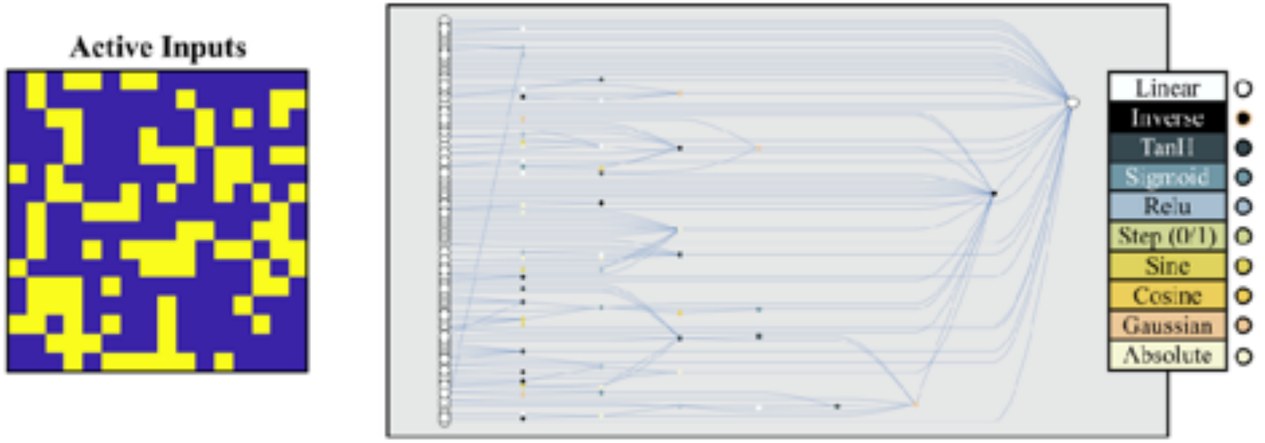


No single weight value has better accuracy on all digits. That WANNs can be instantiated as several different networks has intriguing possibilities for the creation of ensembles.

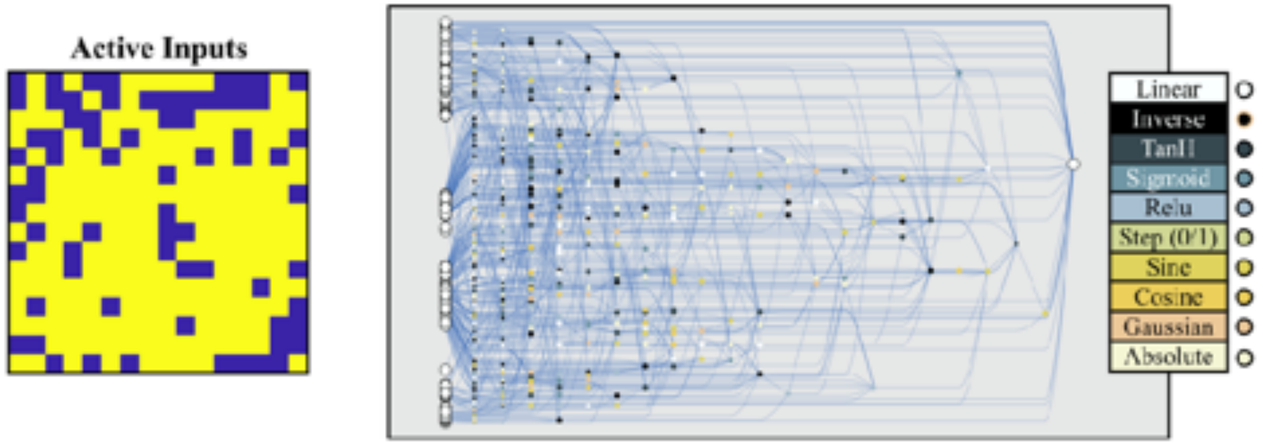
All digits:



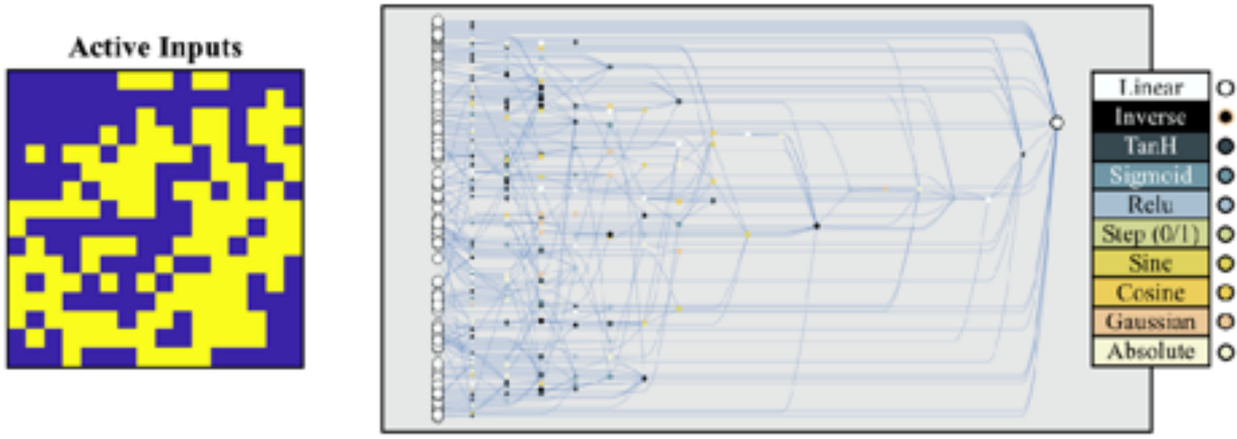
0:



1:



2:



Compute available have grown significantly since the time convolutional neural networks were discovered.

If we are devoting such resources to automated discovery, we should also be searching for new building blocks, not just their arrangements.