Investigating Recurrent Neural Network Memory Structures using Neuro-Evolution

Travis Desell (tjdvse@rit.edu)
Associate Professor
Department of Software Engineering

Co-Authors:
AbdElRahman ElSaid (PhD GRA)
Alex Ororbia (Assistant Professor, RIT Computer Science)

Collaborators:
Steven Benson, Shuchita Patwardhan, David Stadem (Microbeam Technologies, Inc.)
James Higgins, Mark Dusenbury, Brandon Wild (University of North Dakota)

R·I·T
ROCHESTER INSTITUTE OF TECHNOLOGY

Overview

- What is Neuro-Evolution?
- Background:
  - Recurrent Neural Networks for Time Series Prediction
  - Recurrent Memory Cells
- EXAMM:
  - NEAT Innovations
  - Edge and Node Mutations
  - Crossover
- Distributed Neuro-Evolution
- Results
- Future Work
- Discussion

Motivation

Neuro-Evolution

- Applying evolutionary strategies to artificial neural networks (ANNs):
  - **EAs to train ANNs (weight selection)**
  - **EAs to design ANNs (what architecture is best?)**
  - Hyperparameter optimization (what parameters do we use for our backpropagation algorithm)

Neuro-Evolution for Recurrent Neural Networks

- Most people use human-designed ANNs, selecting from a few architectures that have done well in the literature.
- No guarantees these are most optimal (e.g., in size, predictive ability, generalizability, robustness, etc).
- Recurrent edges can go back farther in time than the previous time step -- dramatically increases the search space for RNN architectures.
- With so many memory cell structures and architectures to choose from, which are best? What cells and architectures perform best, does this change across data sets and why?

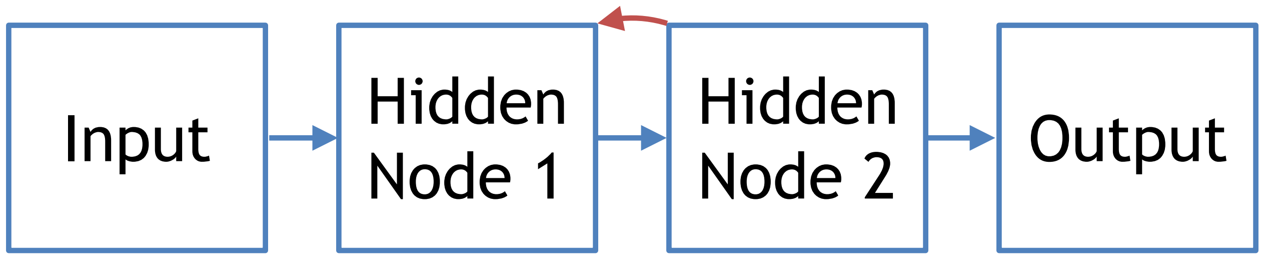
Neuro-Evolution

- Applying evolutionary strategies to artificial neural networks (ANNs):
  - EAs to train ANNs (weight selection)
  - EAs to design ANNs (what architecture is best?)
  - Hyperparameter optimization (what parameters do we use for our backpropagation algorithm)

  - **Using NE to better understand and guide ML: what structures and architectures are evolutionarily selected and why?**

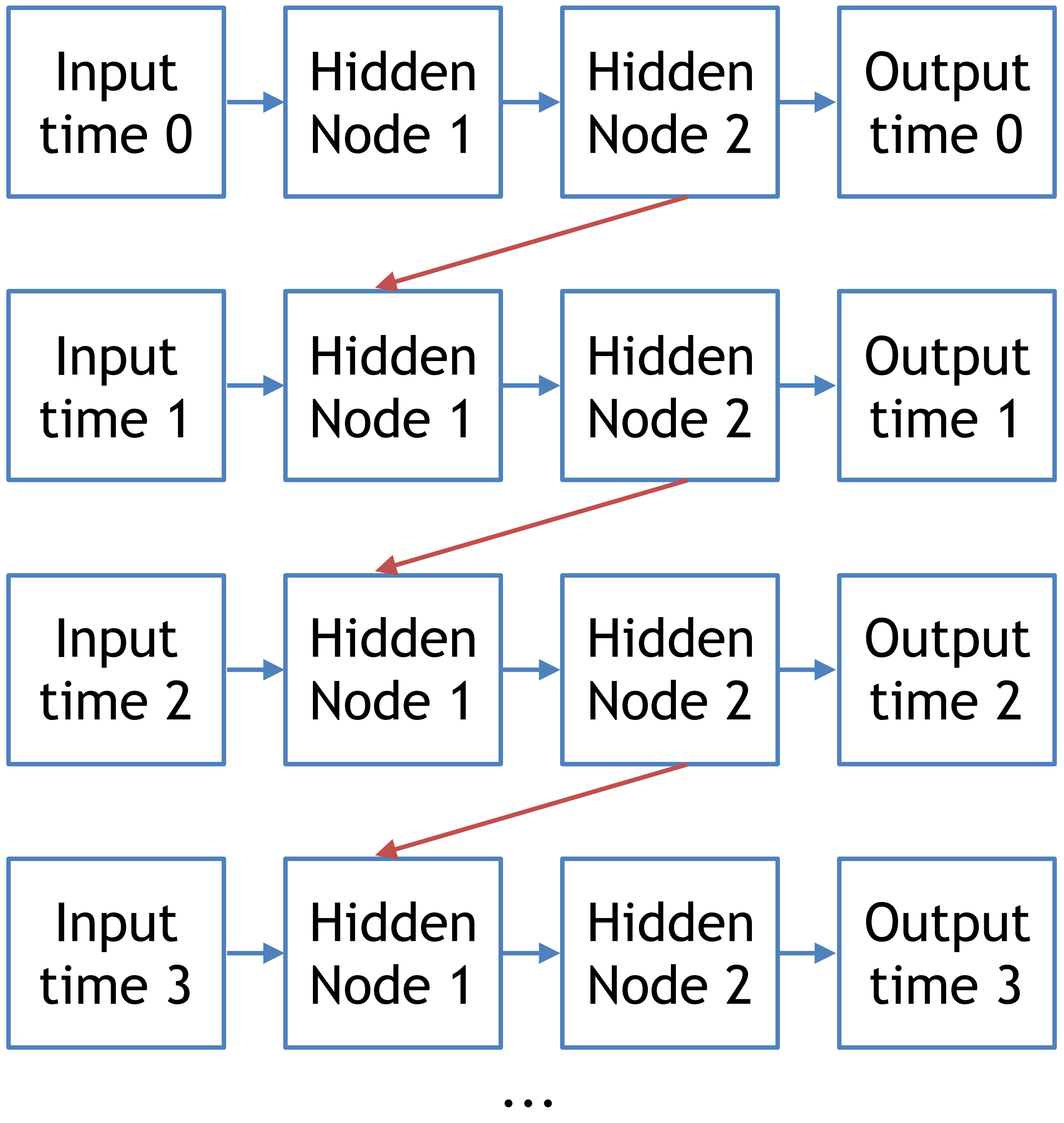
Background

Recurrent Neural Networks

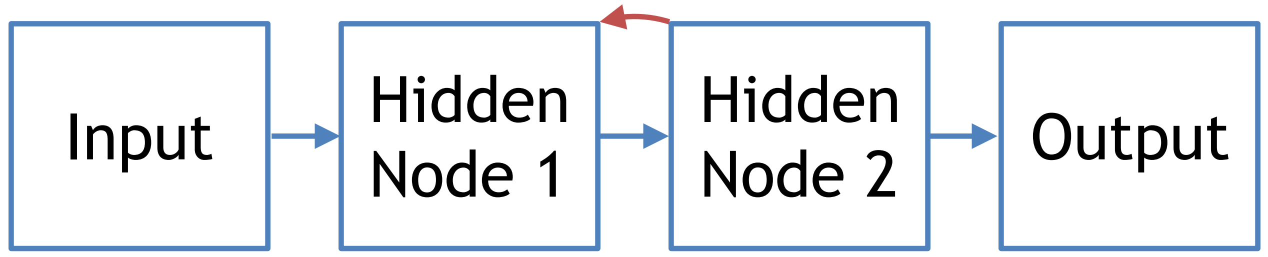


Recurrent Neural Networks can be extremely challenging to train due to the *exploding/ vanishing gradients problem*. In short, when training a RNN over a time series (via backpropagation through time), it needs to be completely unrolled over the time series.

For the simple example above (blue arrows are forward connections, red are recurrent), backpropagating the error from time 3 reaches all the way back to input at time 0 (right). Even with this extremely simple RNN, we end up having an **extremely deep network** to train.

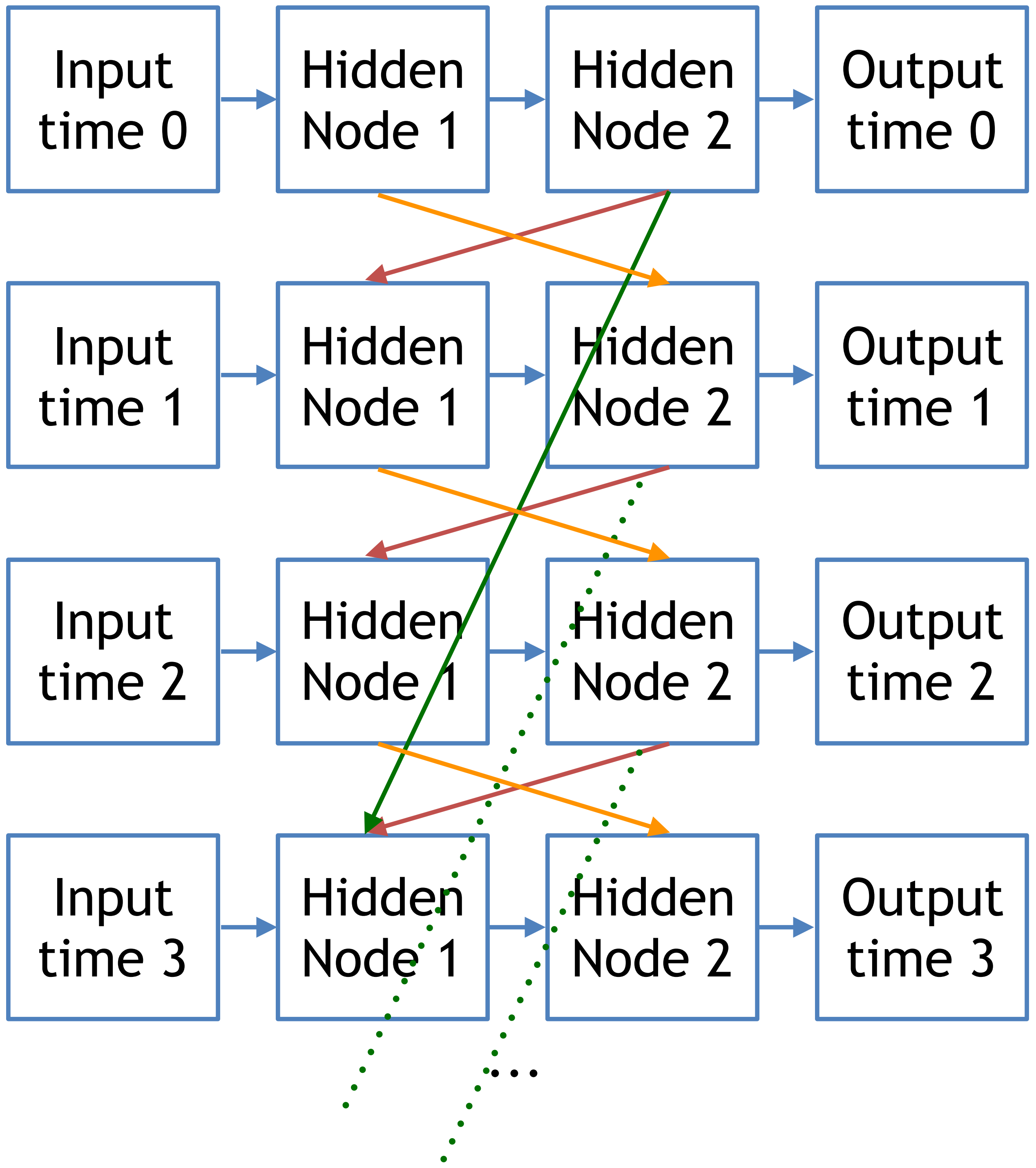


Recurrent Neural Networks



Traditionally, RNN connections go back a single time step and to the same or a previous node in the network.

This is not a requirement - they can go back multiple time steps (green) or forward in the network (orange). However it is not as well studied due to additional complexity and dramatically increasing the architectural search space.



Classification vs. Time Series Data Prediction

RNNs are perhaps more commonly used for classification (and have been mixed with CNNs for image identification). This involves outputs being fed through a softmax layer which results in probabilities for the input being a particular class. The error minimized is for the output being an incorrect class:

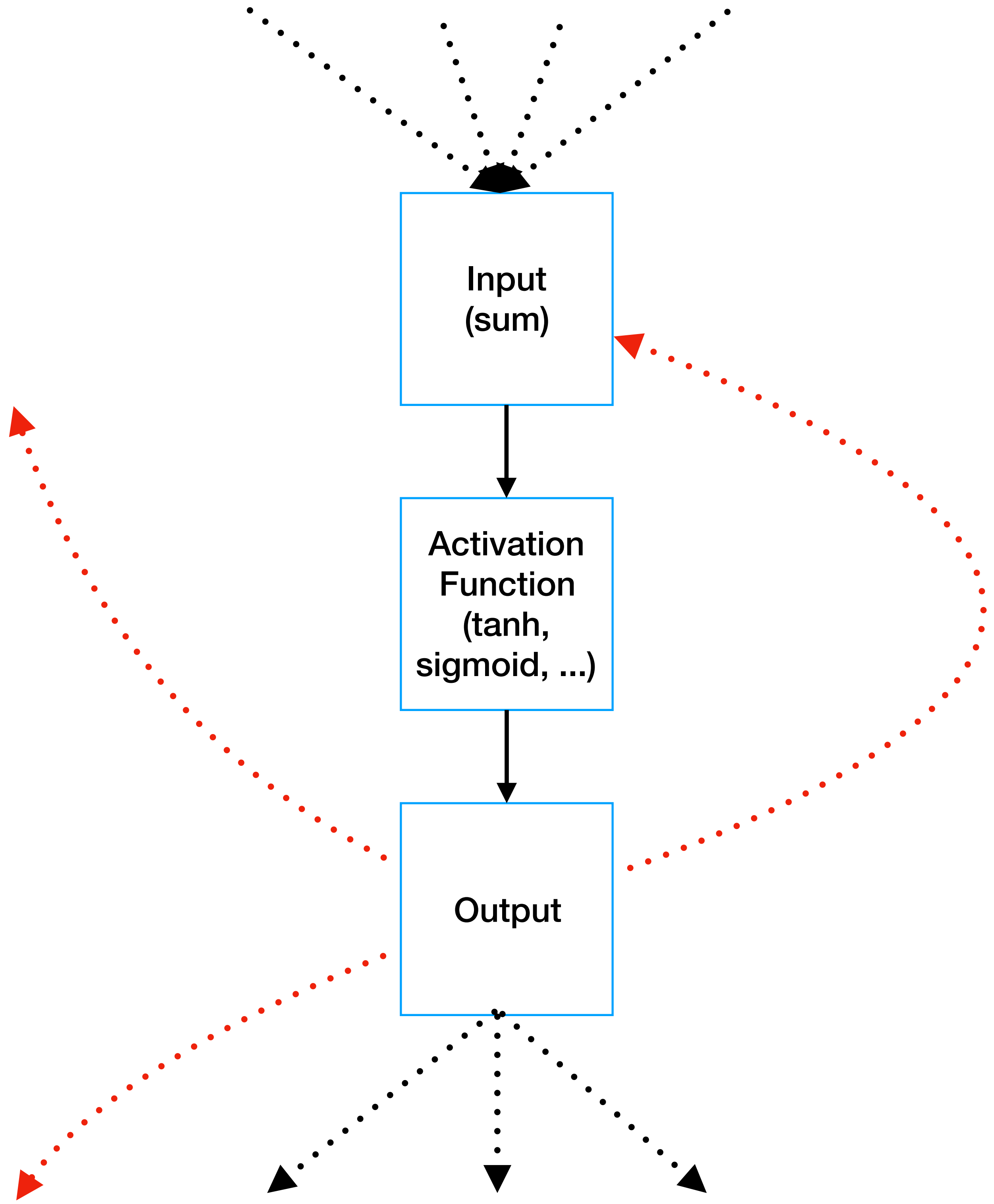
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

RNNs can also be used for time series data prediction, however in this case the RNN is predicting an exact value of a time series, some number of time steps in the future. The error being minimized is typically the mean squared error (1) or mean absolute error (2). *This is an important distinction.*

$$Error = \frac{0.5 \times \sum(Actual\ Vib - Predicted\ Vib)^2}{Testing\ Seconds} \qquad (1)$$

$$Error = \frac{\sum[ABS(Actual\ Vib - Predicted\ Vib)]}{Testing\ Seconds} \qquad (2)$$

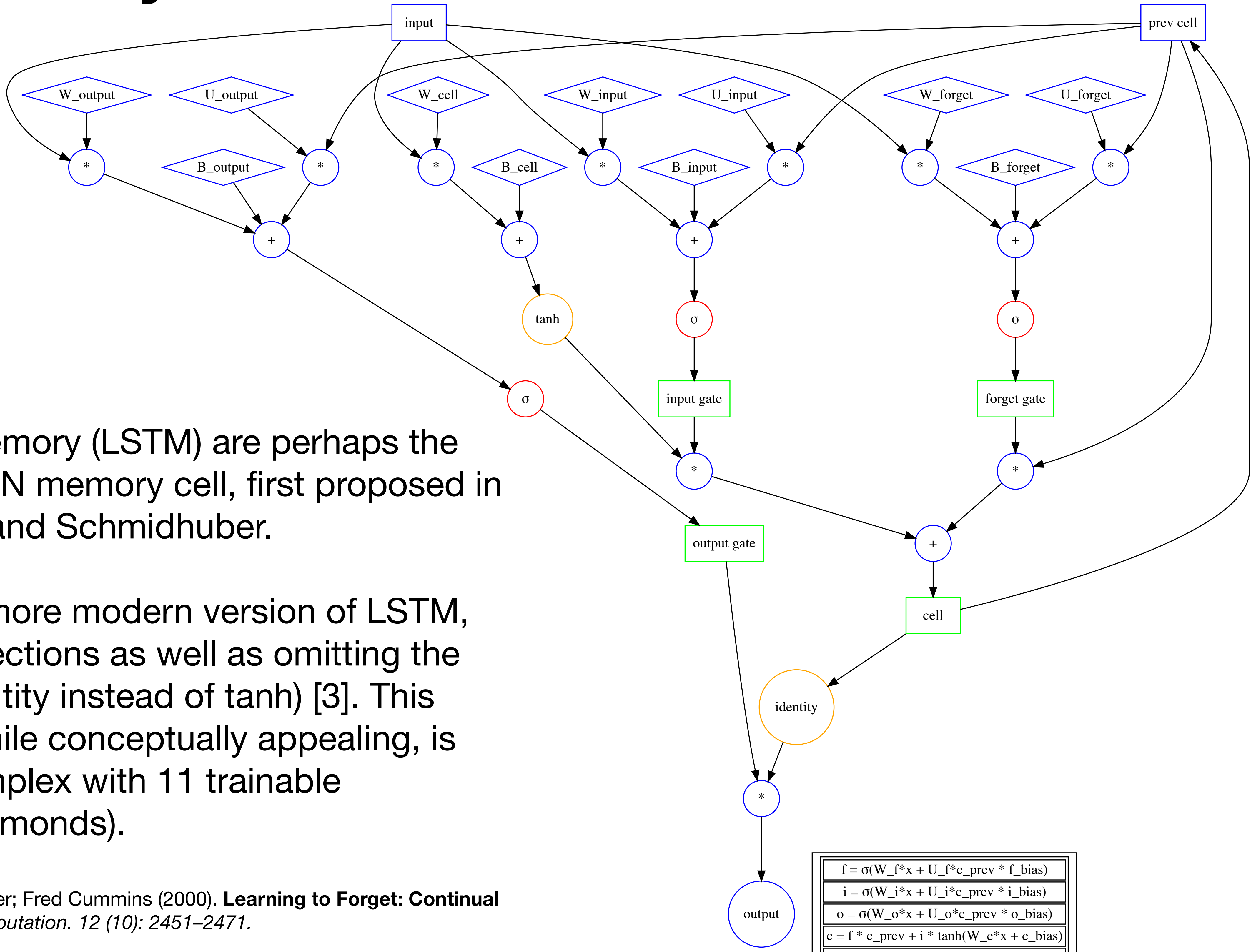
Memory Cell Structures - Simple Neuron



- "Simple" neurons are the basic neural network building block. Inputs are summed, and activation function is applied and the output is feed forward to other neurons.

- Simple (and any other memory cell structure) can also have recurrent edges (in red) added through the evolutionary process. These can loop back to the same neuron (e.g., an Elman-like connection) or to any other neuron in the network (shallower, same layer or deeper).

Memory Cell Structures - LSTM



f = σ(W_f*x + U_f*c_prev * f_bias)
i = σ(W_i*x + U_i*c_prev * i_bias)
o = σ(W_o*x + U_o*c_prev * o_bias)
c = f * c_prev + i * tanh(W_c*x + c_bias)
h = o * c

Long Short-Term Memory (LSTM) are perhaps the most well known RNN memory cell, first proposed in 1997 by Hochreiter and Schmidhuber.

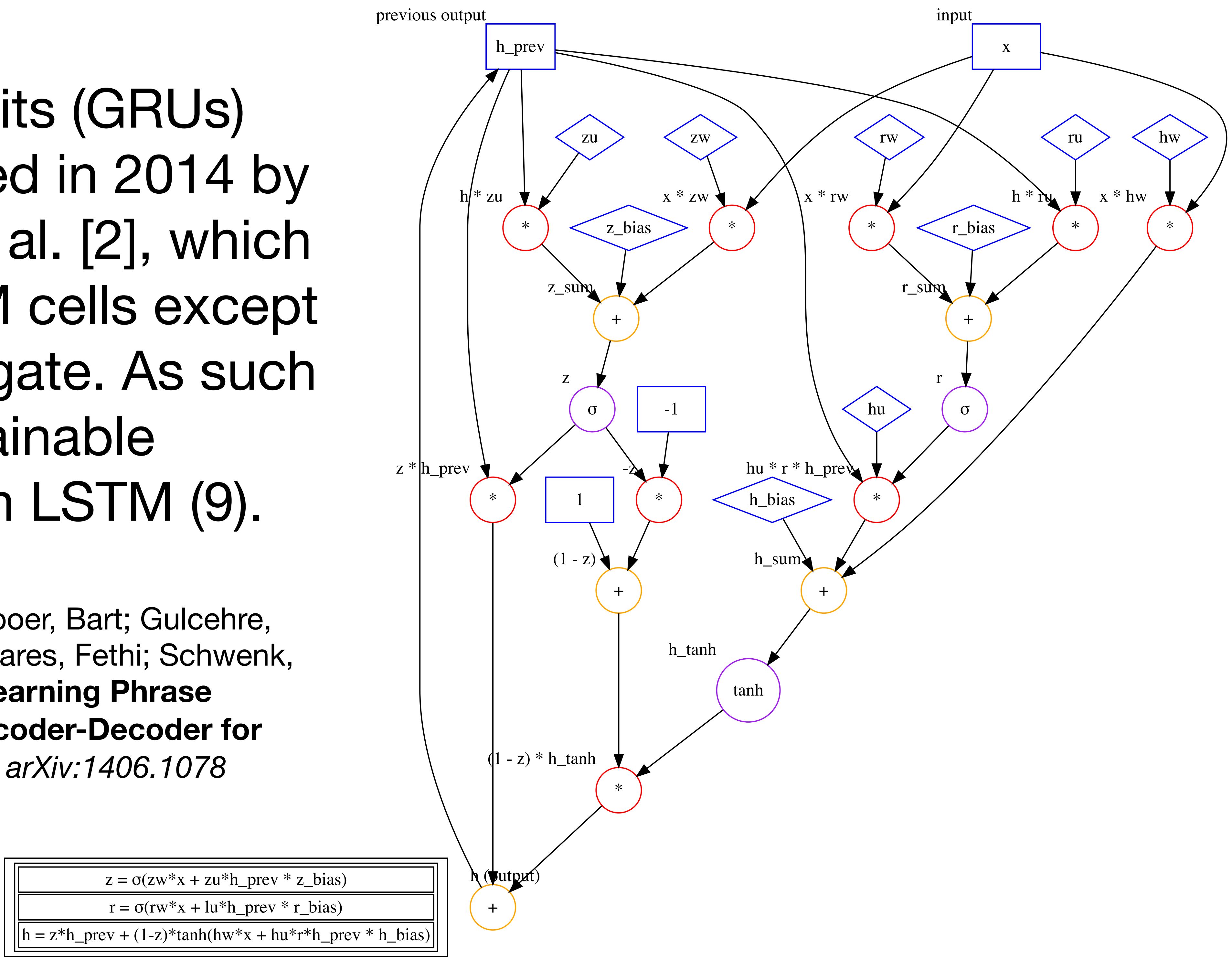
This work uses the more modern version of LSTM, with peephole connections as well as omitting the output function (identity instead of tanh) [3]. This cellular structure, while conceptually appealing, is computationally complex with 11 trainable parameters (blue diamonds).

[1] Felix A. Gers; Jürgen Schmidhuber; Fred Cummins (2000). **Learning to Forget: Continual Prediction with LSTM**. *Neural Computation. 12 (10): 2451–2471.*

Memory Cell Structures - GRU

Gated recurrent units (GRUs) were first introduced in 2014 by Kyunghyun Cho et al. [2], which are similar to LSTM cells except without an output gate. As such it requires fewer trainable parameters than an LSTM (9).

[2] Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). **Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation**. *arXiv:1406.1078*

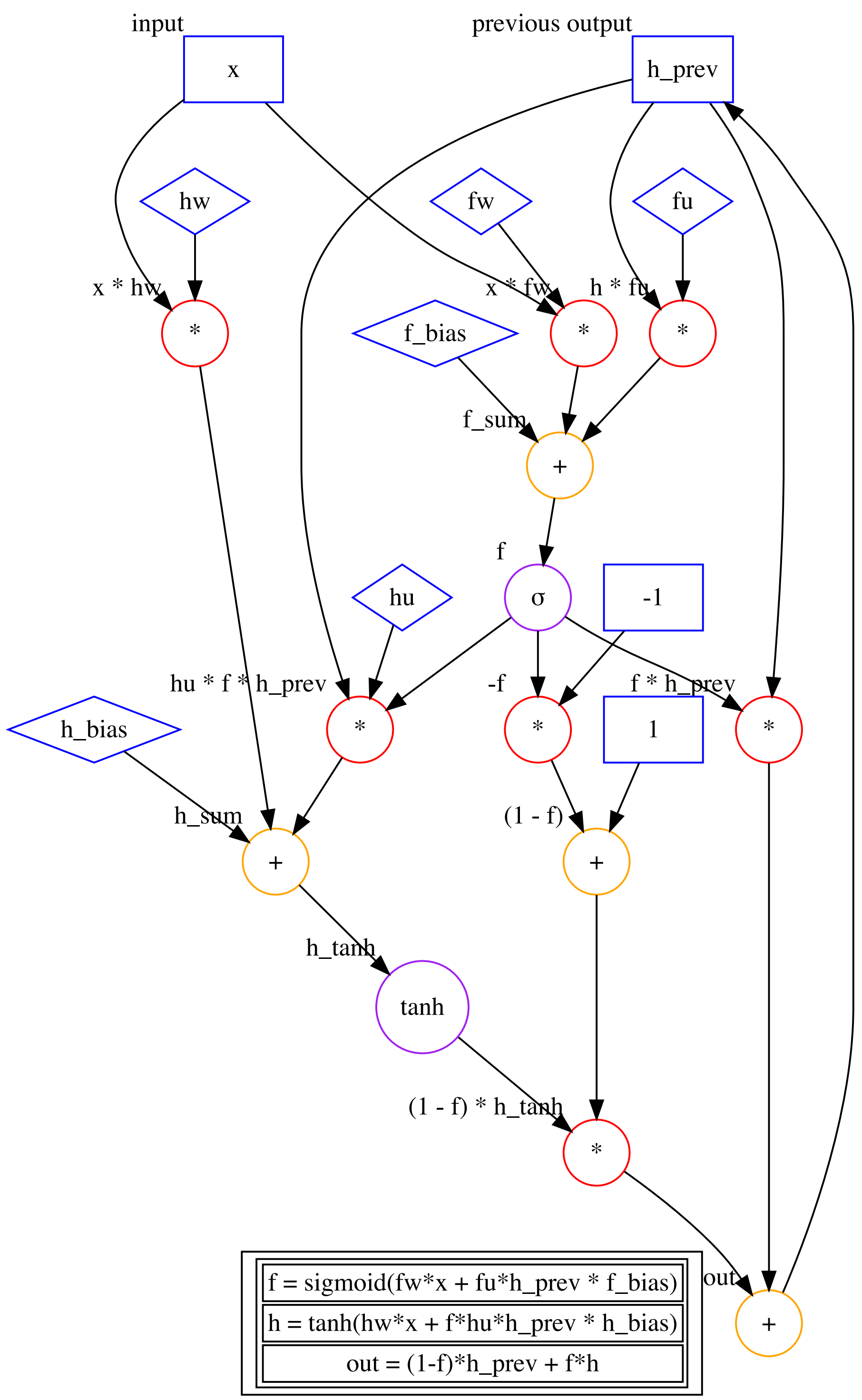


z = σ(zw*x + zu*h_prev * z_bias)

r = σ(rw*x + lu*h_prev * r_bias)

h = z*h_prev + (1-z)*tanh(hw*x + hu*r*h_prev * h_bias)

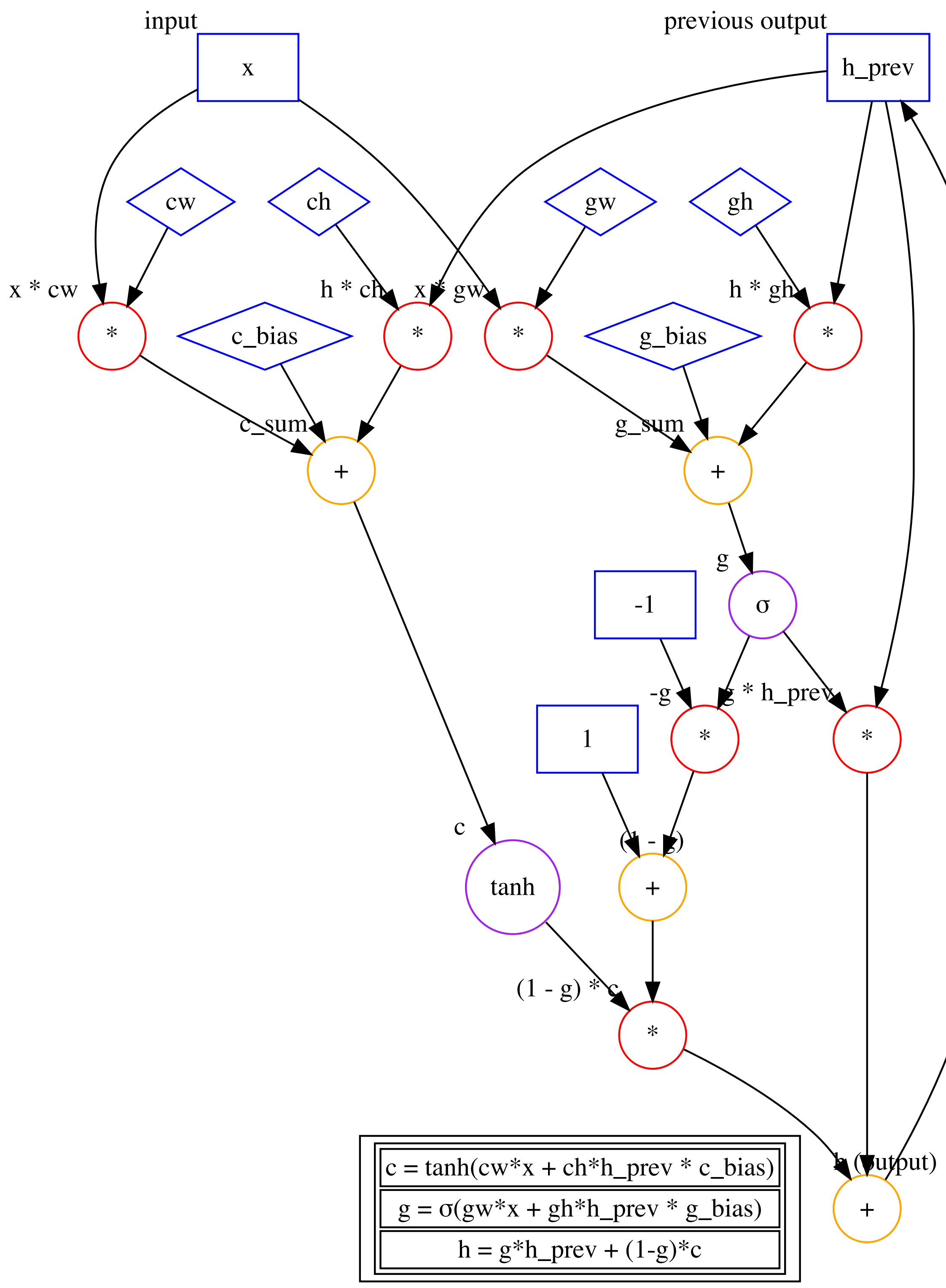
Memory Cell Structures - MGU



Minimal gated units (MGUs) were proposed by Zhou et al. in 2016 [3]. It is another example of an effective recurrent cell with a low number of trainable parameters (6).

[3] Gou-Bing Zhou,Jianxin Wu, Chen-Lin Zhang and Zhi-Hua Zhou. **Minimal gated unit for recurrent neural networks.** *International Journal of Automation and Computing 13.3 (2016): 226-234. APA*

Memory Cell Structures - UGRNN

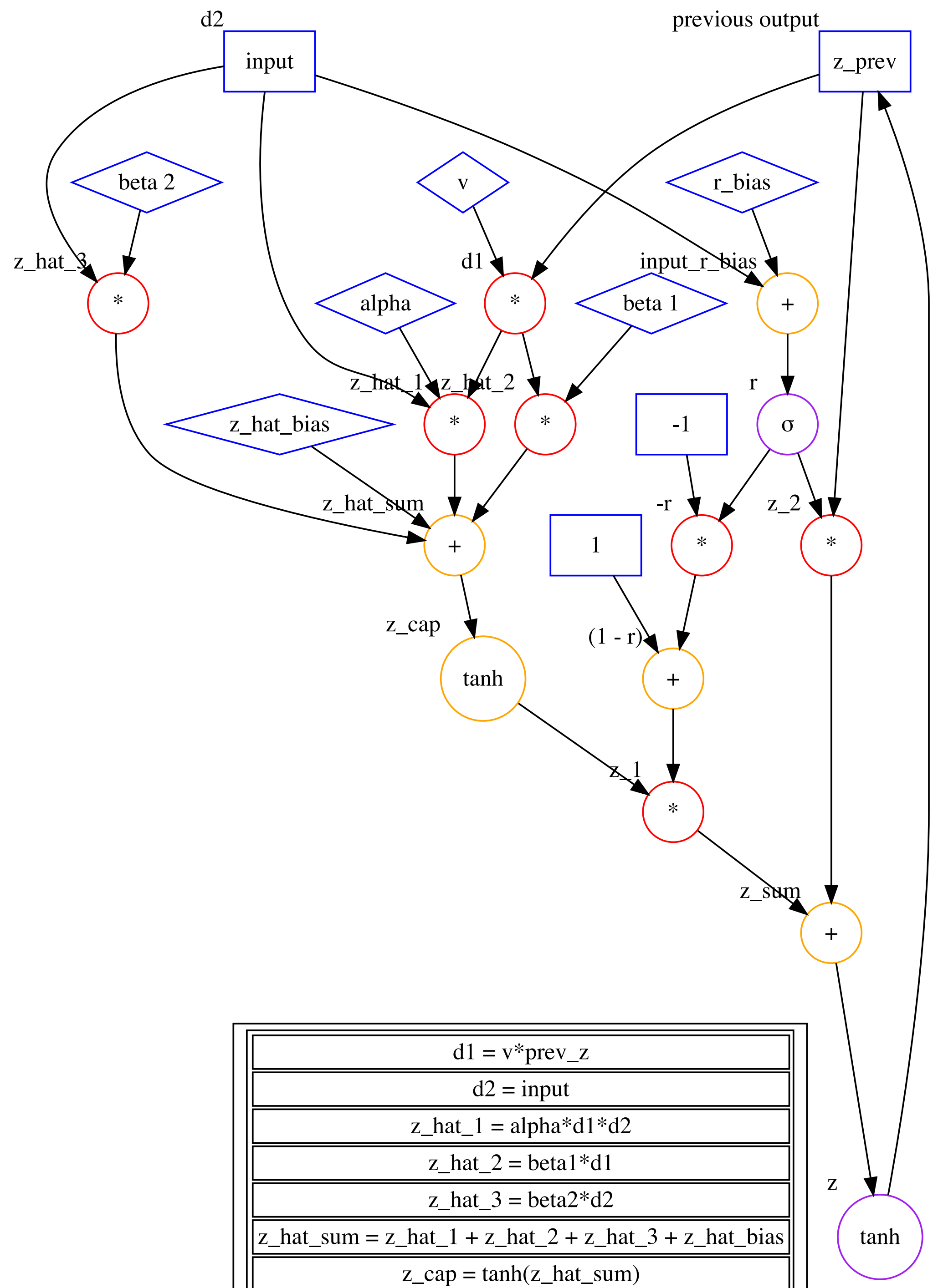


Update gate recurrent neural networks (UGRNNs) were introduced in 2016 by Collins et al. [4]

UGRNNs are another simple model with only 6 trainable parameters.

[4] Collins, Jasmine, Jascha Sohl-Dickstein, and David Sussillo. **Capacity and trainability in recurrent neural networks.** *arXiv preprint arXiv:1611.09913 (2016).*

Memory Cell Structures - Delta-RNN



d1 = v*prev_z
d2 = input
z_hat_1 = alpha*d1*d2
z_hat_2 = beta1*d1
z_hat_3 = beta2*d2
z_hat_sum = z_hat_1 + z_hat_2 + z_hat_3 + z_hat_bias
z_cap = tanh(z_hat_sum)
r = σ(d2 + r_bias)
z_1 = (1-r)*z_cap
z_2 = r*prev_z
z = tanh(z_1 + z_2)

Delta-RNN cells were first developed by Ororbia et al. in 2017 [5], and have shown to have comparable performance to other memory cells with fewer trainable parameters (6).

[5] Ororbia II, Alexander G., Tomas Mikolov, and David Reitter. **Learning simpler language models with the differential state framework.** *Neural computation 29.12 (2017): 3327-3352.*

EXAMM: Evolutionary eXploration of Augmenting Memory Models

EXAMM

- Neuro-Evolution algorithm inspired by Neuro-Evolution of Augmenting Topologies (NEAT) [6].

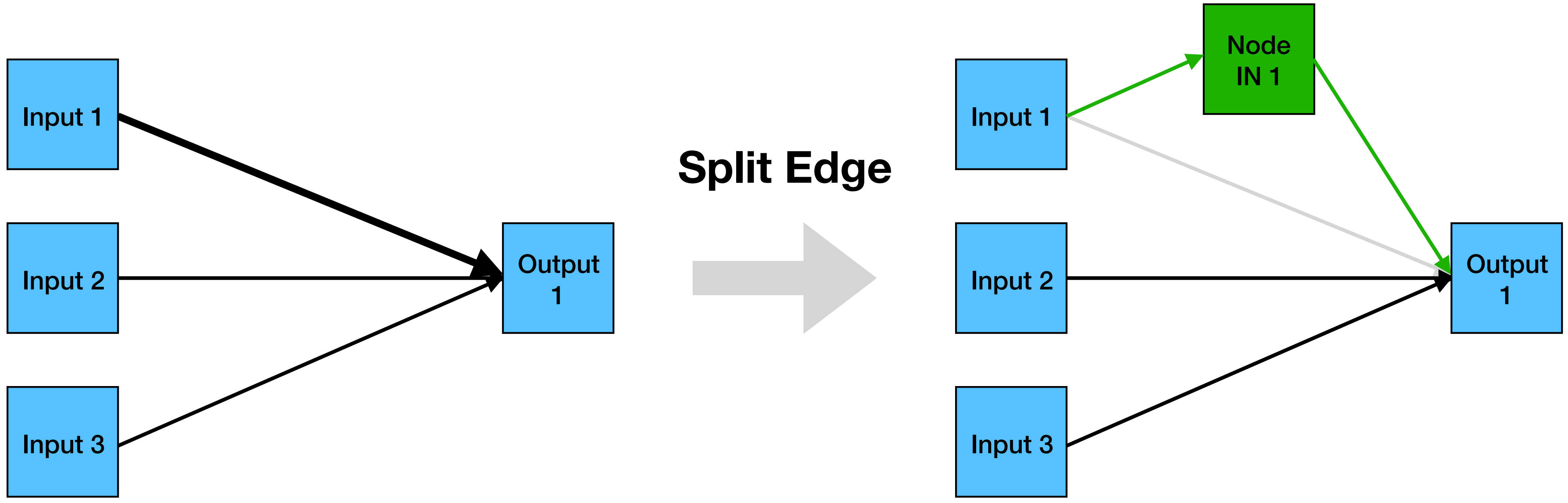
- Advancement of the earlier Evolutionary Exploration of Augmenting LSTM Topologies (EXALT) [7]:
  - Progressively grows RNNs: nodes can be simple neurons or LSTMs.
  - Parallel in nature.
  - Node-level mutations not present in NEAT.
  - Uses Lamarckian/Epigenetic weight initialization - child RNNs utilize weights from their parent(s).

- Evolutionary Exploration of Augmenting Memory Models (EXAMM)
  - Based on EXALT, except with a library of memory cells. Nodes can be simple, LSTM, GRU, UGRNN, MGU, or Delta-RNNs.
  - Island-based Parallelism.
  - Mutations have further refinements from EXALT.

[6] Kenneth Stanley and Risto Miikkulainen. **Evolving neural networks through augmenting topologies.** *Evolutionary computation: 10, 2*. (2002), 99–127.
[7] AbdElRahman ElSaid, Steven Benson, Shuchita Patwardhan, David Stadem, and Travis Desell. **Evolving Recurrent Neural Networks for Time Series Data Prediction of Coal Plant Parameters.** *The 22nd International Conference on the Applications of Evolutionary Computation (EvoStar: EvoApps 2019).*

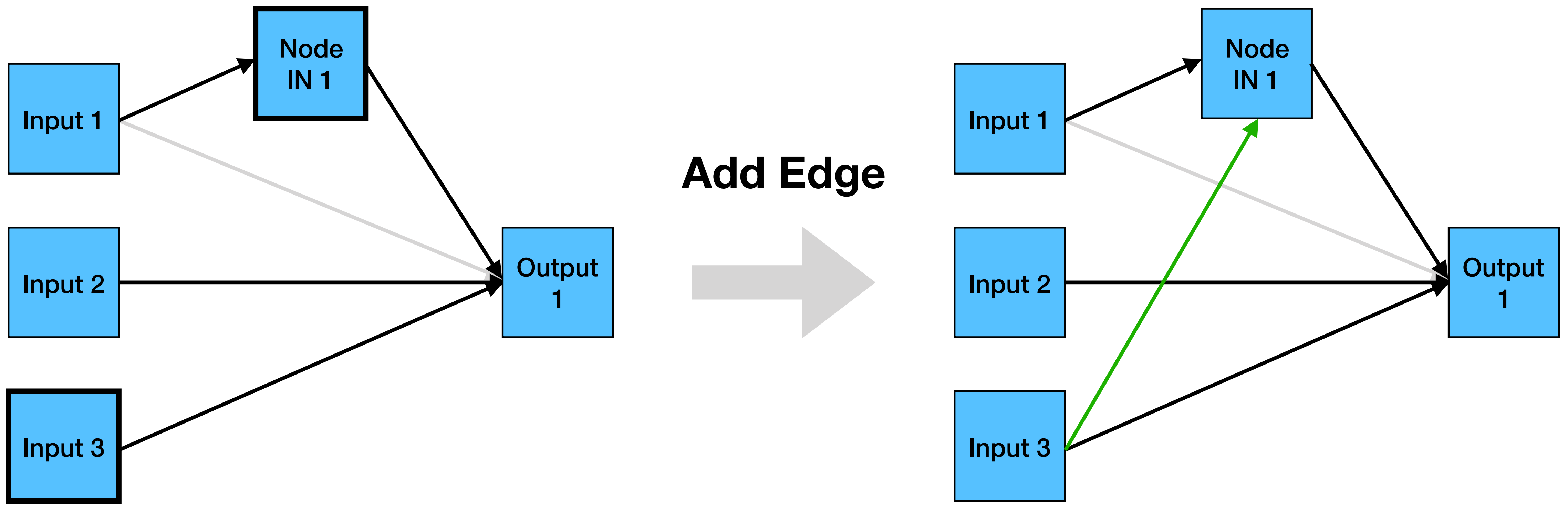
Edge and Node Mutations

Edge Mutations: Split Edge



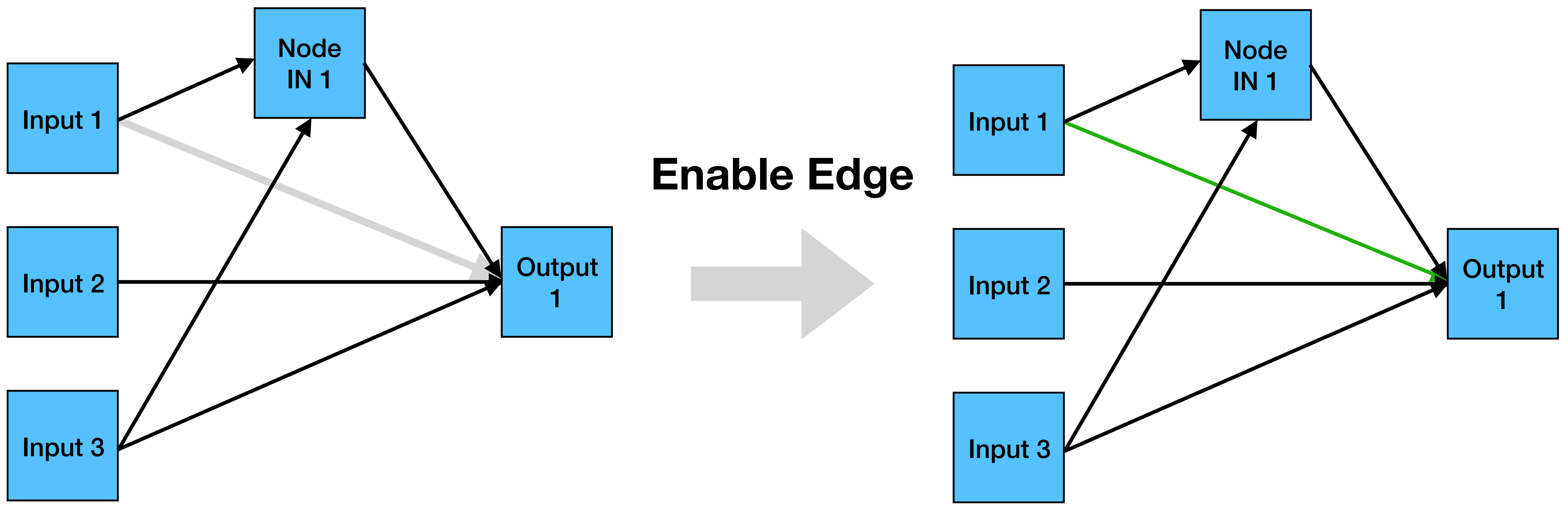
- EXAMM always starts with a minimal feed forward network (top left) with input nodes for each input parameter fully connected to output nodes for each output parameter (no hidden nodes).
- The edge between Input 1 and Output 1 is selected to be split. A new node with innovation number (IN) 1 is created.

Edge Mutations: Add Edge



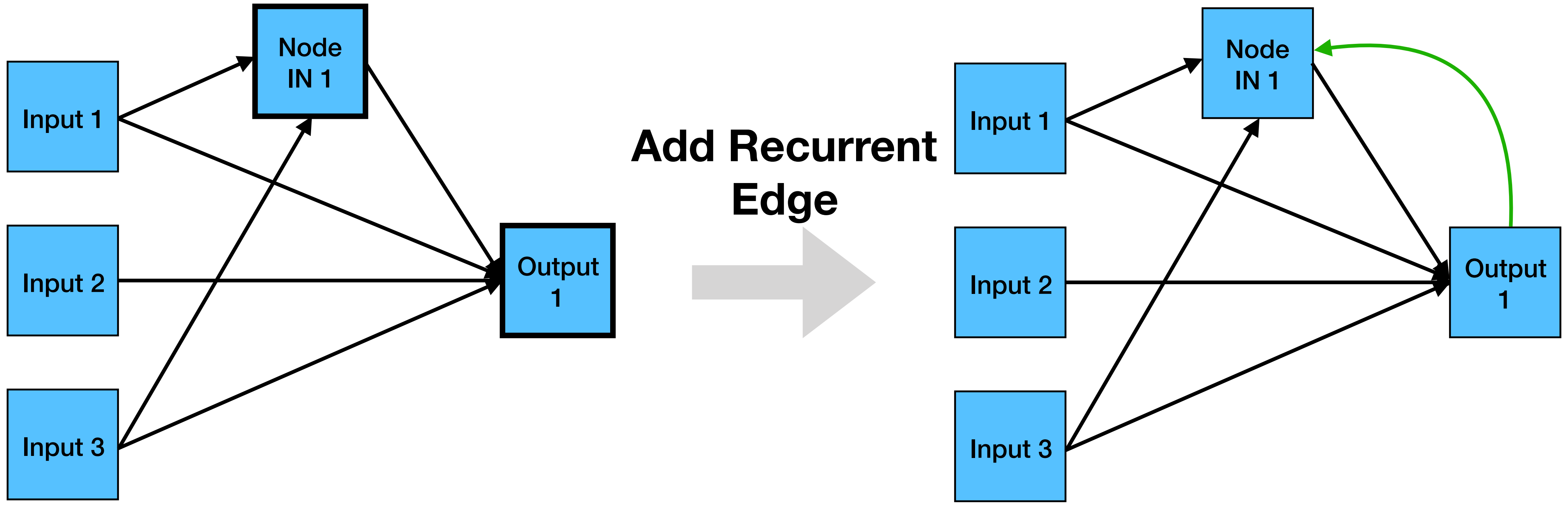
- Input 3 and Node IN 1 are selected to have an edge between them added.

Edge Mutations: Enable Edge



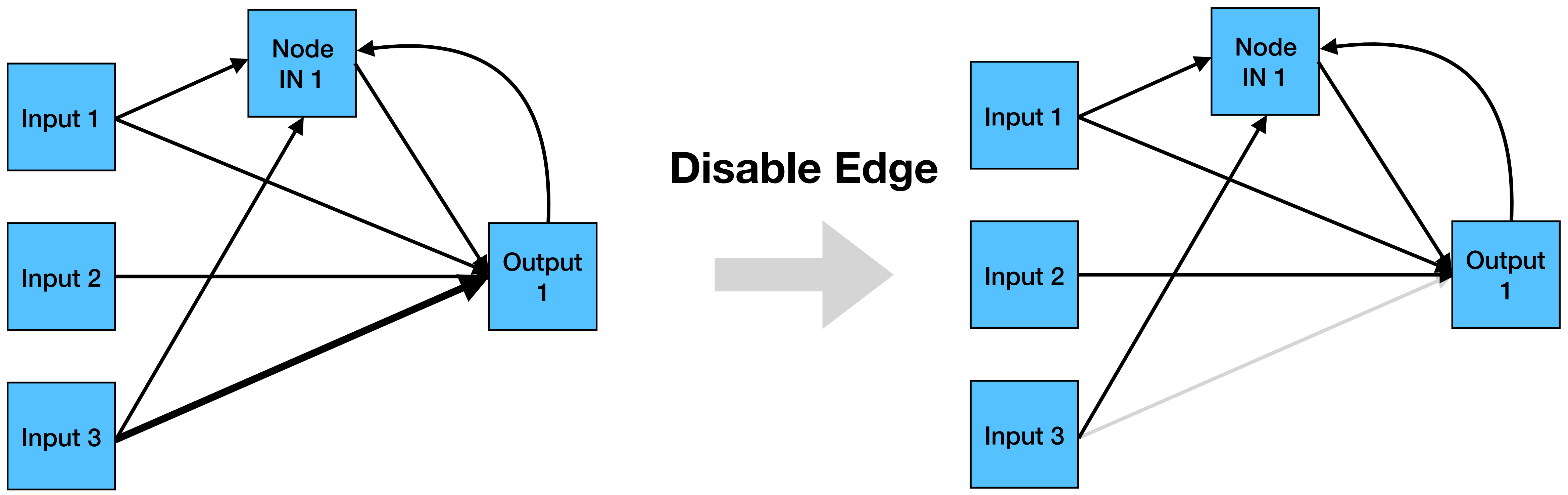
- The edge between Input 3 and Output 1 is enabled.

Edge Mutations: Add Recurrent Edge



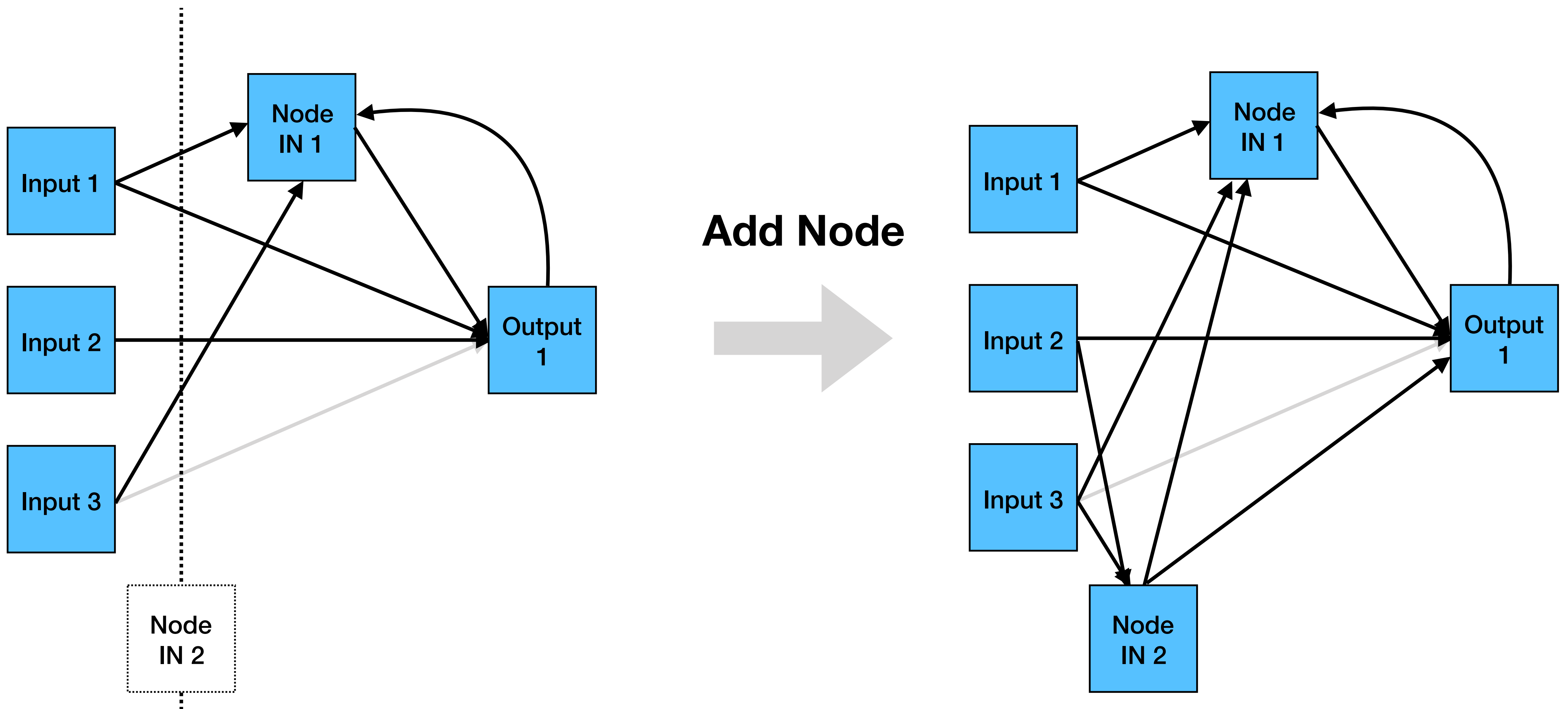
- A recurrent edge is added between Output 1 and Node IN 1.

Edge Mutations: Disable Edge



- The edge between Input 3 and Output 1 is disabled.

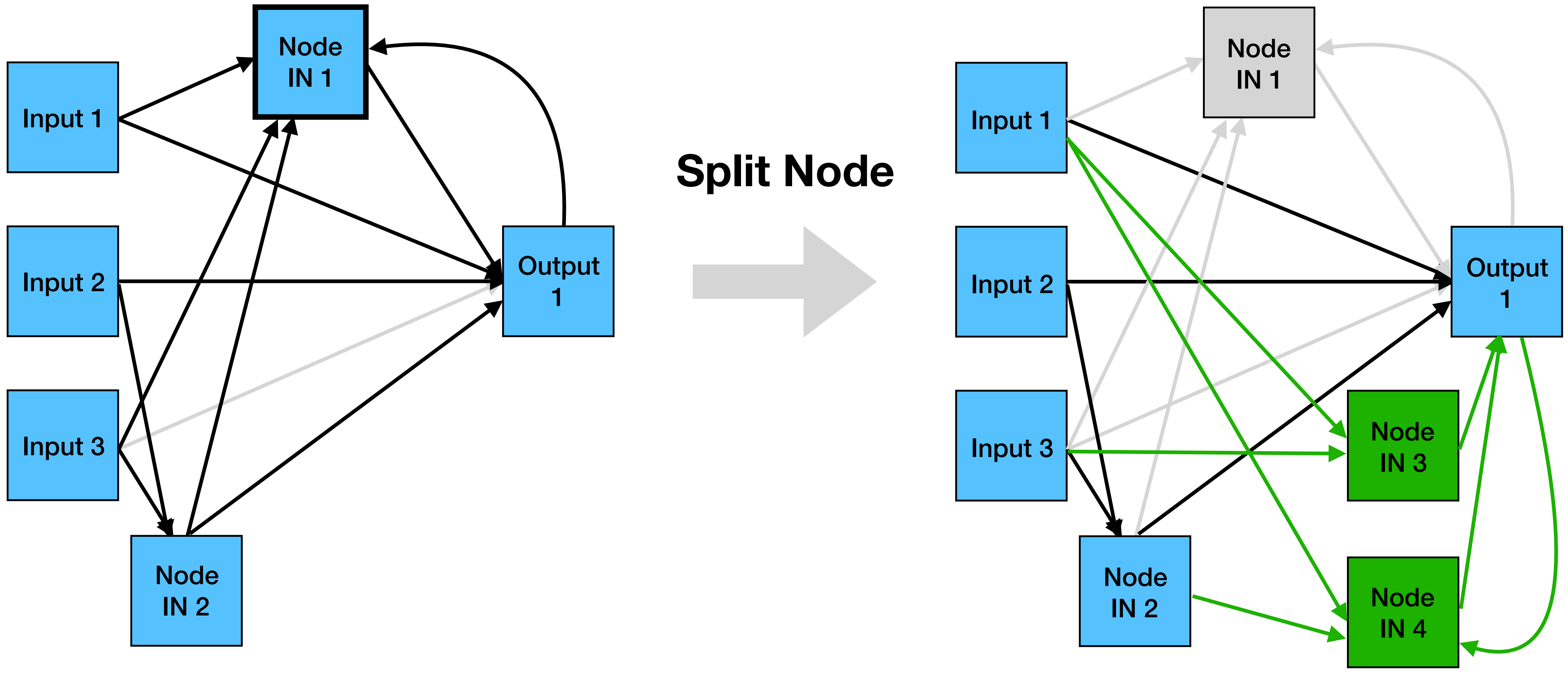
Node Mutations: Add Node



- A node with IN 2 is selected to be added at a depth between the inputs & Node IN 1. Edges are randomly added to Input 2 and 3, and Node IN 1 and Output 1.

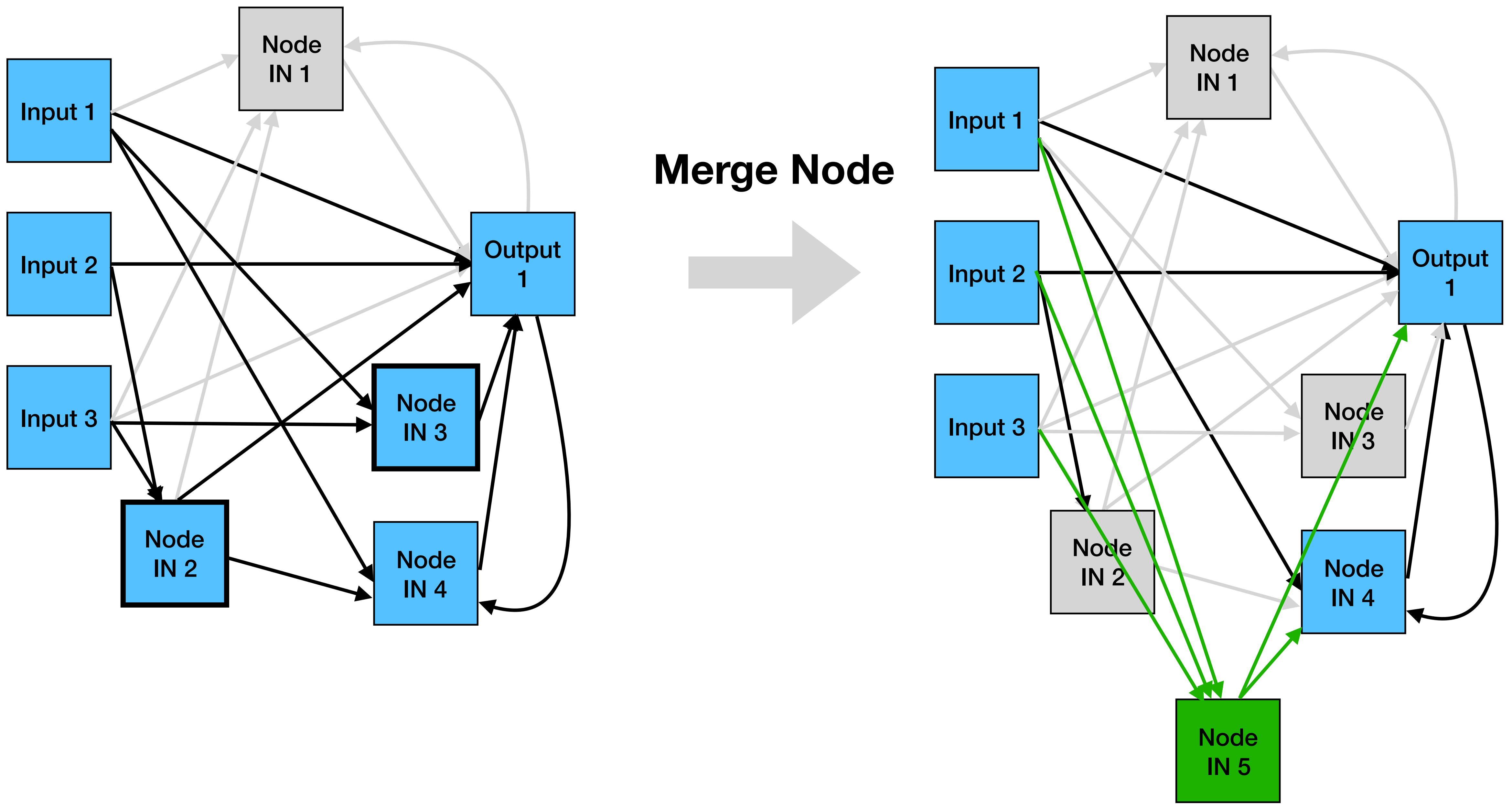
- The number of edges added is determined by calculating the mean and variance of the number of input and output edges for all other nodes in the network, and selecting randomly via a normal distribution with those means/variances.

Node Mutations: Split Node



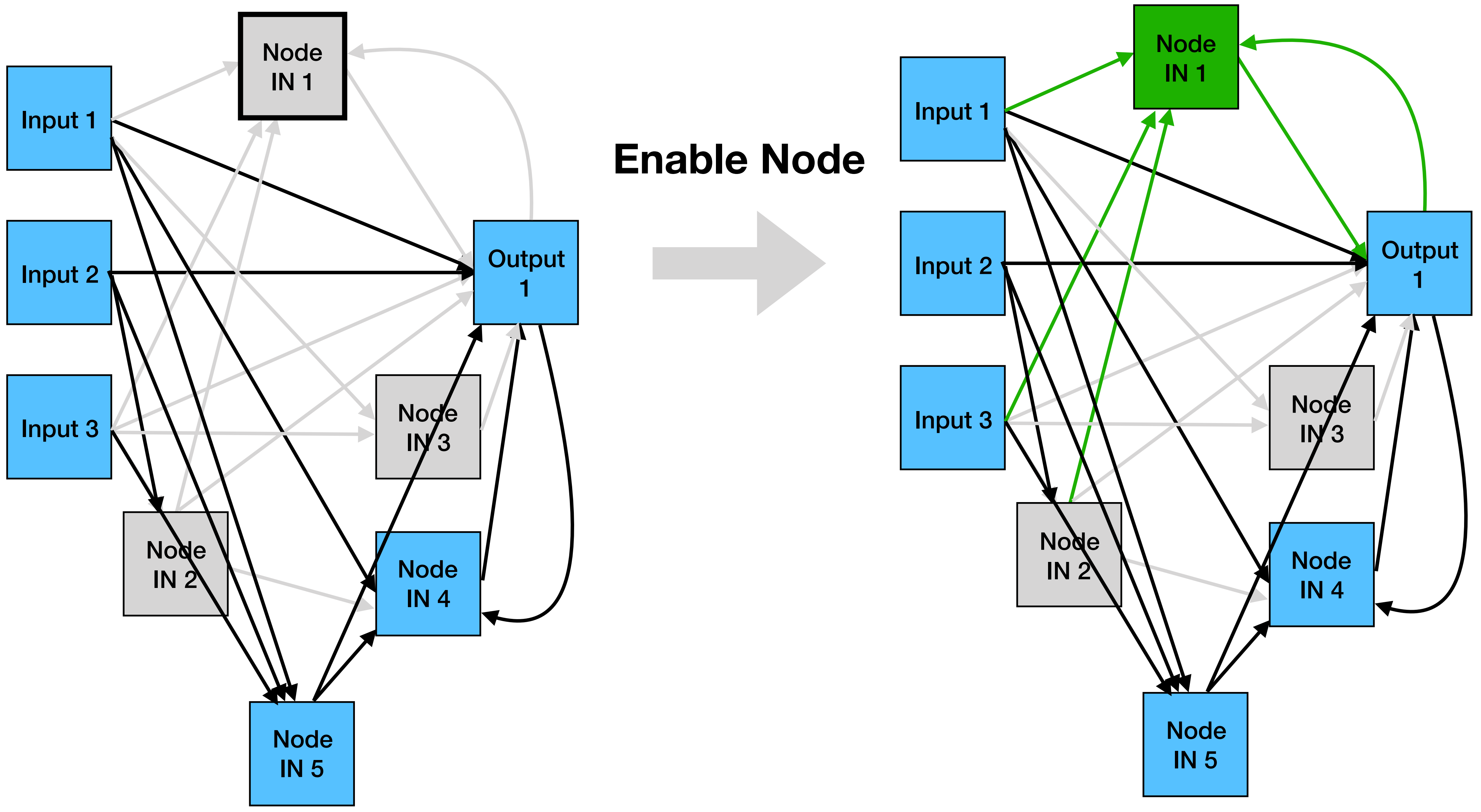
- Node IN 1 is selected to be split. It is disabled with its input/output edges. It is split into Nodes IN 3 and 4, which get half the inputs. Both have an output edge to Output 1 since there was only one output from Node IN 1.

Node Mutations: Merge Node



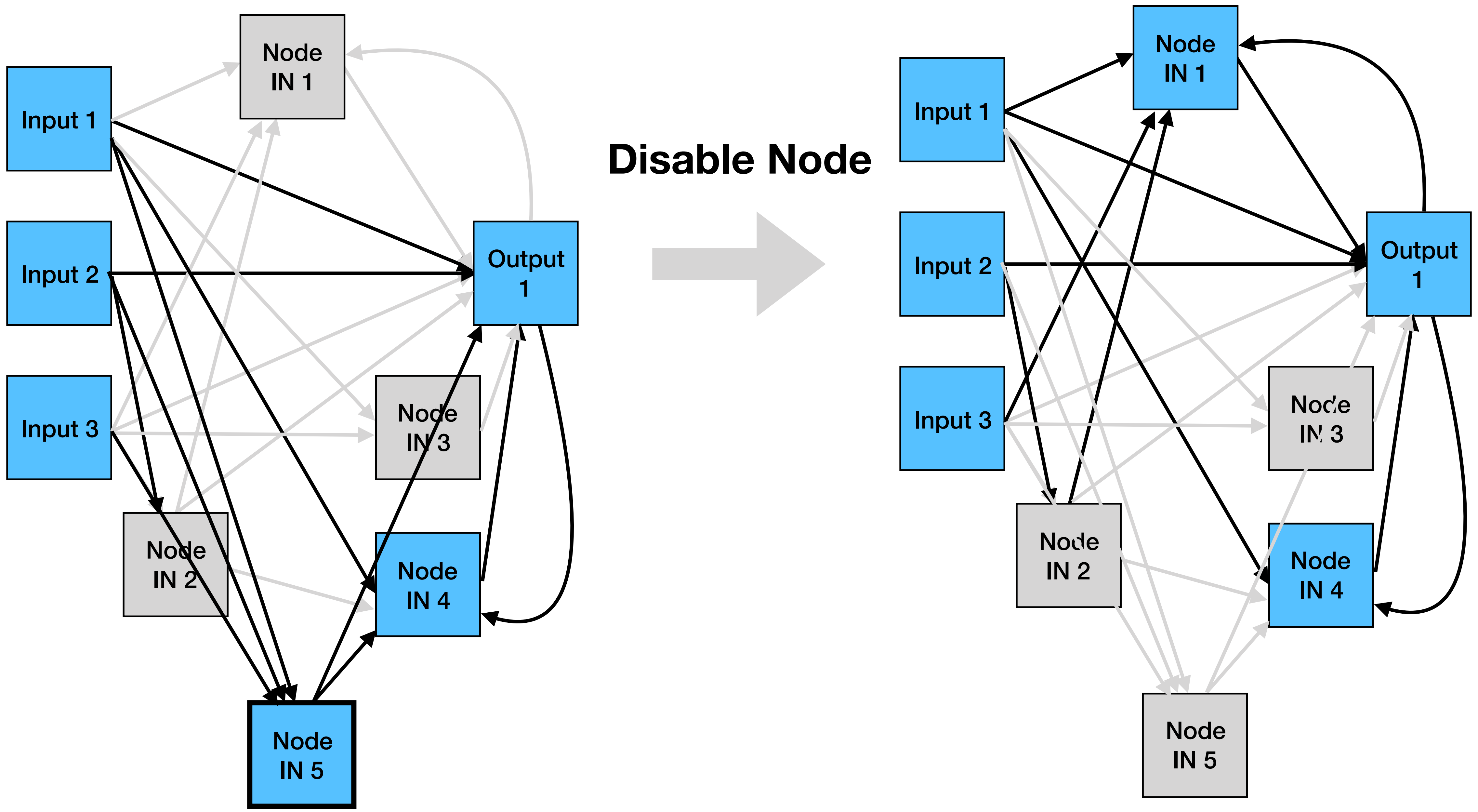
- Node IN 2 and 3 are selected for a merger (input/output edges are disabled). Node IN 5 is created with edges between all their inputs/outputs.

Node Mutations: Enable Node



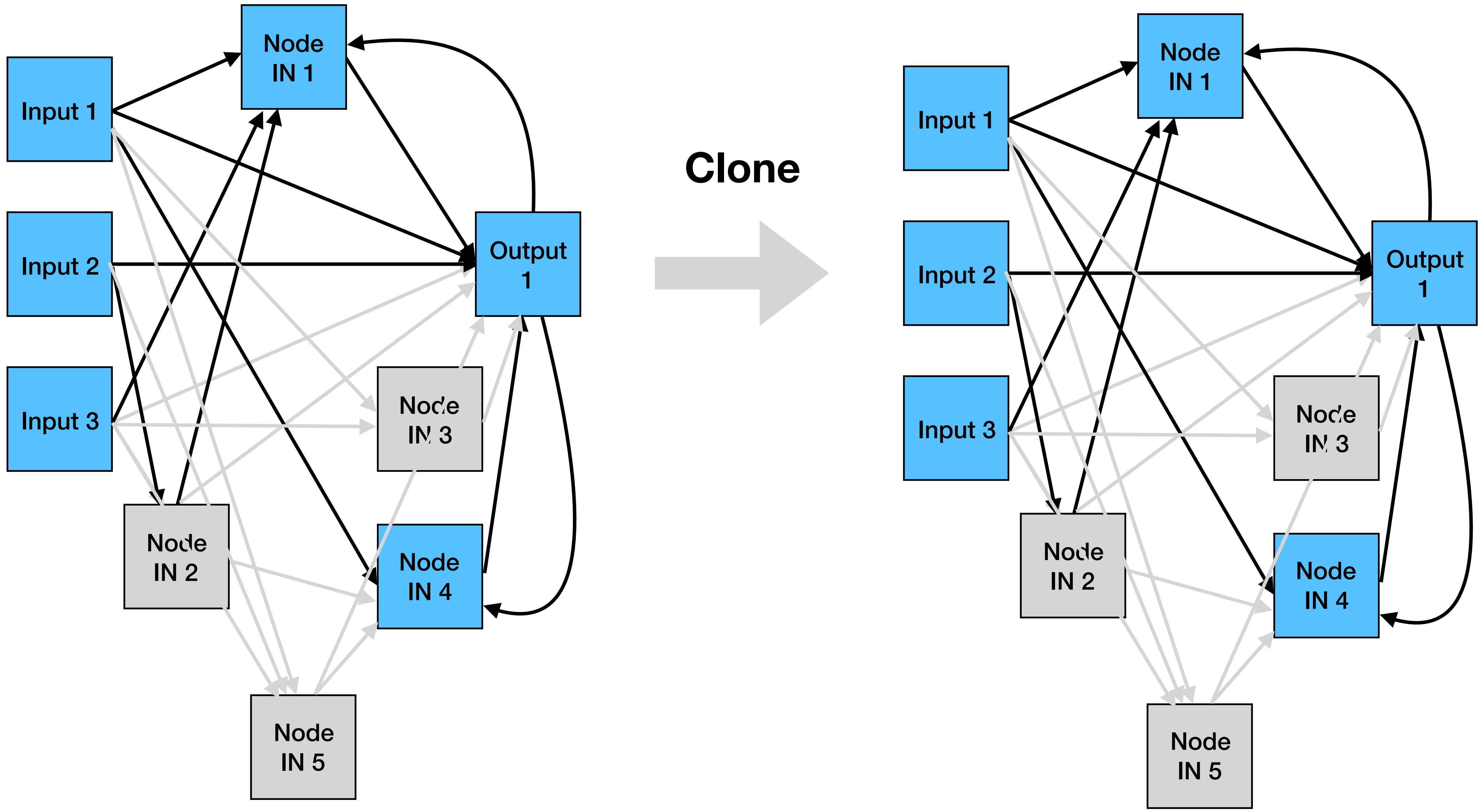
- Node IN 1 is selected to be enabled, along with all its input and output edges.

Node Mutations: Disable Node



- Node IN 5 is selected to be disabled, along with all its input and output edges.

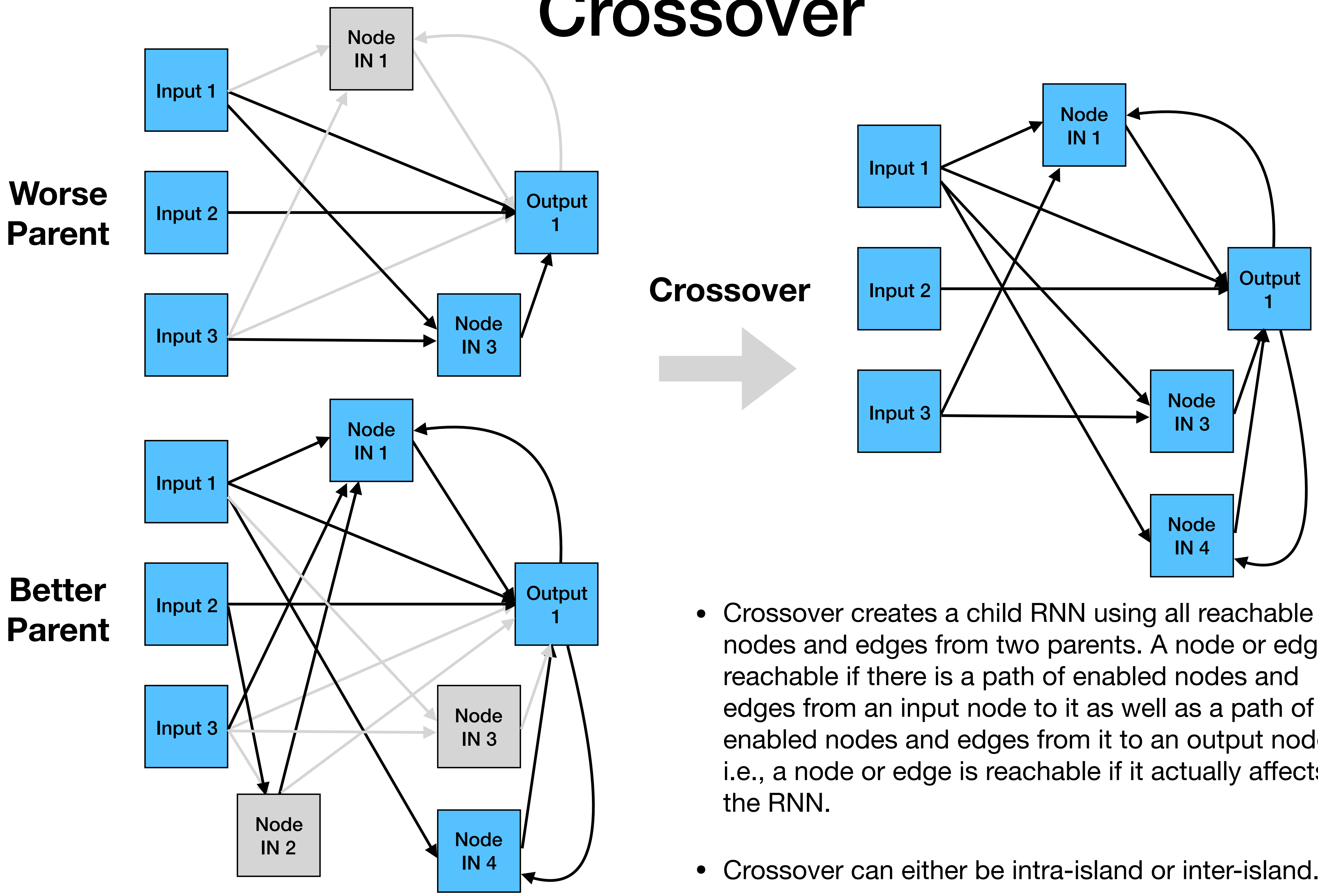
Clone



- Clone makes no modifications at all to the parent, allowing it to continue with the back propagation process.

Crossover

Crossover



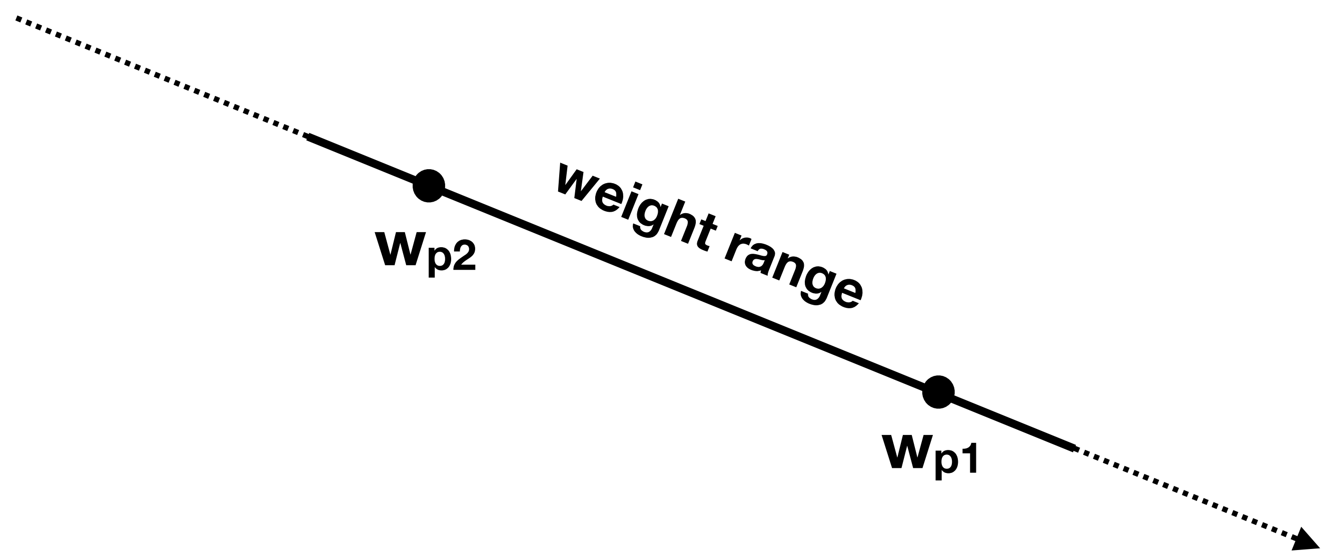
- Crossover creates a child RNN using all reachable nodes and edges from two parents. A node or edge is reachable if there is a path of enabled nodes and edges from an input node to it as well as a path of enabled nodes and edges from it to an output node, i.e., a node or edge is reachable if it actually affects the RNN.

- Crossover can either be intra-island or inter-island.

Crossover: Lamarckian Weight Initialization

- Initial RNN weights generated uniformly at random (between -0.5 and 0.5).
- New components (nodes/edges) are generated a normal distribution based on the average, standard deviation, and variance of the parent(s) weights.

Crossover: Lamarckian Weight Initialization



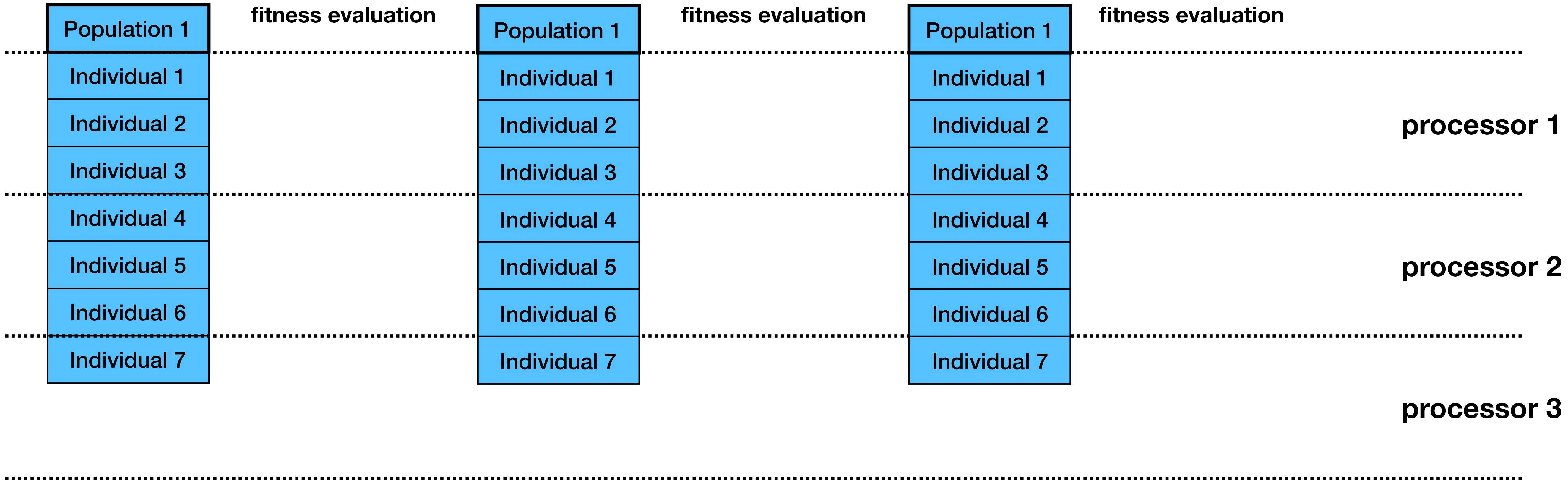
- In crossover where a node/edge exists in both parents we recombine the weights. The child weights, w_c, are generated by recombining the parents' weights:

$$w_c = r(w_{p2} - w_{p1}) + w_{p1}$$

- Where r is a random number $-0.5 <= r <= 1.5$, where w_{p1} is the weight from the more fit parent, and w_{p2} is the weight from the less fit parent. We can change r's bounds to prefer weights near one parent over the other.

Distributed Neuro-Evolution

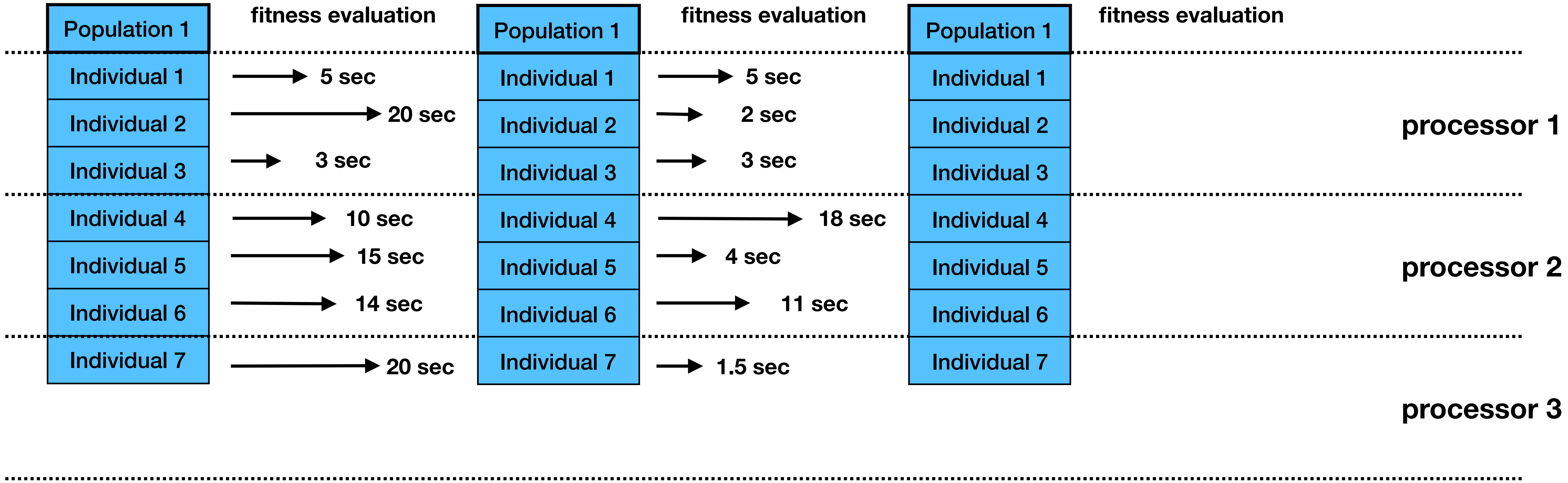
Synchronous/Parallel EAs



- Traditional EAs generate an entire population at a time, evaluate the fitness of every individual and then generate the next population.

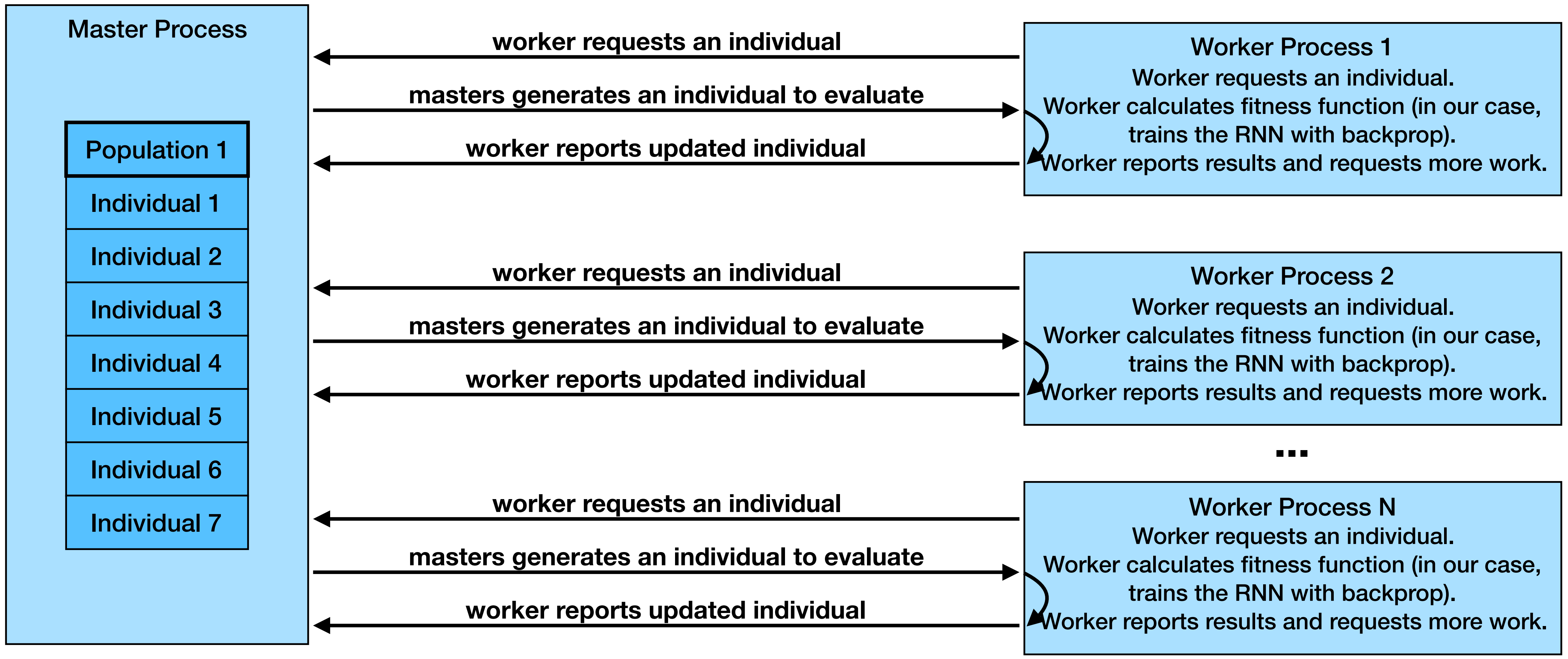
- This has problems in that if the population size is not evenly divisible by the number of processors available there is wasted computation. Also, the population size can't be *less* than the number of processors.

Synchronous/Parallel EAs



- Things are even more challenging if the fitness evaluation times of the individuals are different or even worse nondeterministic. Lots of waiting and unused cycles.

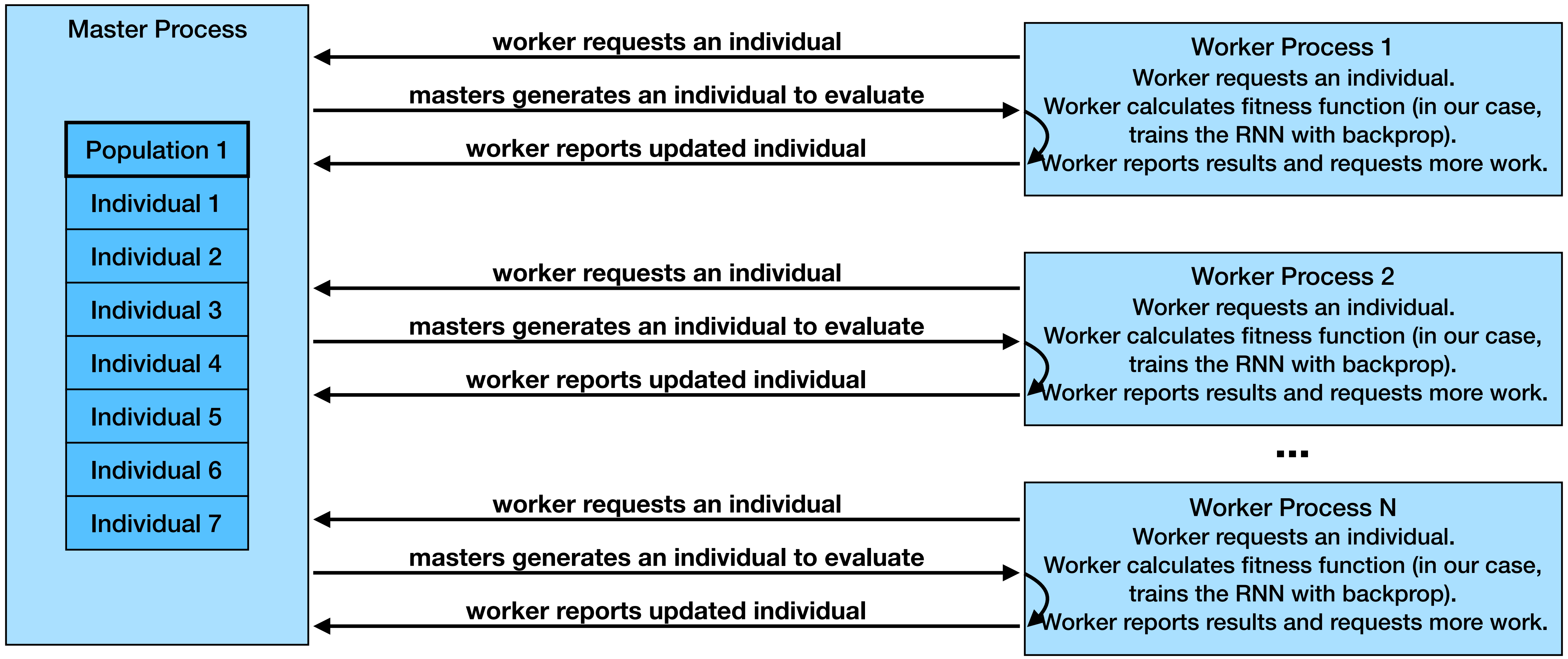
Asynchronous EAs



Master Process
Population 1
Individual 1
Individual 2
Individual 3
Individual 4
Individual 5
Individual 6
Individual 7
worker requests an individual
masters generates an individual to evaluate
worker reports updated individual
Worker Process 1
Worker requests an individual.
Worker calculates fitness function (in our case, trains the RNN with backprop).
Worker reports results and requests more work.
Worker Process 2
Worker Process N
...

- The Master process keeps a "steady state" popuation.
- Workers independently request work (master generates new RNNs to train), calculate fitness and report results.
- No worker waits on another worker - naturally load balanced. Workers can even request a queue of work to reduce latency.
- Number of worker processes is independent of population size.

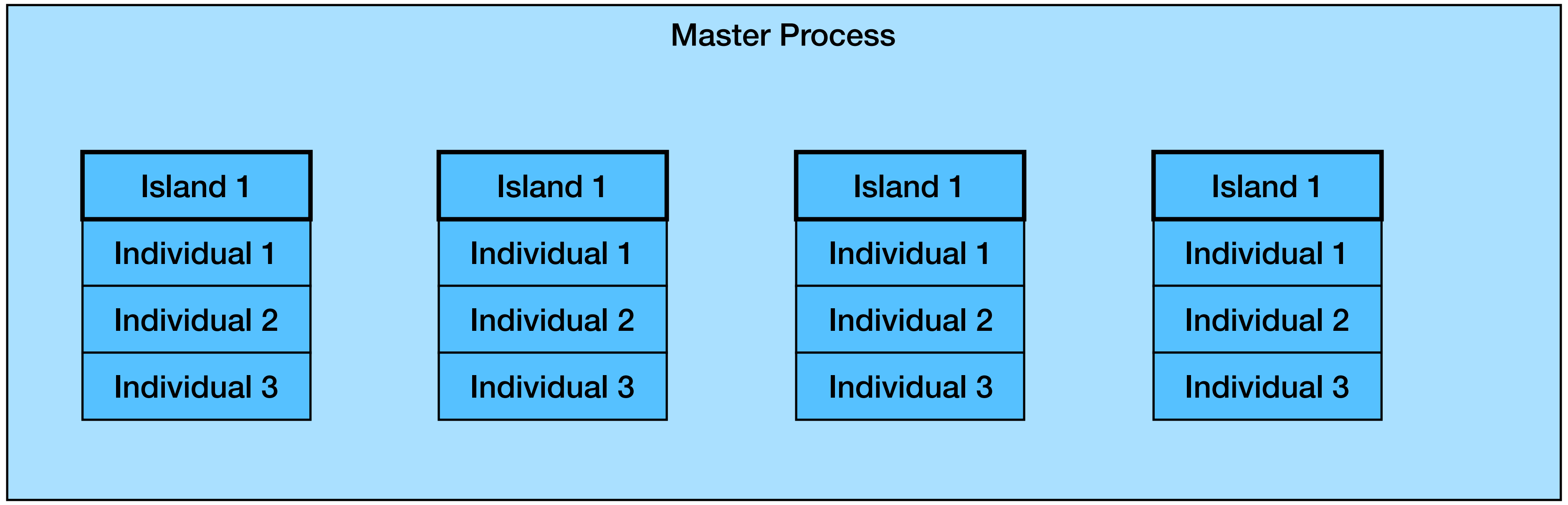
Asynchronous EAs


Master Process
Population 1
Individual 1
Individual 2
Individual 3
Individual 4
Individual 5
Individual 6
Individual 7
worker requests an individual
masters generates an individual to evaluate
worker reports updated individual
Worker Process 1
Worker requests an individual.
Worker calculates fitness function (in our case, trains the RNN with backprop).
Worker reports results and requests more work.
worker requests an individual
masters generates an individual to evaluate
worker reports updated individual
Worker Process 2
Worker requests an individual.
Worker calculates fitness function (in our case, trains the RNN with backprop).
Worker reports results and requests more work.
...
worker requests an individual
masters generates an individual to evaluate
worker reports updated individual
Worker Process N
Worker requests an individual.
Worker calculates fitness function (in our case, trains the RNN with backprop).
Worker reports results and requests more work.

- Asynchronous EAs can scale to millions of processors, whereas synchronous EAs are very limited [1].

[1] Travis Desell, David P. Anderson, Malik Magdon-Ismail, Heidi Newberg, Boleslaw Szymanski and Carlos A. Varela. **An Analysis of Massively Distributed Evolutionary Algorithms**. *In the Proceedings of the 2010 IEEE Congress on Evolutionary Computation (IEEE CEC 2010)*. pages 1-8. Barcelona, Spain. July 2010.

Islands

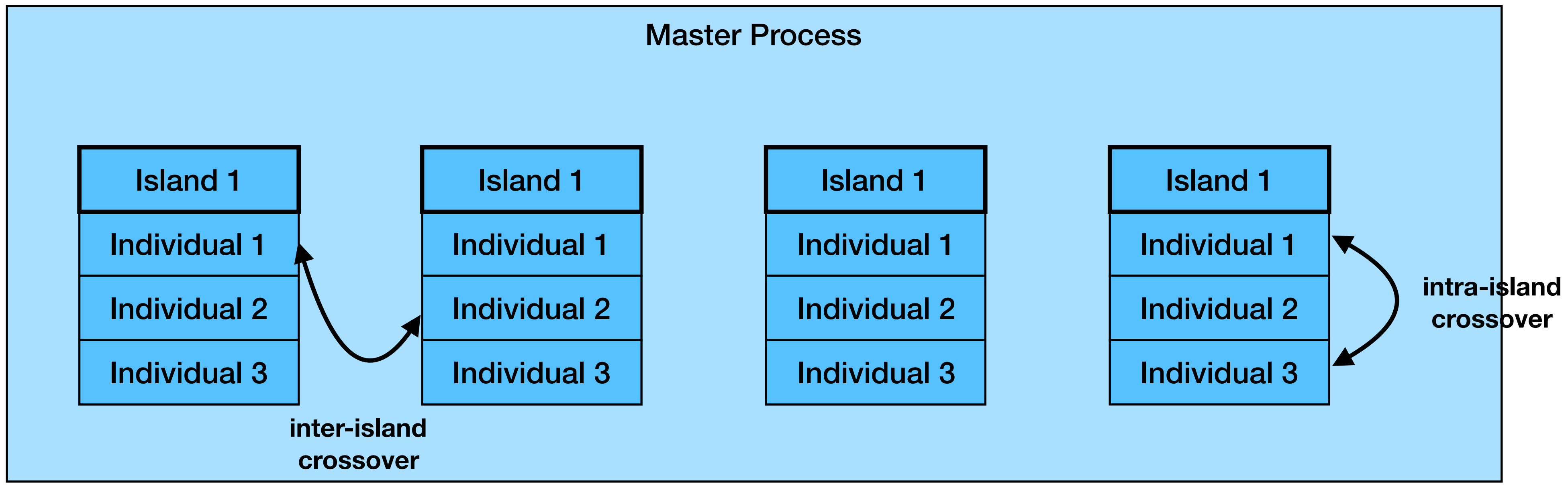


- EXAMM uses islands, which have been shown to potentially provide superlinear speedup on some EAs [2].

- The master process keeps separate "island" populations and performs crossover within islands (intra-island crossover) or crossover between islands (inter-island crossover).

[2] Enrique Alba and Marco Tomassini. 2002. **Parallelism and evolutionary algorithms.** *IEEE Transactions on Evolutionary Computation:* 6, 5 (2002), 443–462.

Islands



- When workers request individuals, the master process generates them from an island in a round-robin manner.
- Individuals are inserted into an island if they are better than the worst individual in that island (and the worst is removed) - Individual islands evolve/speciate faster.
- Periodically crossover happens between islands for most fit individuals, sharing information (a random individual on an island is crossed over with the best individual from another island).

Data Sets

Data Sets

- Two large-scale, real-world data from Aviation and Power industries used to evaluate EXAMM.

- 10 flights from the National General Aviation Flight Information Database (NGAFID):
  - 1-3 hours long
  - per second readings
  - 26 parameters

- 12 coal plant burners from a DOE award with Microbeam Technologies, Inc.
  - 10 days long
  - per minute readings
  - 12 parameters

Data Sets: Coal Plant

12 data files, 12 parameters:

1. Conditioner Inlet Temp
2. Conditioner Outlet Temp
3. Coal Feeder Rate
4. Primary Air Flow
5. Primary Air Split
6. System Secondary Air Flow Total
7. Secondary Air Flow
8. Secondary Air Split
9. Tertiary Air Split
10. Total Combined Air Flow
11. **Supplementary Fuel Flow**
12. **Main Flame Intensity**

- Parameters are non-seasonal and correlated/dependent.
- Predicting Fuel Flow and Flame Intensity
- Data made public on github repo. Pre-normalized and anonymized.

Data Sets: NGAFID

10 data files, 26 parameters:

1. Altitude Above Ground Level (AltAGL)
2. Engine 1 Cylinder Head Temperature 1 (E1 CHT1)
3. Engine 1 Cylinder Head Temperature 2 (E1 CHT2)
4. Engine 1 Cylinder Head Temperature 3 (E1 CHT3)
5. Engine 1 Cylinder Head Temperature 4 (E1 CHT4)
6. Engine 1 Exhaust Gas Temperature 1 (E1 EGT1)
7. Engine 1 Exhaust Gas Temperature 2 (E1 EGT2)
8. Engine 1 Exhaust Gas Temperature 3 (E1 EGT3)
9. Engine 1 Exhaust Gas Temperature 4 (E1 EGT4)
10. Engine 1 Oil Pressure (E1 OilP)
11. Engine 1 Oil Temperature (E1 OilT)
12. **Engine 1 Rotations Per minute (E1 RPM)**
13. Fuel Quantity Left (FQtyL)
14. Fuel Quantity Right (FQtyR)
15. GndSpd - Ground Speed (GndSpd)
16. Indicated Air Speed (IAS)
17. Lateral Acceleration (LatAc)
18. Normal Acceleration (NormAc)
19. Outside Air Temperature (OAT)
20. **Pitch**
21. Roll
22. True Airspeed (TAS)
23. Voltage 1 (volt1)
24. Voltage 2 (volt2)
25. Vertical Speed (VSpd)
26. Vertical Speed Gs (VSpdG)

- Parameters are non-seasonal and correlated/dependent.
- Predicting RPM and Pitch
- Data made public on github repo. Non-normalized and anonymized.

Results

Computing Environment

- RIT Research Computing systems used to gather results.
- Compute nodes heterogeneous:
  - 10 core 2.3 GHz Intel Xeon CPU E5-2650 v3
  - 32 core 2.6 GHz AMD Opteron Processor 6282 SE
  - 48 core 2.5 GHz AMD Opteron Processor 6180 SEs
- All compute nodes ran RedHat Enterprise Linux 6.10.
- EXAMM runs utilized different compute notes as determined by RC's SLURM scheduler.

EXAMM Experimental Setup

- EXAMM run with individual memory cells, individual memory cells + simple neurons, and with all memory cells and simple neurons.

- Memory cell types:
  - Delta-RNN
  - GRU
  - LSTM
  - MGU
  - UGRNN

- K-fold cross validation (2 files per fold), 10 repeats per fold, 2 output parameters (RPM, Pitch) on NGAFID data - 1100 runs.
- K-fold cross validation (2 files per fold), 10 repeats per fold, 2 output parameters (Flame Intensity, Fuel Flow) on Coal Data - 1320 runs.

- EXALT trained 2000 RNNs for 10 epochs each, distributed across 20 processes.
- 4,840,000 RNNs trained in total in ~24,200 CPU hours

Flame Intensity

Flame Intensity

Best Case		**Avg. Case**		**Worst Case**	
Δ-RNN	-0.92312	Δ-RNN+simple	-1.7775	all	-1.5404
Δ-RNN+simple	-0.90534	LSTM+simple	-1.7148	LSTM+simple	-1.1066
all	-0.71602	MGU+simple	-0.53749	MGU+simple	-1.1026
UGRNN+simple	-0.71451	Δ-RNN	-0.026901	MGU	-0.59787
LSTM	-0.46836	GRU	0.18143	GRU	-0.33703
LSTM+simple	-0.42565	UGRNN	0.19272	Δ-RNN	0.035348
GRU	-0.10578	GRU+simple	0.30281	LSTM	0.61246
MGU+simple	0.31264	all	0.42371	delta+simple	0.69439
UGRNN	0.31964	UGRNN+simple	0.49785	UGRNN	0.9569
MGU	1.5708	LSTM	1.2196	GRU+simple	0.97318
GRU+simple	2.0557	MGU	1.2386	UGRNN+simple	1.4123

- Rankings (deviations from mean) for flame intensity predictions. Lower values (higher on the chart) is better.

Fuel Flow

Fuel flow

Best Case		Avg. Case		Worst Case	
all	-0.92643	LSTM	-1.4415	LSTM+simple	-1.2349
UGRNN+simple	-0.7644	Δ-RNN+simple	-1.2172	LSTM	-1.0818
LSTM	-0.70271	MGU+simple	-1.1255	Δ-RNN+simple	-1.014
UGRNN	-0.66396	GRU+simple	-0.25195	MGU	-0.27097
Δ-RNN	-0.58832	LSTM+simple	-0.23921	MGU+simple	-0.1799
MGU+simple	-0.41037	GRU	-0.14222	GRU+simple	-0.16598
Δ-RNN+simple	-0.22068	all	0.22163	GRU	0.087564
LSTM+simple	-0.1125	MGU	0.44679	all	0.23284
GRU+simple	0.85882	Δ-RNN	0.99531	UGRNN+simple	0.58938
GRU	1.5692	UGRNN+simple	1.3537	Δ-RNN	0.77052
MGU	1.9613	UGRNN	1.4002	UGRNN	2.2672

- Rankings (deviations from mean) for flame intensity predictions. Lower values (higher on the chart) is better.

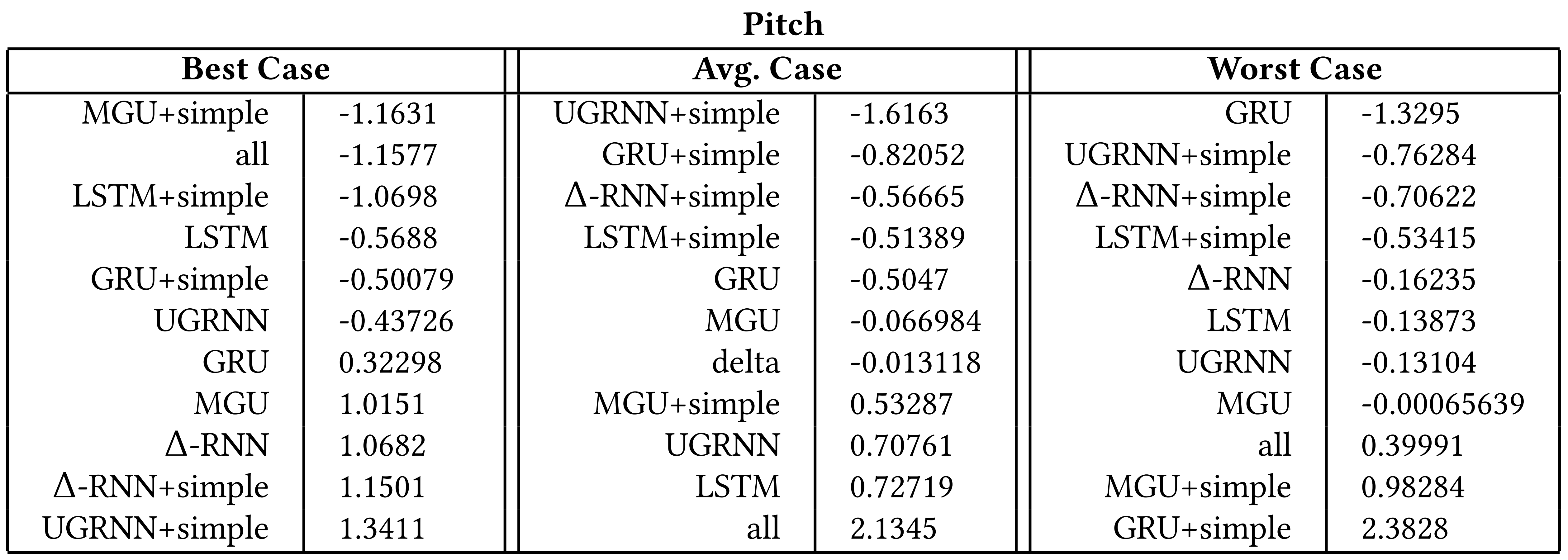
RPM

RPM

Best Case		**Avg. Case**		**Worst Case**	
GRU	-1.444	LSTM+simple	-1.7472	GRU	-1.0958
MGU+simple	-1.1012	MGU+simple	-1.2299	LSTM+simple	-1.0499
Δ-RNN	-1.0347	Δ-RNN	-1.0081	Δ-RNN+simple	-0.87687
LSTM+simple	-0.52825	GRU	-0.4433	MGU+simple	-0.78566
Δ-RNN+simple	-0.29348	Δ-RNN+simple	-0.069508	UGRNN	-0.59783
UGRNN	-0.076276	GRU+simple	0.050686	UGRNN+simple	-0.19645
LSTM	0.18368	UGRNN	0.52115	GRU+simple	0.16258
MGU	0.50967	all	0.76179	Δ-RNN	0.41787
UGRNN+simple	0.9463	MGU	0.93224	all	1.0968
GRU+simple	1.271	LSTM	1.0852	LSTM	1.1219
all	1.5672	UGRNN+simple	1.147	MGU	1.8033

- Rankings (deviations from mean) for flame intensity predictions. Lower values (higher on the chart) is better.

Pitch



Pitch

Best Case		**Avg. Case**		**Worst Case**	
MGU+simple	-1.1631	UGRNN+simple	-1.6163	GRU	-1.3295
all	-1.1577	GRU+simple	-0.82052	UGRNN+simple	-0.76284
LSTM+simple	-1.0698	Δ-RNN+simple	-0.56665	Δ-RNN+simple	-0.70622
LSTM	-0.5688	LSTM+simple	-0.51389	LSTM+simple	-0.53415
GRU+simple	-0.50079	GRU	-0.5047	Δ-RNN	-0.16235
UGRNN	-0.43726	MGU	-0.066984	LSTM	-0.13873
GRU	0.32298	delta	-0.013118	UGRNN	-0.13104
MGU	1.0151	MGU+simple	0.53287	MGU	-0.00065639
Δ-RNN	1.0682	UGRNN	0.70761	all	0.39991
Δ-RNN+simple	1.1501	LSTM	0.72719	MGU+simple	0.98284
UGRNN+simple	1.3411	all	2.1345	GRU+simple	2.3828

- Rankings (deviations from mean) for flame intensity predictions. Lower values (higher on the chart) is better.

Overall Rankings

Overall Combined

Best Case		Avg. Case		Worst Case	
MGU+simple	-0.59051	LSTM+simple	-1.0538	LSTM+simple	-0.98141
LSTM+simple	-0.53405	Δ-RNN+simple	-0.90771	GRU	-0.6687
LSTM	-0.38905	MGU+simple	-0.59001	Δ-RNN+simple	-0.47566
Δ-RNN	-0.36948	GRU	-0.2272	MGU+simple	-0.27133
all	-0.30824	GRU+simple	-0.17974	all	0.047292
UGRNN	-0.21446	Δ-RNN	-0.013211	LSTM	0.12847
Δ-RNN+simple	-0.067358	UGRNN+simple	0.34556	MGU	0.23345
GRU	0.085614	LSTM	0.39761	UGRNN+simple	0.26059
UGRNN+simple	0.20212	MGU	0.63765	Δ-RNN	0.26535
GRU+simple	0.9212	UGRNN	0.70542	UGRNN	0.62381
MGU	1.2642	all	0.88541	GRU+simple	0.83814

- Combined rankings from all 4 prediction parameters.

EXAMM Results

- **No memory structure was the best.**
- Delta-RNN, LSTM, and MGU tended better than GRU, and UGRNN (except in fuel flow, best avg case for pitch).
- Delta-RNNs compared competitively with LSTMs while requiring less weights (i.e., a less complex structure).

EXAMM Results

- **Allowing all memory cells has risks and benefits.**
- All cell types + simple neurons found the best networks in the case of fuel flow, 2nd best in the case of pitch, and 3rd best in the case of flame intensity.
- Using all memory cell types generally performed better than the mean on the best case, however performed worse in the average and worst cases.
- Allowing EXAMM to select from all possible memory cells was not entirely a bad strategy. In many cases it found the best but on average it did not do as well (most likely due to it having a much larger search space - tweaking hyperparameters for more exploration could improve this).
- **Open Question:** Can we further improve results by dynamically adapting the rates at which memory cells are generated?

EXAMM Results

- **Adding simple neurons generally helped - with some notable exceptions.**
- All memory cell types improved with them **except** GRU.
- Simple neurons + MGUs resulted in dramatic improvement, bringing them from some of the worst rankings to some of the best rankings (e.g., in the overall rankings for best found networks, MGU cells alone performed the worst while MGU and feedforward performed the best).

- Other cell types (LSTM and Delta-RNN) showed less of an improvement.

- This finding may highlight that the MGU cells could stand to benefit from further development.

- Even the rather simple change of allowing simple neurons can result in significant changes in RNN predictive ability. Selection of node and cell types for neuro-evolution should be done carefully.

- **Open question:** Why do GRU cells performed worse with simple neurons added? Why do MGU cells perform so much better?

EXAMM Results

- **Larger networks tended to perform better, yet memory cell count correlation to MSE was not a great indicator of which cells performed the best.**

- Challenges for developing neuro-evolution algorithms:
  - Compared the memory cells types most correlated to improved performance against the memory cell types most frequently selected by EXAMM.
  - EXAMM was not selecting cell types that would produce the best performing RNNs, rather cell types that provided an improvement to the population (most did) -- this can be a non-optimal choice.
  - An RNN with a small number of well trained memory cells was sufficient to yield good predictions, and adding more cells to the network only served to confuse the predictions.

- **Open problems:**
  - Running a neuro-evolution strategy allowing all memory cell types and then utilizing counts or correlations to select a single memory cell type for future runs may not produce the best results.
  - Dynamically tuning which memory cells are selected by a neuro-evolution strategy is more challenging since the process may not select the best cell types (e.g., when the network already has enough memory cells) – so this would at least need to be coupled with another strategy to determine when the network is “big enough”.

Future Work

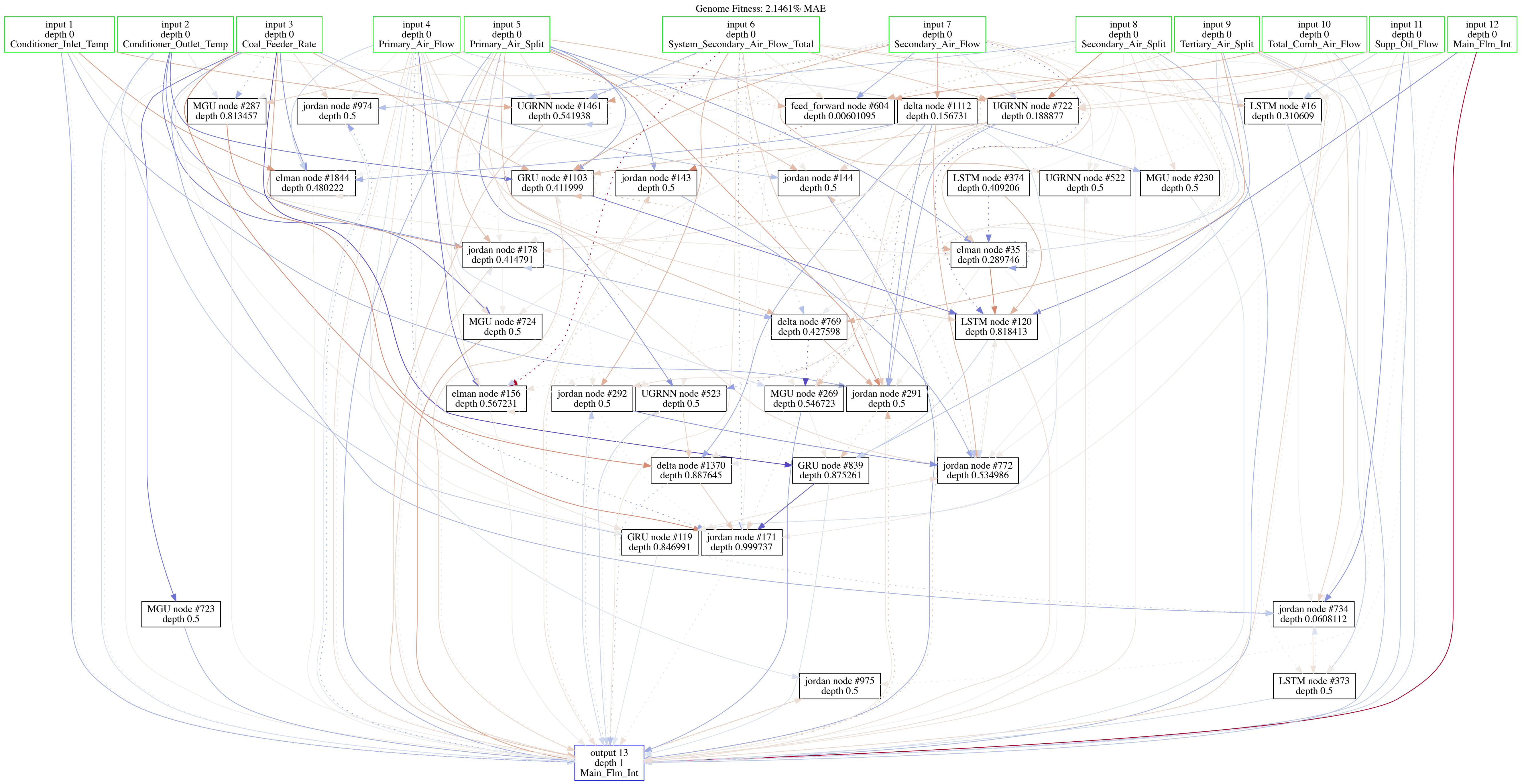
- Evolving memory cell structures.
- Allowing multiple activation functions (tanh mostly used in this work).
- Hyperparameter optimization for RNN training.
- Layer-level mutations to speed evolution
- Self-tuning EXAMM.
- EXAMM for transfer learning: take a pre-evolved/pre-trained network and evolve it to new problems.

- Use our findings to examine and improve existing RNN memory cell structures.

Discussion/Questions?

https://github.com/travisdesell/exact

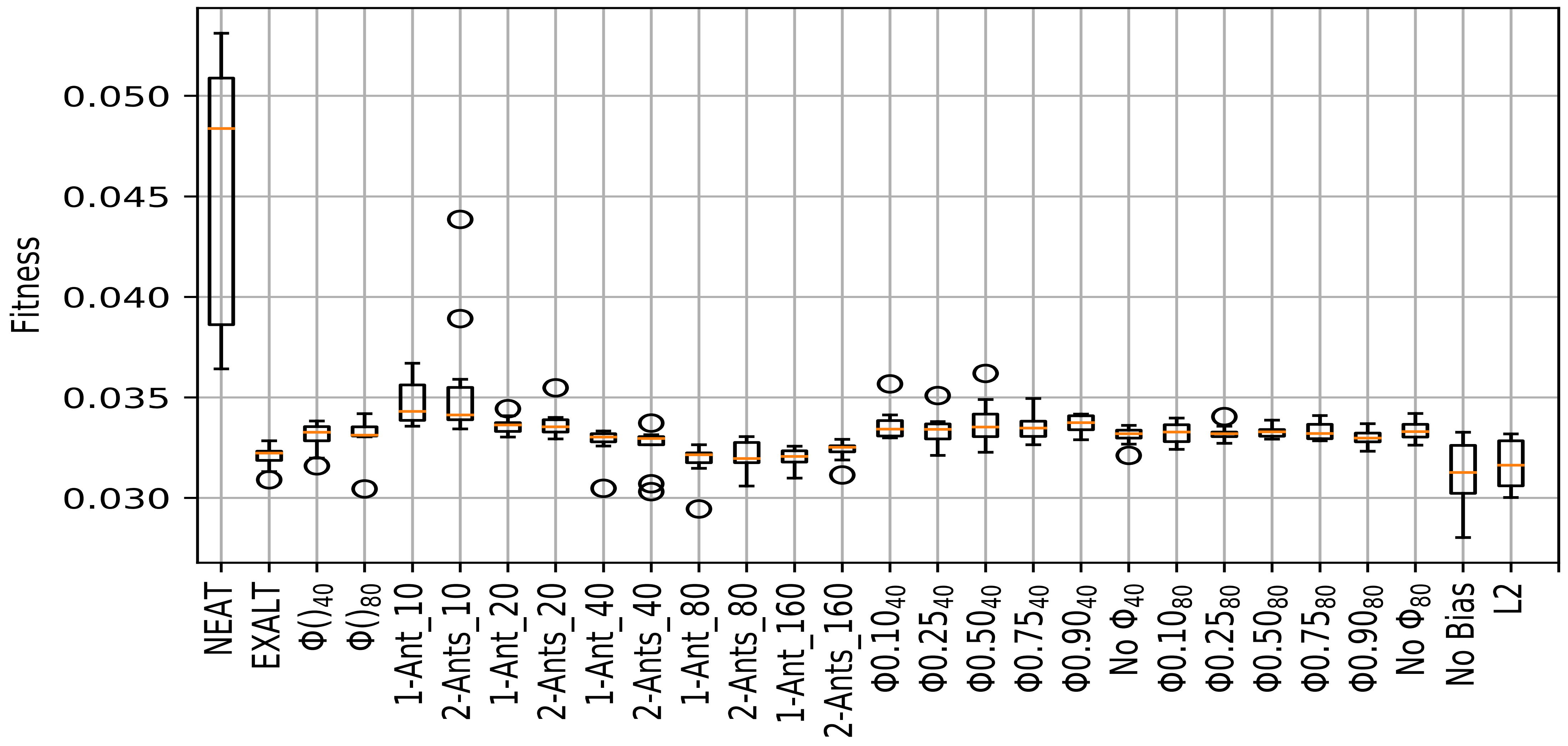
Example Network


Genome Fitness: 2.1461% MAE
input 1 depth 0 Conditioner_Inlet_Temp
input 2 depth 0 Conditioner_Outlet_Temp
input 3 depth 0 Coal_Feeder_Rate
input 4 depth 0 Primary_Air_Flow
input 5 depth 0 Primary_Air_Split
input 6 depth 0 System_Secondary_Air_Flow_Total
input 7 depth 0 Secondary_Air_Flow
input 8 depth 0 Secondary_Air_Split
input 9 depth 0 Tertiary_Air_Split
input 10 depth 0 Total_Comb_Air_Flow
input 11 depth 0 Supp_Oil_Flow
input 12 depth 0 Main_Flm_Int
MGU node #287 depth 0.813457
jordan node #974 depth 0.5
UGRNN node #1461 depth 0.541938
feed_forward node #604 depth 0.00601095
delta node #1112 depth 0.156731
UGRNN node #722 depth 0.188877
LSTM node #16 depth 0.310609
elman node #1844 depth 0.480222
GRU node #1103 depth 0.411999
jordan node #143 depth 0.5
jordan node #144 depth 0.5
LSTM node #374 depth 0.409206
UGRNN node #522 depth 0.5
MGU node #230 depth 0.5
jordan node #178 depth 0.414791
elman node #35 depth 0.289746
MGU node #724 depth 0.5
delta node #769 depth 0.427598
LSTM node #120 depth 0.818413
elman node #156 depth 0.567231
jordan node #292 depth 0.5
UGRNN node #523 depth 0.5
MGU node #269 depth 0.546723
jordan node #291 depth 0.5
delta node #1370 depth 0.887645
GRU node #839 depth 0.875261
jordan node #772 depth 0.534986
GRU node #119 depth 0.846991
jordan node #171 depth 0.999737
MGU node #723 depth 0.5
jordan node #734 depth 0.0608112
jordan node #975 depth 0.5
LSTM node #373 depth 0.5
output 13 depth 1 Main_Flm_Int

RIT | Rochester Institute of Technology

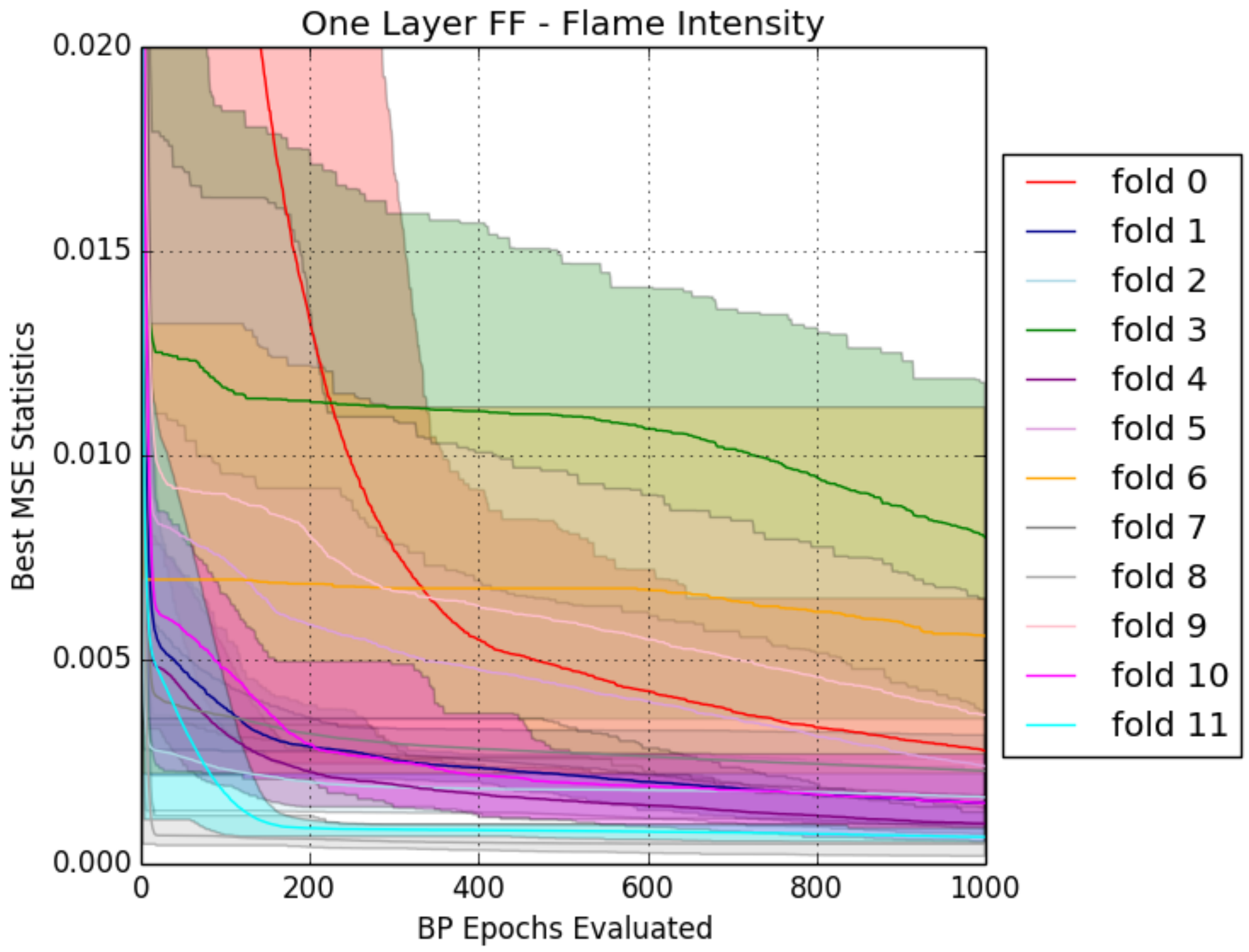
The 2019 Genetic and Evolutionary Computation Conference
Prague, Czech Republic
July 13th-17th, 2019

EXAMM vs. NEAT vs. ACO



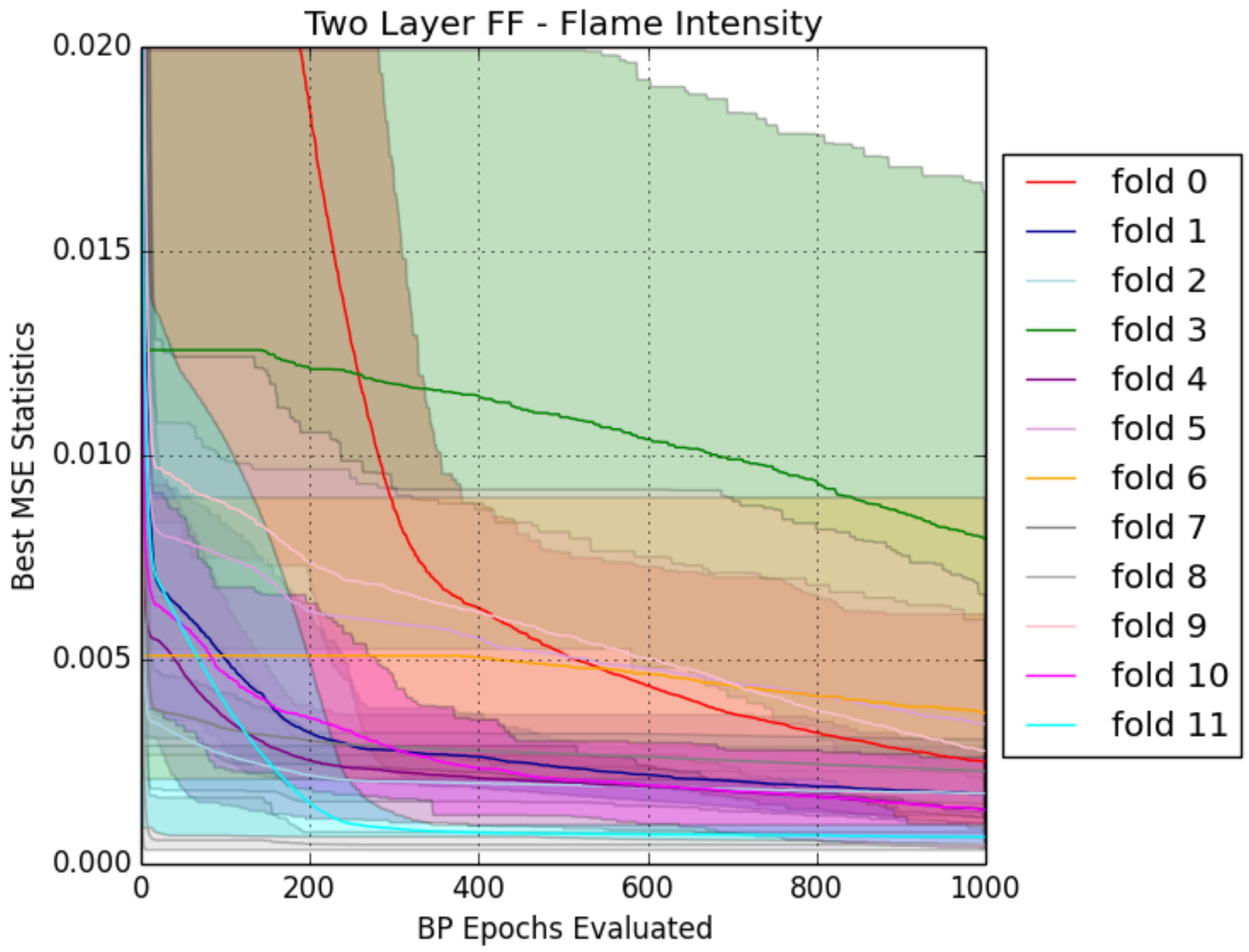
- Recent results comparing NEAT to EXAMM/EXALT on flame intensity data set.

1 Layer Feed Forward



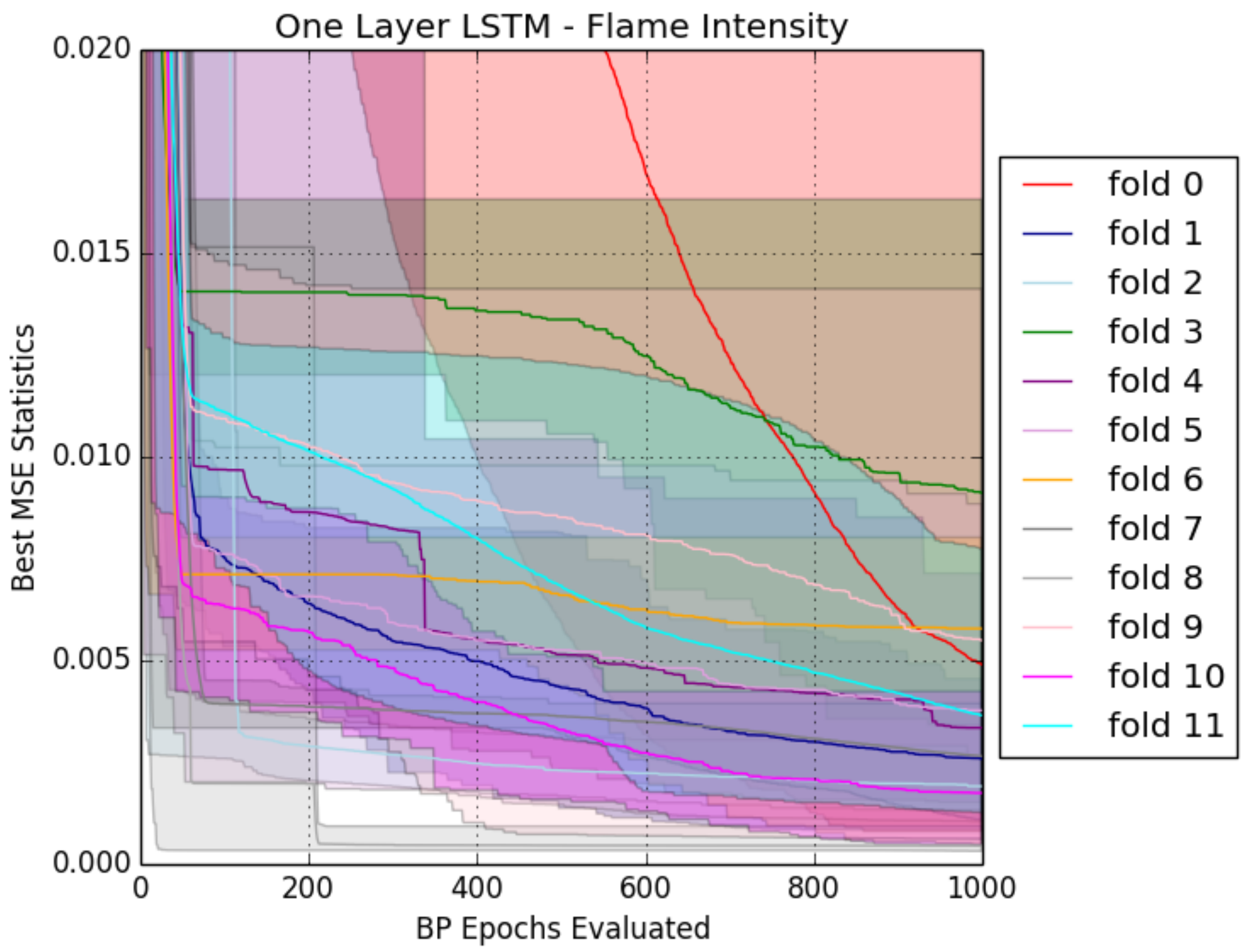
- Min/avg/max mean squared error while training for each fold.

2 Layer Feed Forward



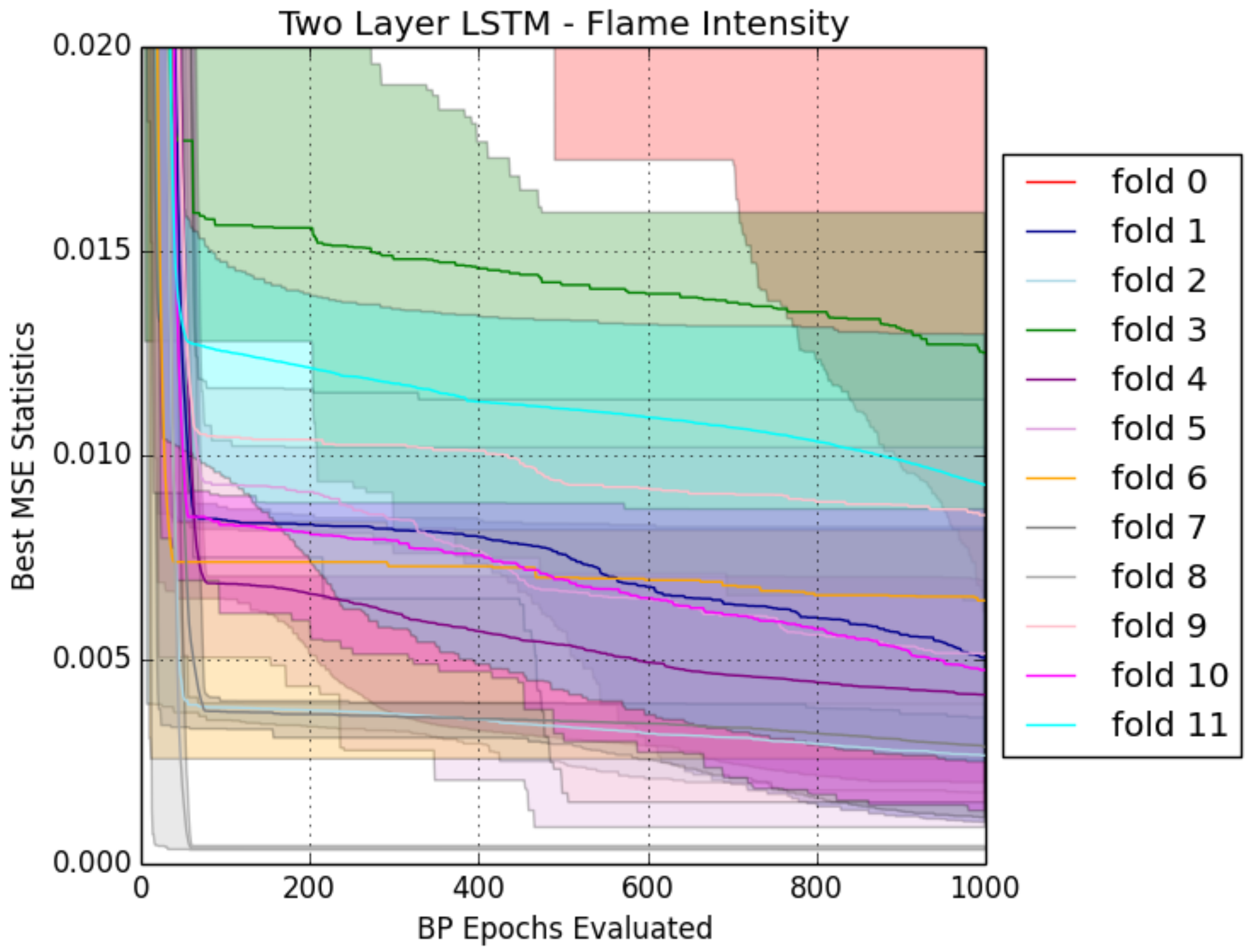
- Min/avg/max mean squared error while training for each fold.

1 Layer LSTM



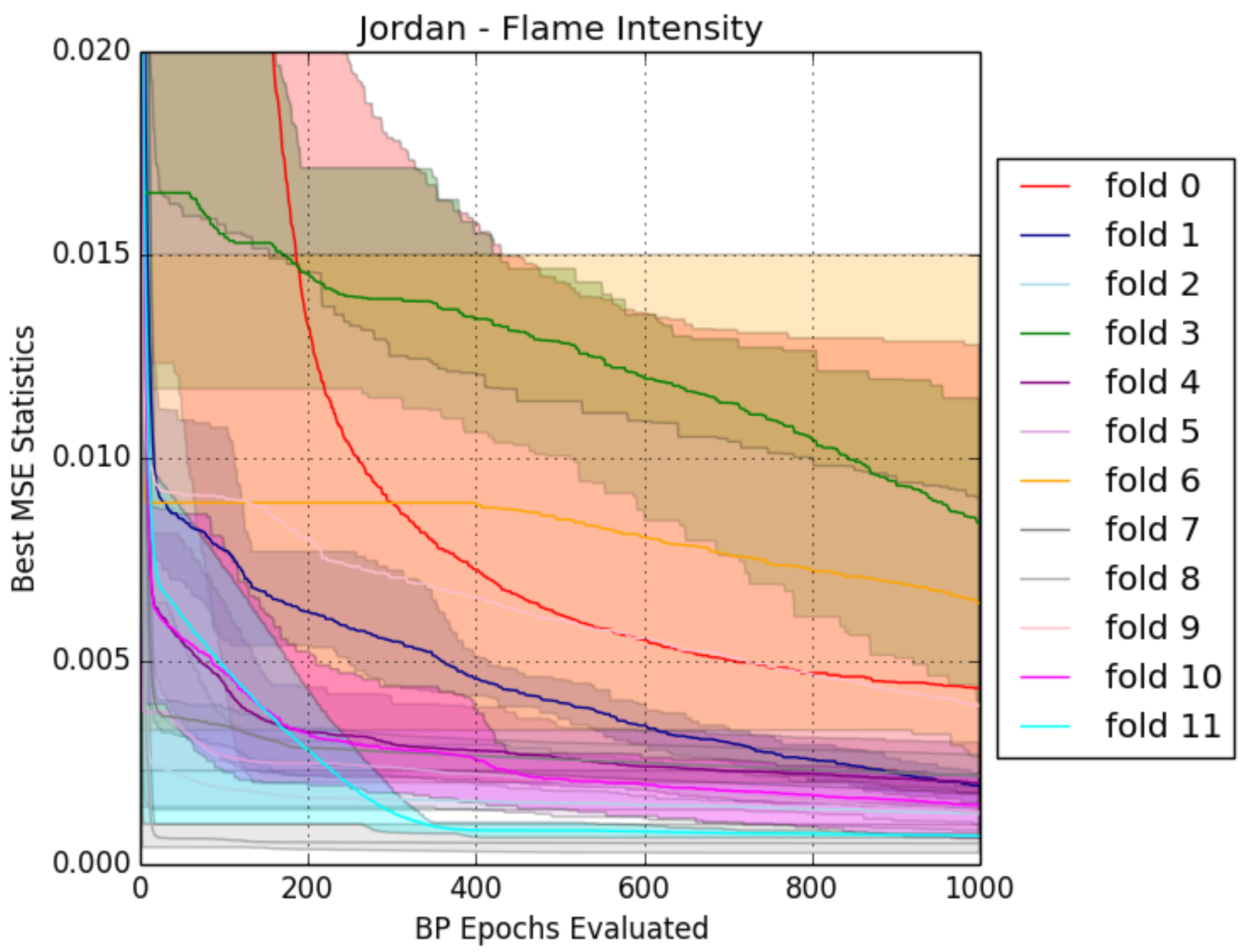
- Min/avg/max mean squared error while training for each fold.

2 Layer LSTM



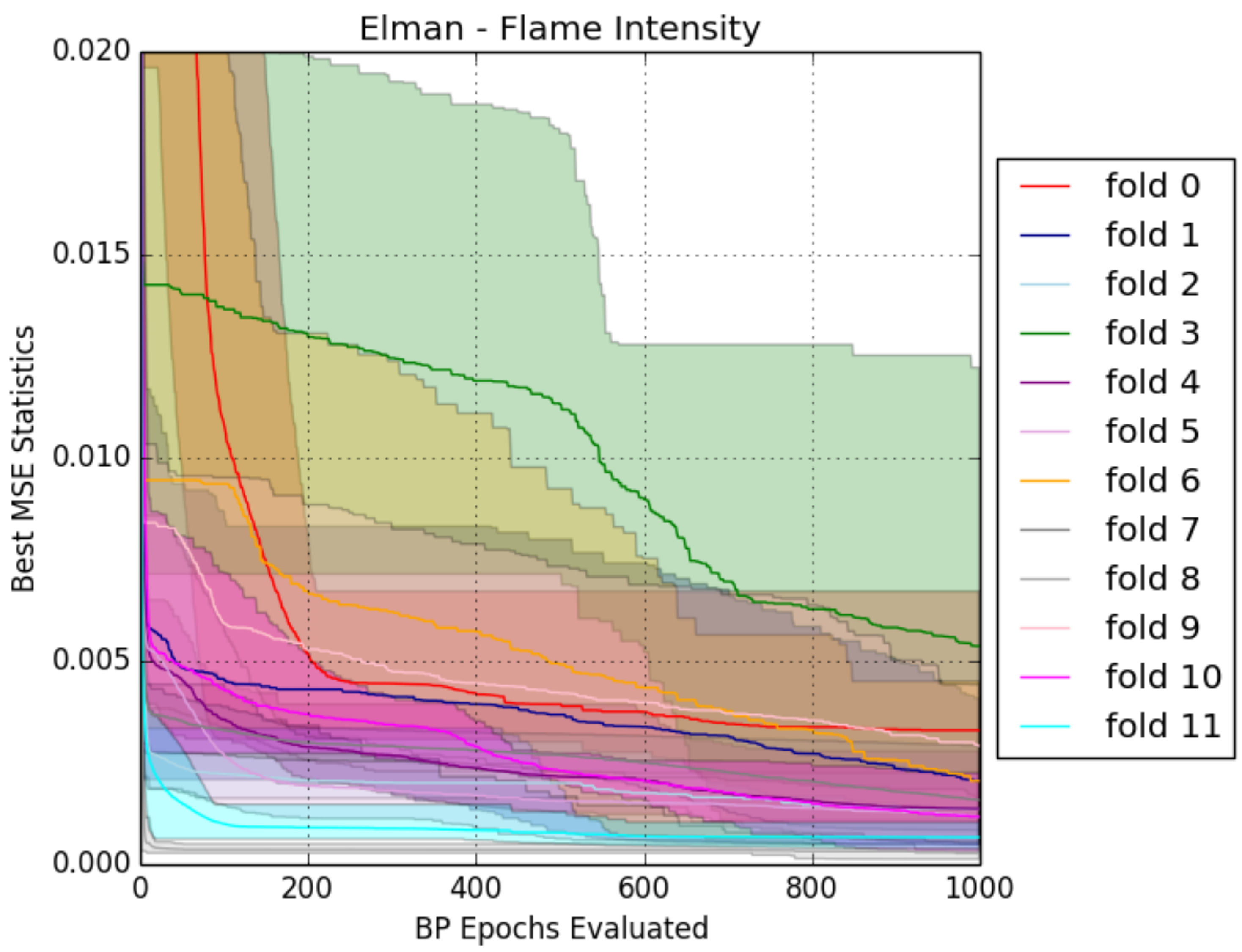
- Min/avg/max mean squared error while training for each fold.

Jordan



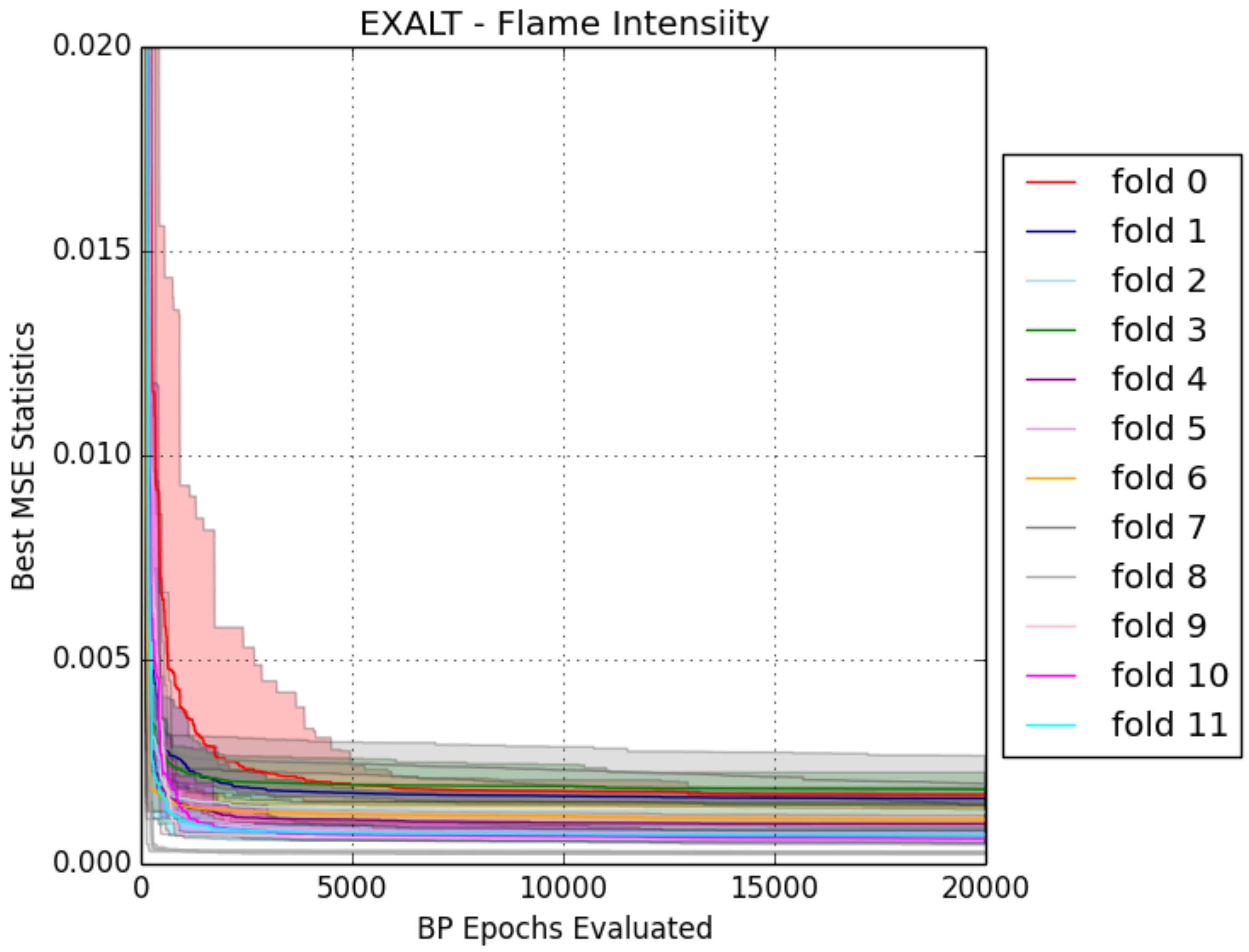
- Min/avg/max mean squared error while training for each fold.

Elman



- Min/avg/max mean squared error while training for each fold.

EXALT



- Min/avg/max mean squared error while training for each fold.

EXALT Results

	Nodes	Edges	Rec. Edges	Weights
One Layer FF	25	156	0	181
Two Layer FF	37	300	0	337
Jordan RNN	25	156	12	193
Elman RNN	25	156	144	325
One Layer LSTM	25	156	0	311
Two Layer LSTM	37	300	0	587
EXALT Best Avg.	14.7	26.2	14.6	81.5

- Significantly more reliable than the fixed architectures.
- Wallclock time was **faster** in terms of training time, 2-10x faster than the fixed RNNs.
- EXALT's RNNs were smaller (see above).

- However, some of the fixed RNNs did find slightly better performance in the best case across all the repeats.

AbdElRahman ElSaid, Steven Benson, Shuchita Patwardhan, David Stadem and Travis Desell. 2019. **Evolving Recurrent Neural Networks for Time Series Data Prediction of Coal Plant Parameters.** *In The 22nd International Conference on the Applications of Evolutionary Computation.* Leipzig, Germany. April 22-24, 2019. **To appear.**