Investigating Recurrent Neural Network Memory Structures using Neuro-Evolution

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

Collaborators:
AbdElRahman ElSaid (PhD GRA)
Alex Ororbia (GCCIS Computure Science)
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
- EXALT and EXAMM:
  - NEAT Innovations
  - Edge and Node Mutations
  - Crossover
  - Distributed Neuro-Evolution
- Results
  - Performance vs. Traditional
  - EXALT Results
  - EXAMM Results
- Future Work
- Discussion

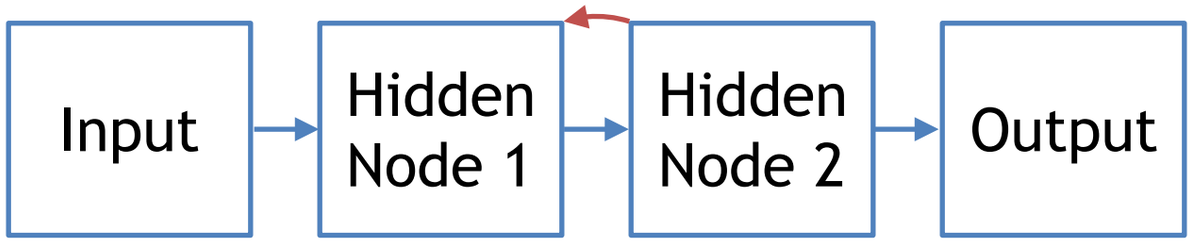
Motivation

What is Neuro-Evolution?

- Most people use human-designed ANNs, selecting from a few architectures that have done will in the literature.
- No guarantees these are most optimal.
- 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)

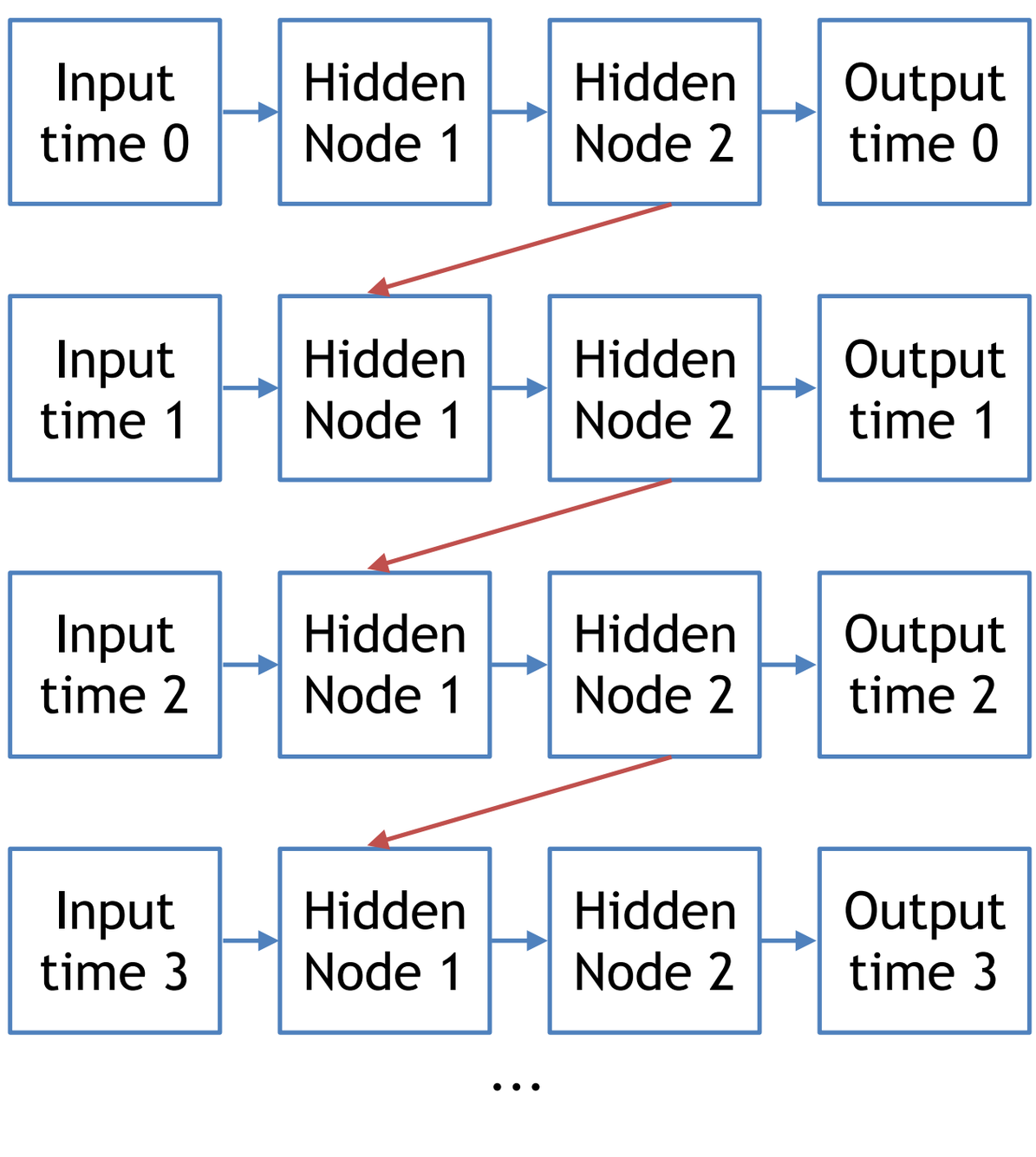
Background

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.



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.*

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

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

Evolutionary Algorithms in a Nutshell

Evolutionary algorithms contain a population of individuals (potential problem solutions), each with a fitness (a representation of how well they perform on the task).

They progressively generate new populations through mutation, recombination and selection operations.

- *Mutation:* Select a single individual as a parent and perform a modification to it to create a child individual.
- *Recombination:* Select two (or more!) individuals as parents and combine them to create a child individual.
- *Selection:* Retain a fit individual in the population to preserve best found solutions.

EXALT and EXAMM

RIT | Rochester Institute of Technology

CHAI AI Seminar
February 11, 2019

EXALT and EXAMM

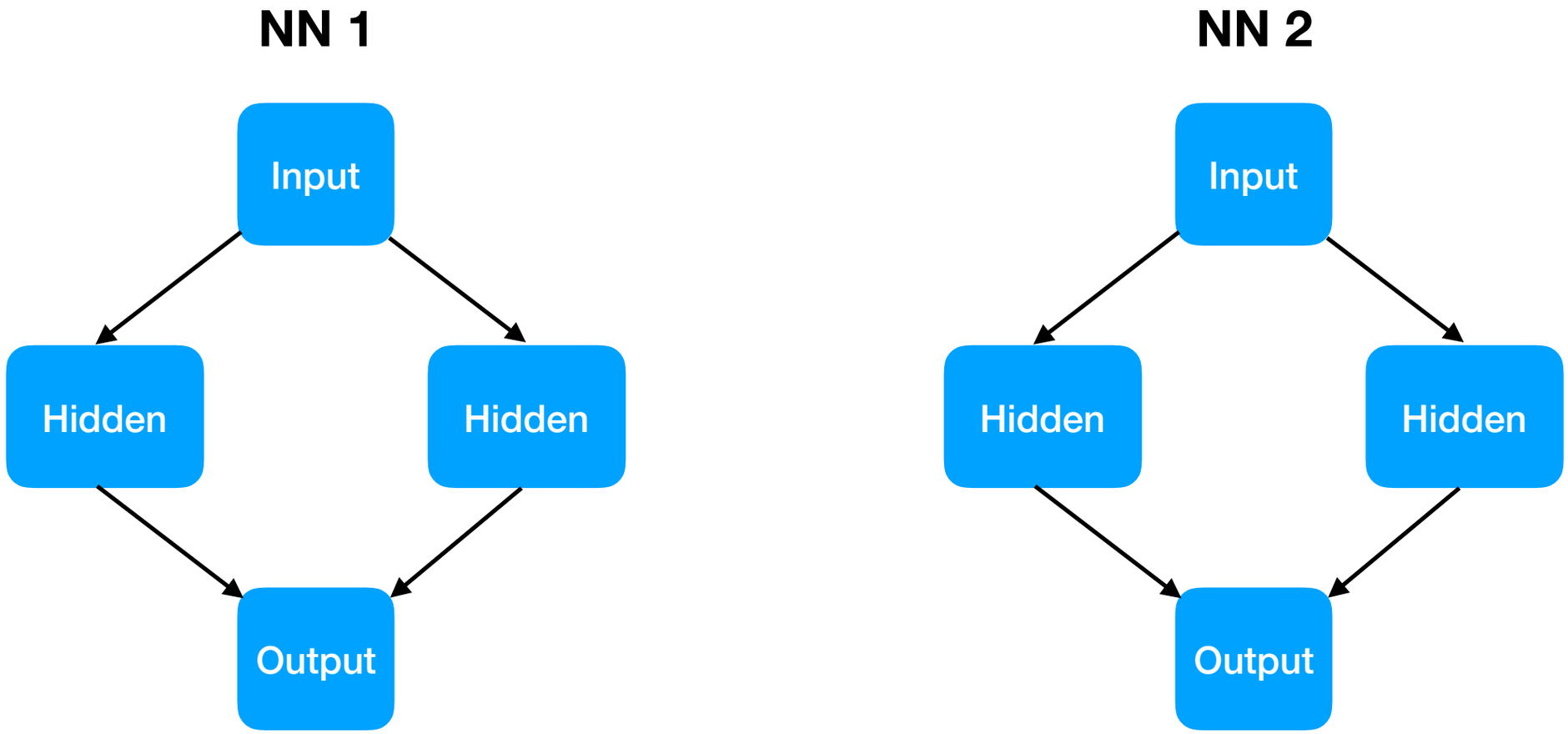
- Neuro-Evolution algorithms based of Neuro-Evolution of Augmenting Topologies (NEAT) [1].

- Evolutionary Exploration of Augmenting LSTM Topologies (EXALT):
  - Progressively grows RNNs: nodes can be simple neurons or LSTMs.
  - Parallel in nature.
  - Node-level mutations not present in NEAT.

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

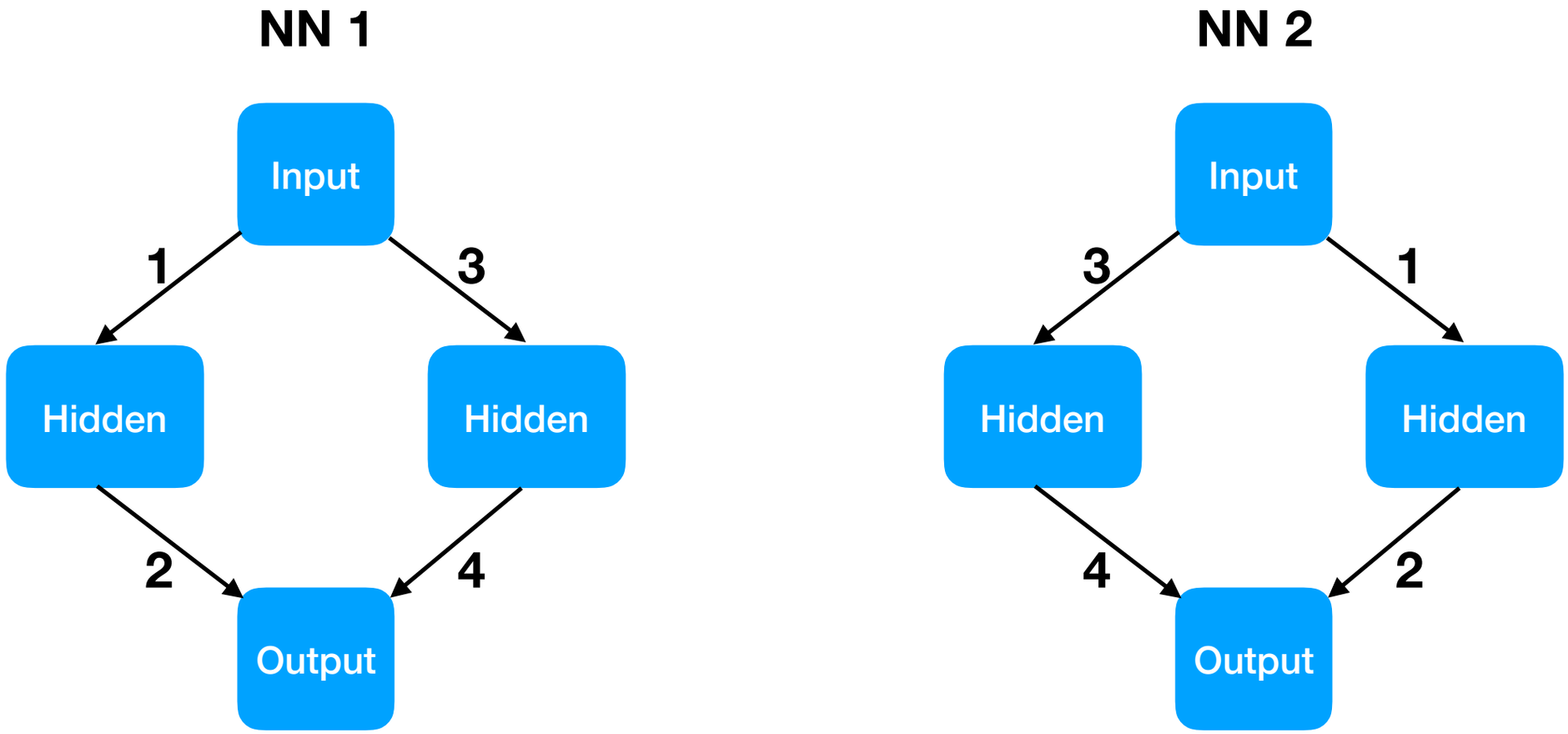
[1] Kenneth Stanley and Risto Miikkulainen. **Evolving neural networks through augmenting topologies.** *Evolutionary computation: 10, 2*. (2002), 99–127.

NEAT Innovation Numbers



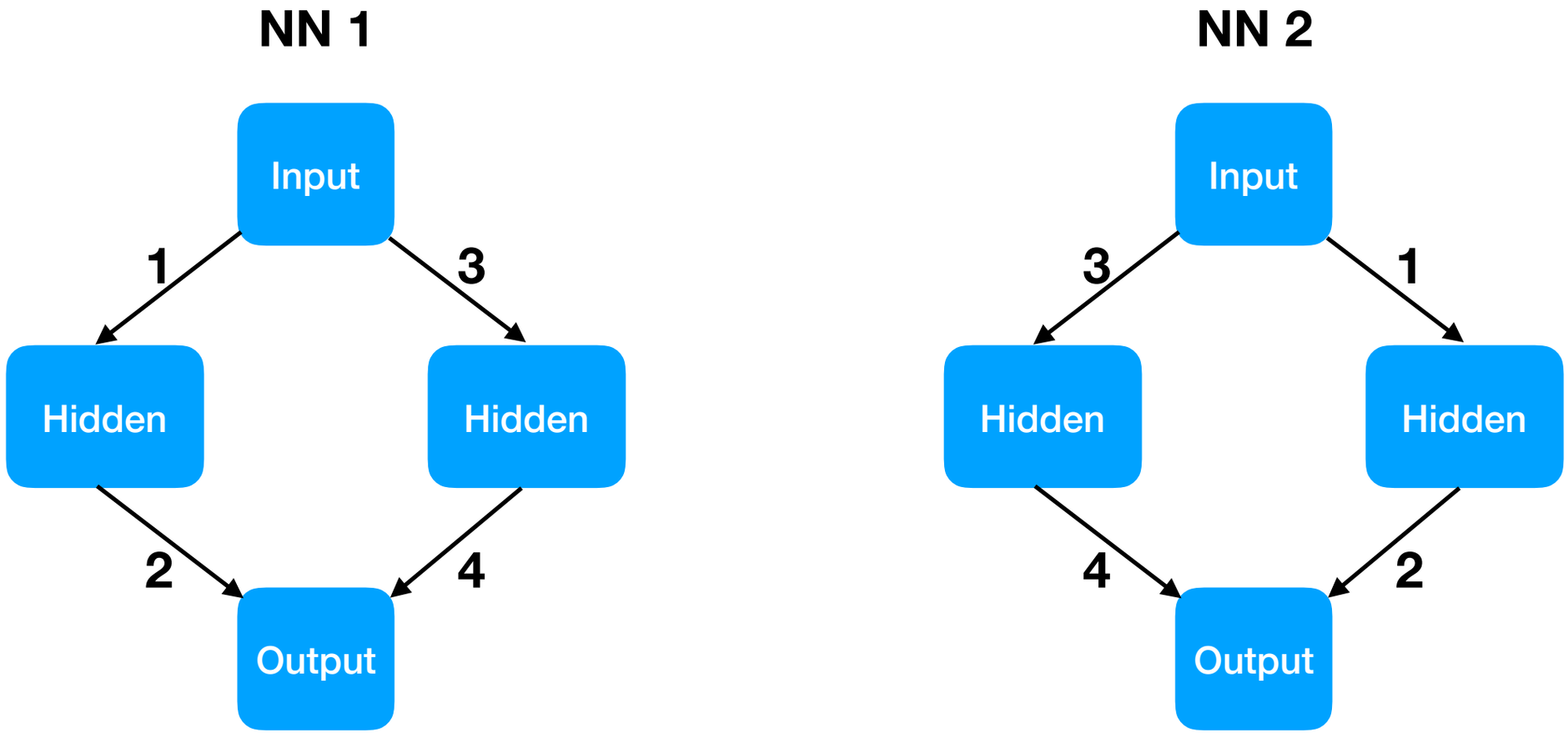
- In neuro-evolution, we need to perform crossover/ recombination between progressively grown neural networks.
- How do we know which edges are the "same" in the above neural networks?

NEAT Innovation Numbers



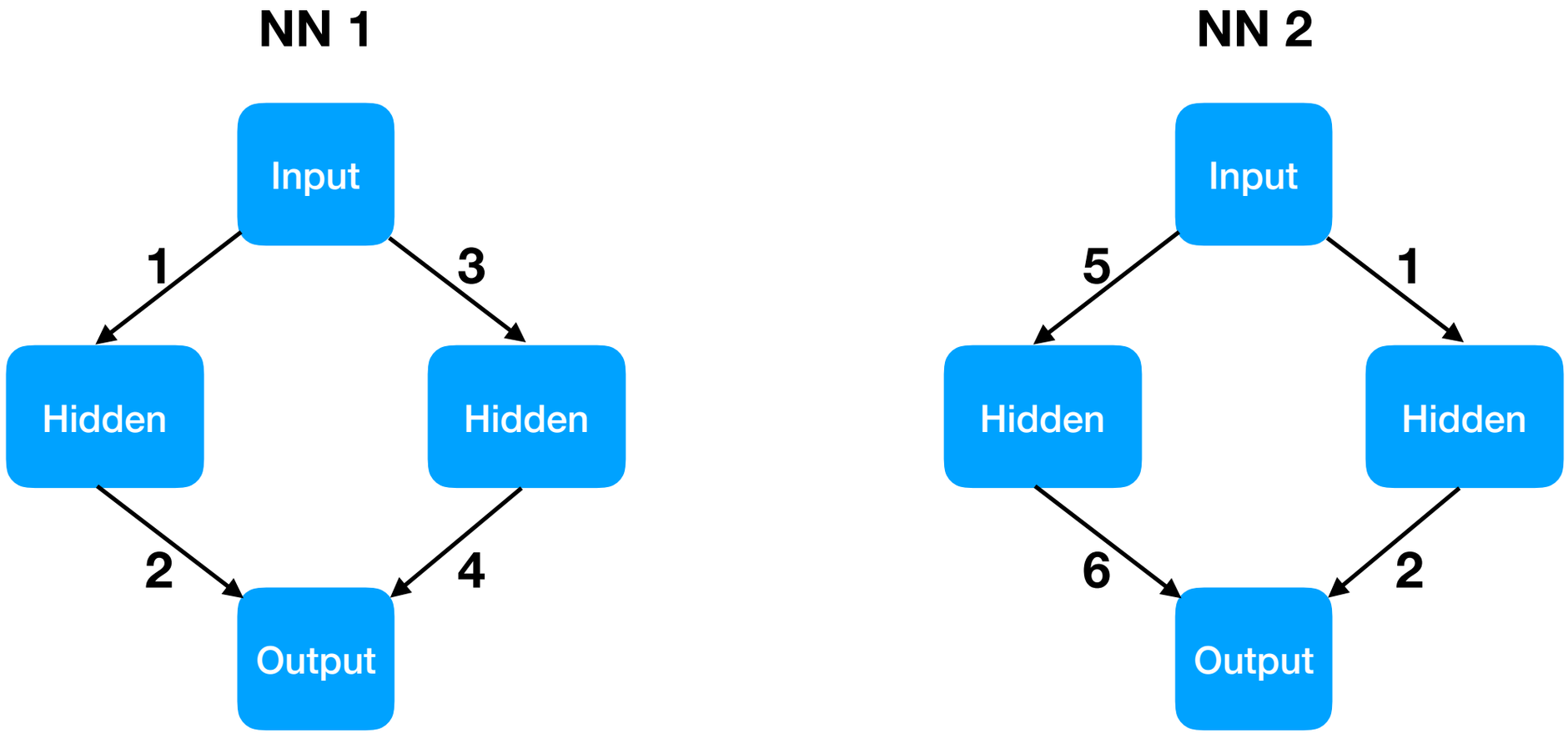
- NEAT assigns a unique "innovation number" to each newly generated edge.
- This allows NN graphs to be compared in linear (assuming edges are sorted according to innovation numbers) time - otherwise NN graphs could be ambiguous and very computationally expensive to compare.

NEAT Innovation Numbers



- In the above example, the edges on the left of NN 1 correspond to the same edges on the right of NN 2.

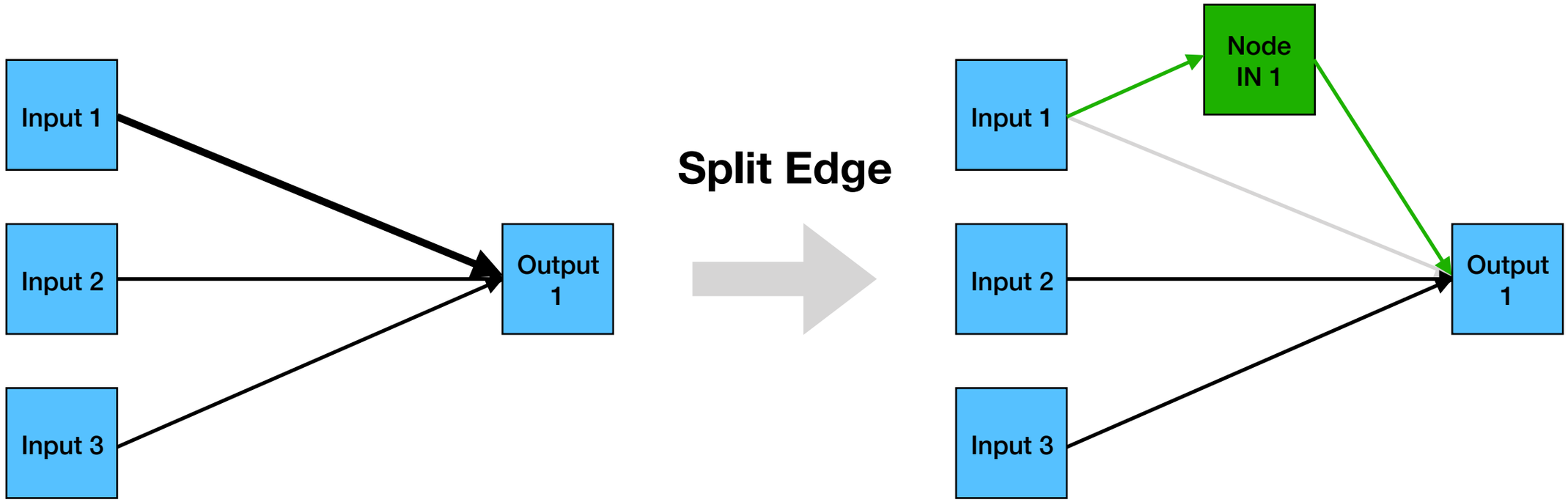
NEAT Innovation Numbers



- However, a similar structure may have been generated evolutionarily with "different" edges - in this case they will have different innovation numbers.
- This way we know the edges on the right of NN 2 are the same as those on the left of NN 1, but the other edges occurred through a different evolutionary process and should be treated differently.

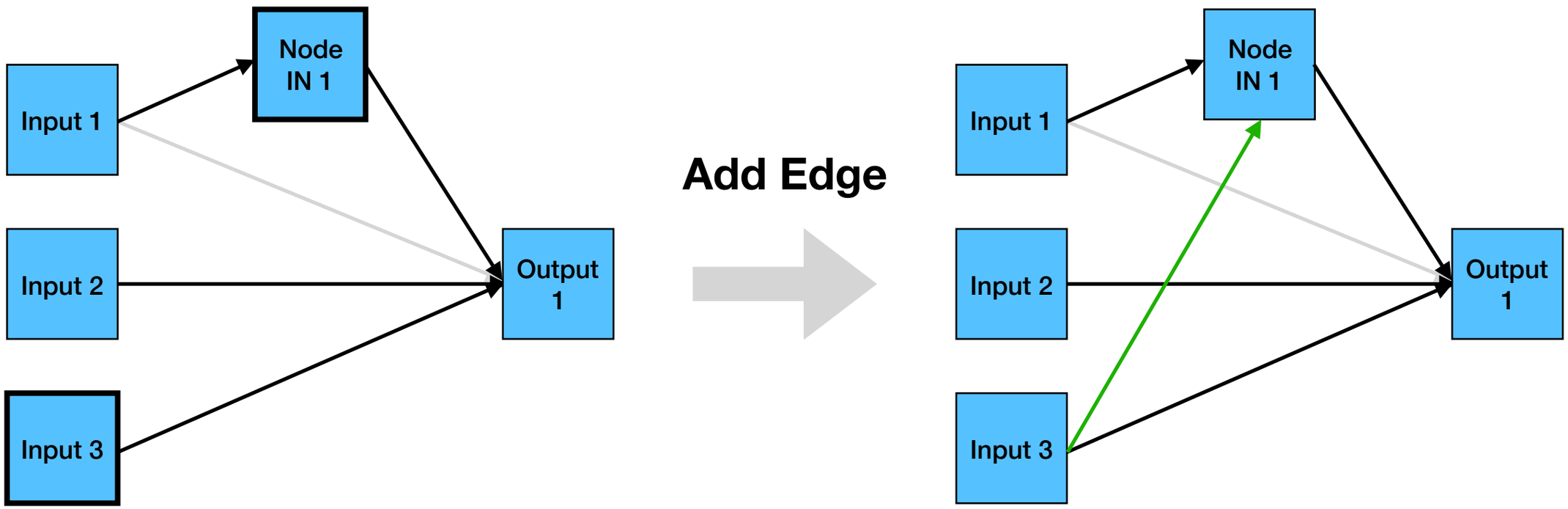
Edge and Node Mutations

Edge Mutations: Split Edge



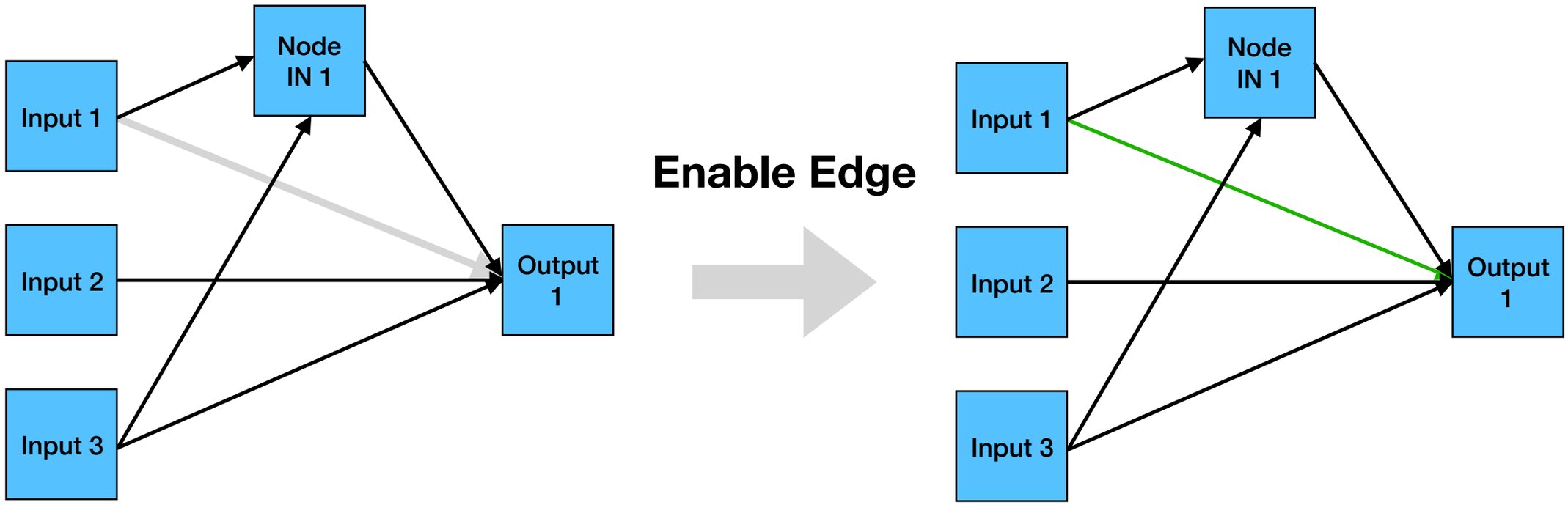
- EXALT/EXAMM always start with a minimal feed forward network (top left) with input nodes for each input parameter fully connected to output nodes for each output parameter (no hidden nodes).
- The edge between Input 1 and Output 1 is selected to be split. A new node with innovation number (IN) 1 is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

Edge Mutations: Add Edge



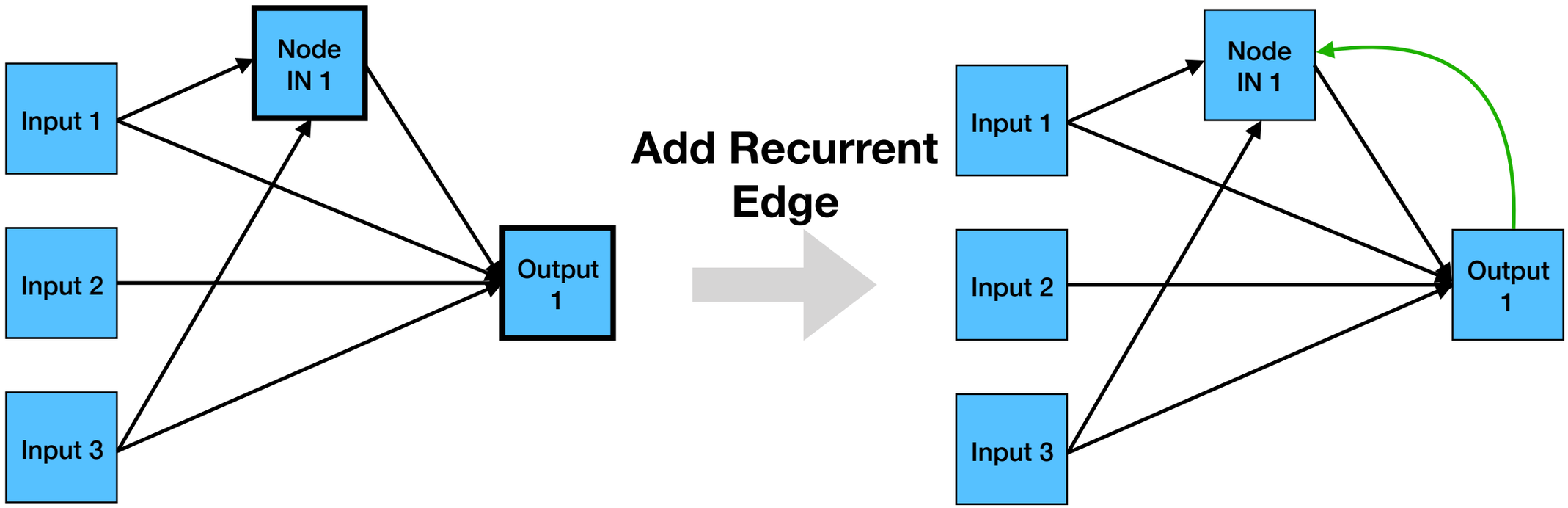
- Input 3 and Node IN 1 are selected to have an edge between them added.

Edge Mutations: Enable Edge



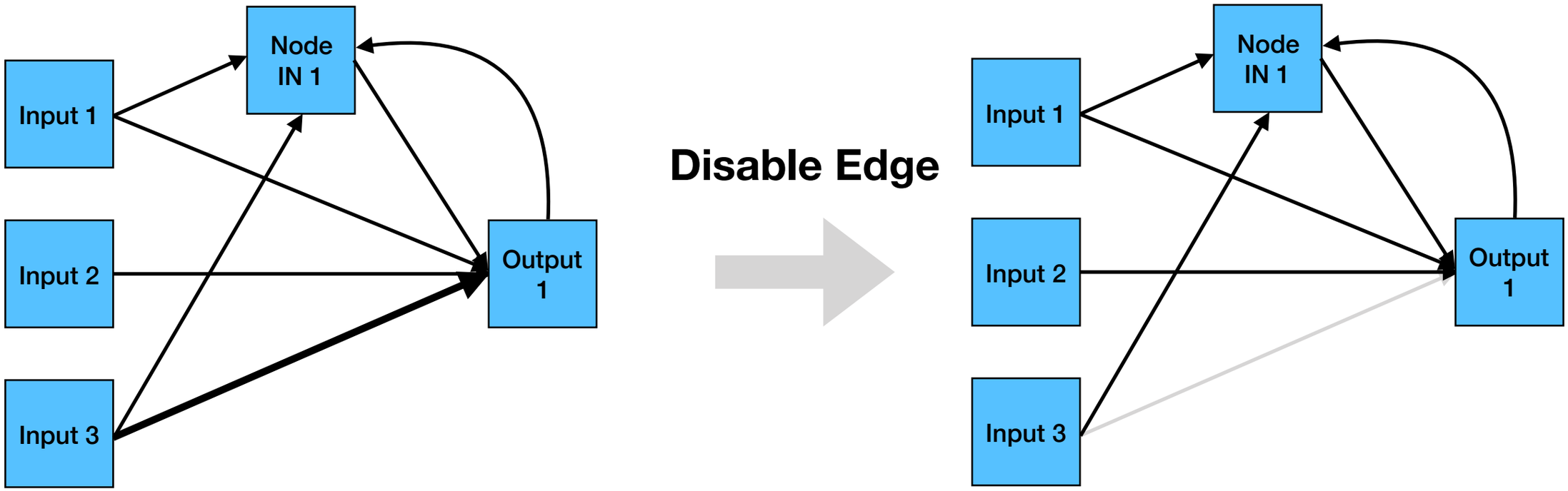
- The edge between Input 3 and Output 1 is enabled.

Edge Mutations: Add Recurrent Edge



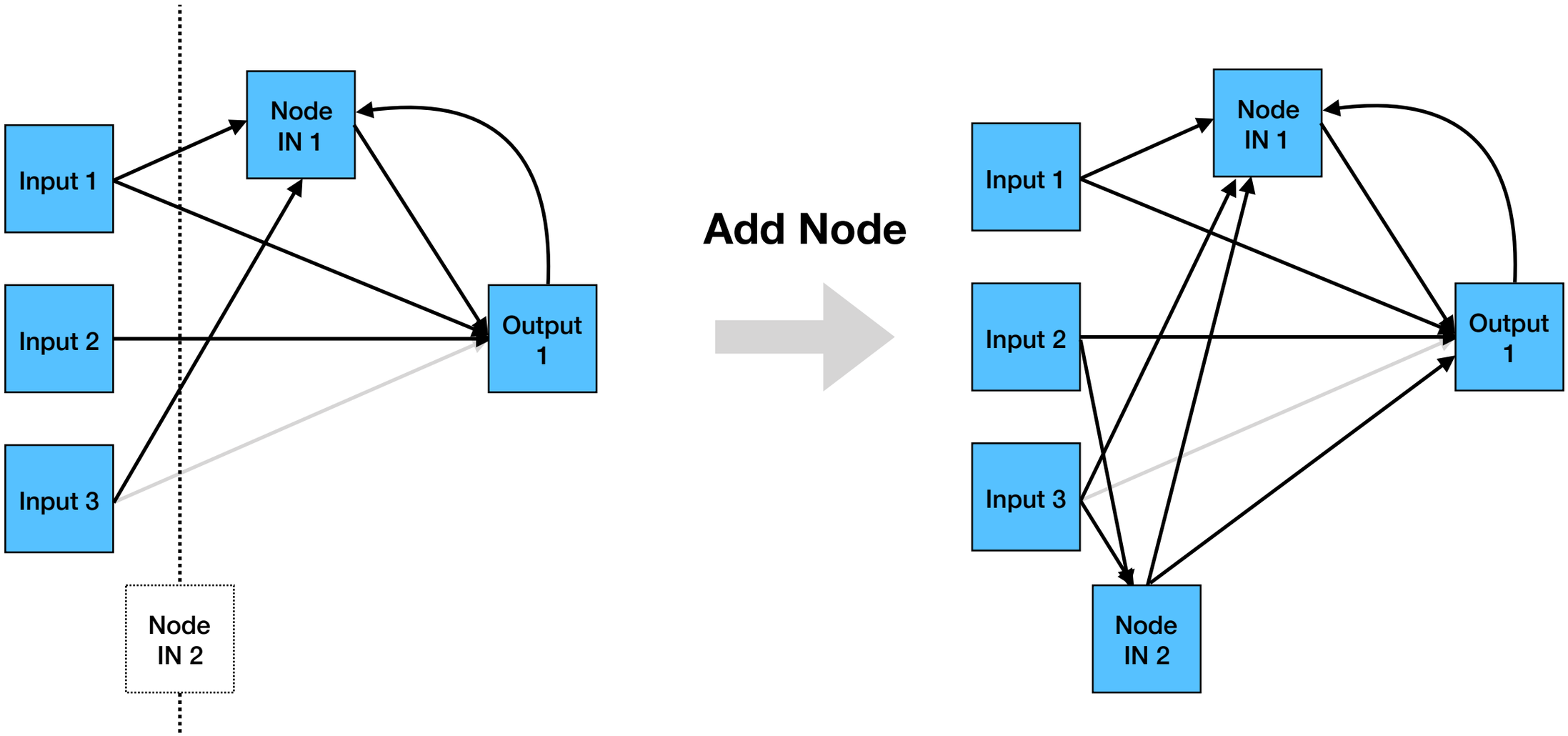
- A recurrent edge is added between Output 1 and Node IN 1.

Edge Mutations: Disable Edge



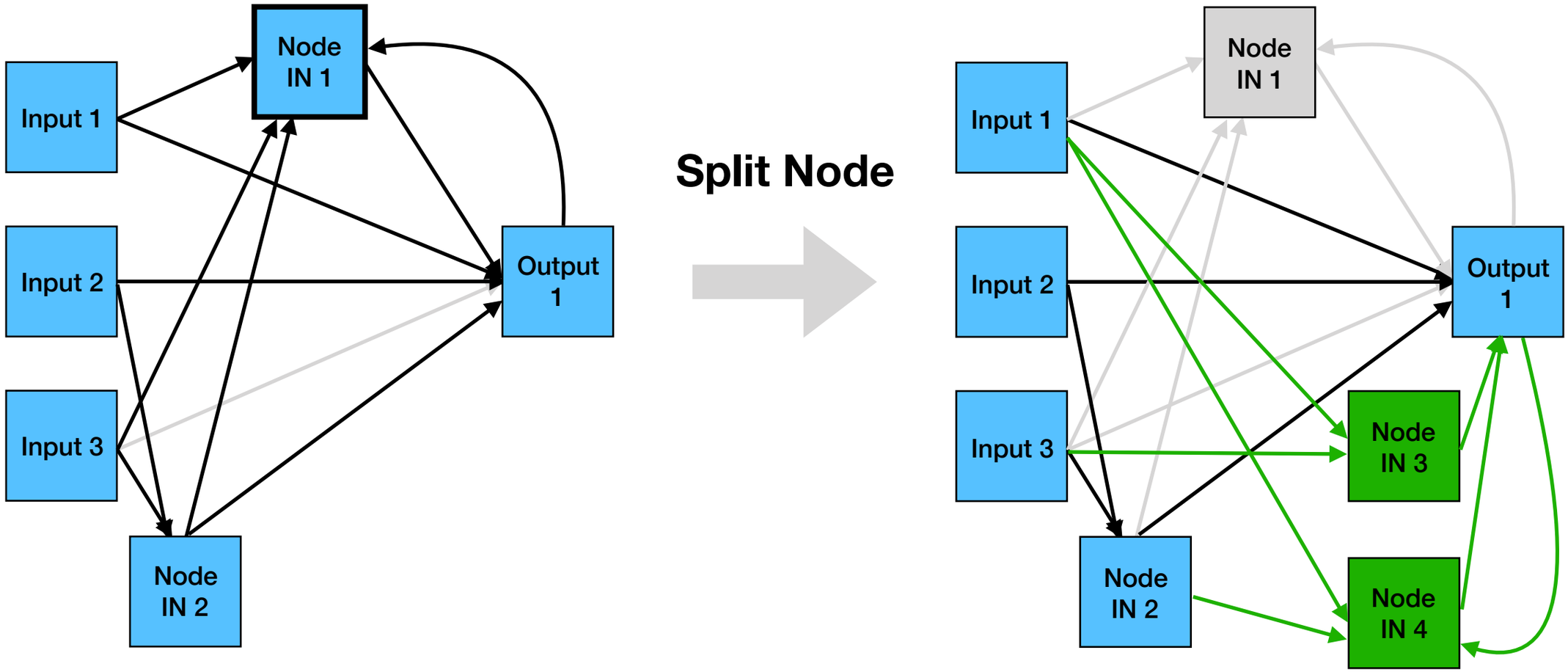
- The edge between Input 3 and Output 1 is disabled.

Node Mutations: Add Node



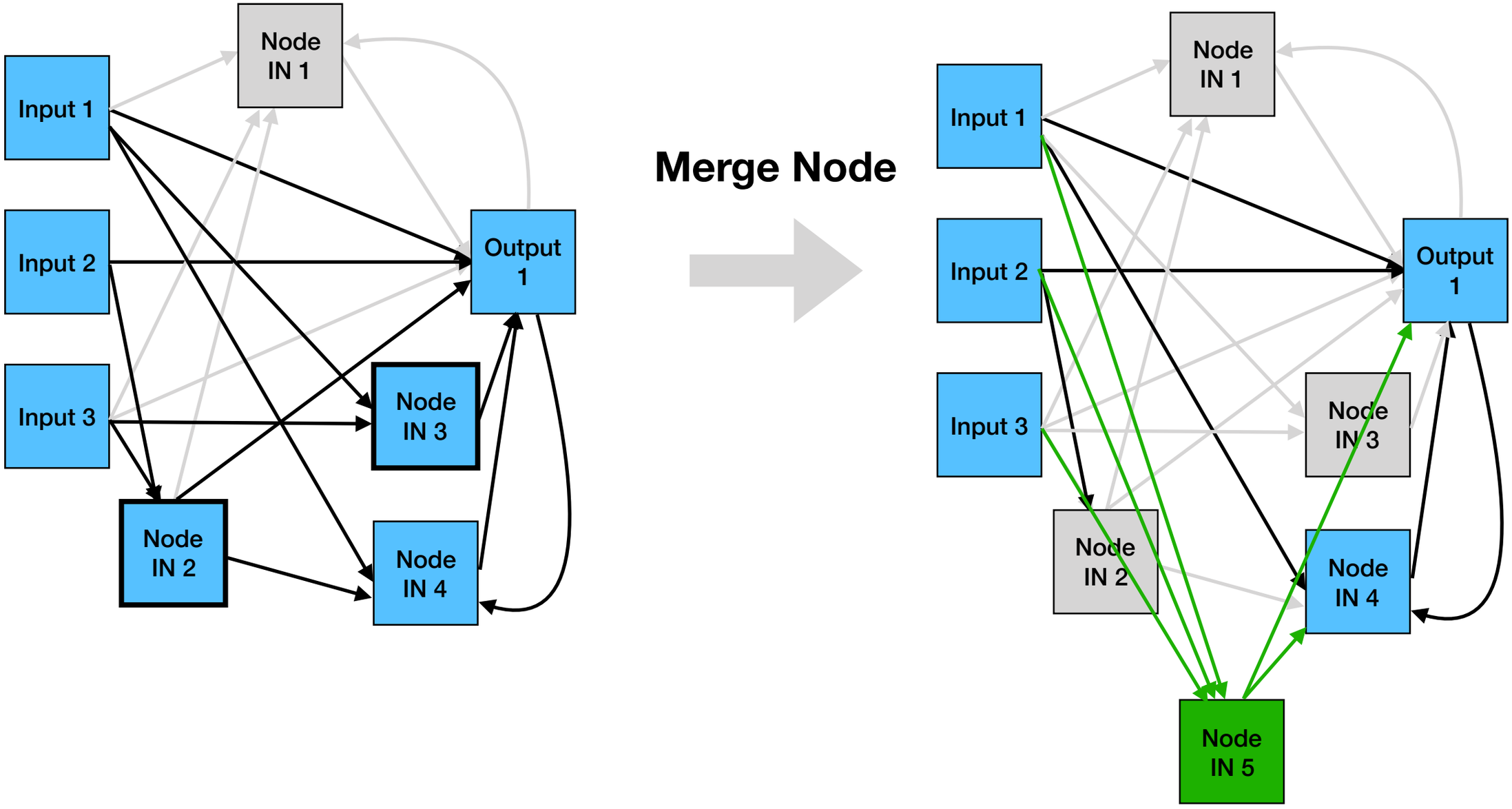
- A node with IN 2 is selected to be added at a depth between the inputs & Node IN 1. Edges are randomly added to Input 2 and 3, and Node IN 1 and Output 1.

Node Mutations: Split Node



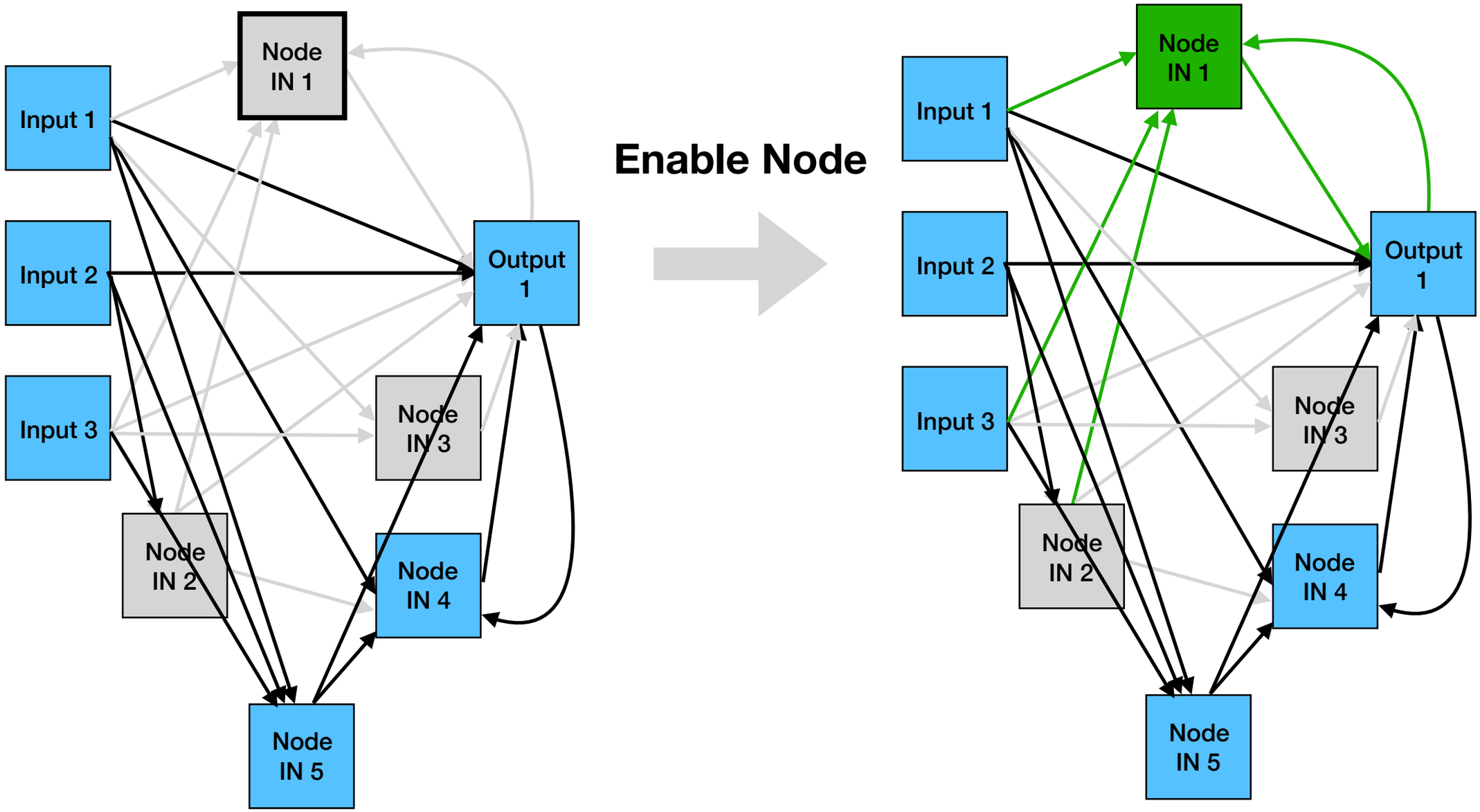
- Node IN 1 is selected to be split. It is disabled with its input/output edges. It is split into Nodes IN 3 and 4, which get half the inputs. Both have an output edge to Output 1 since there was only one output from Node IN 1.

Node Mutations: Merge Node



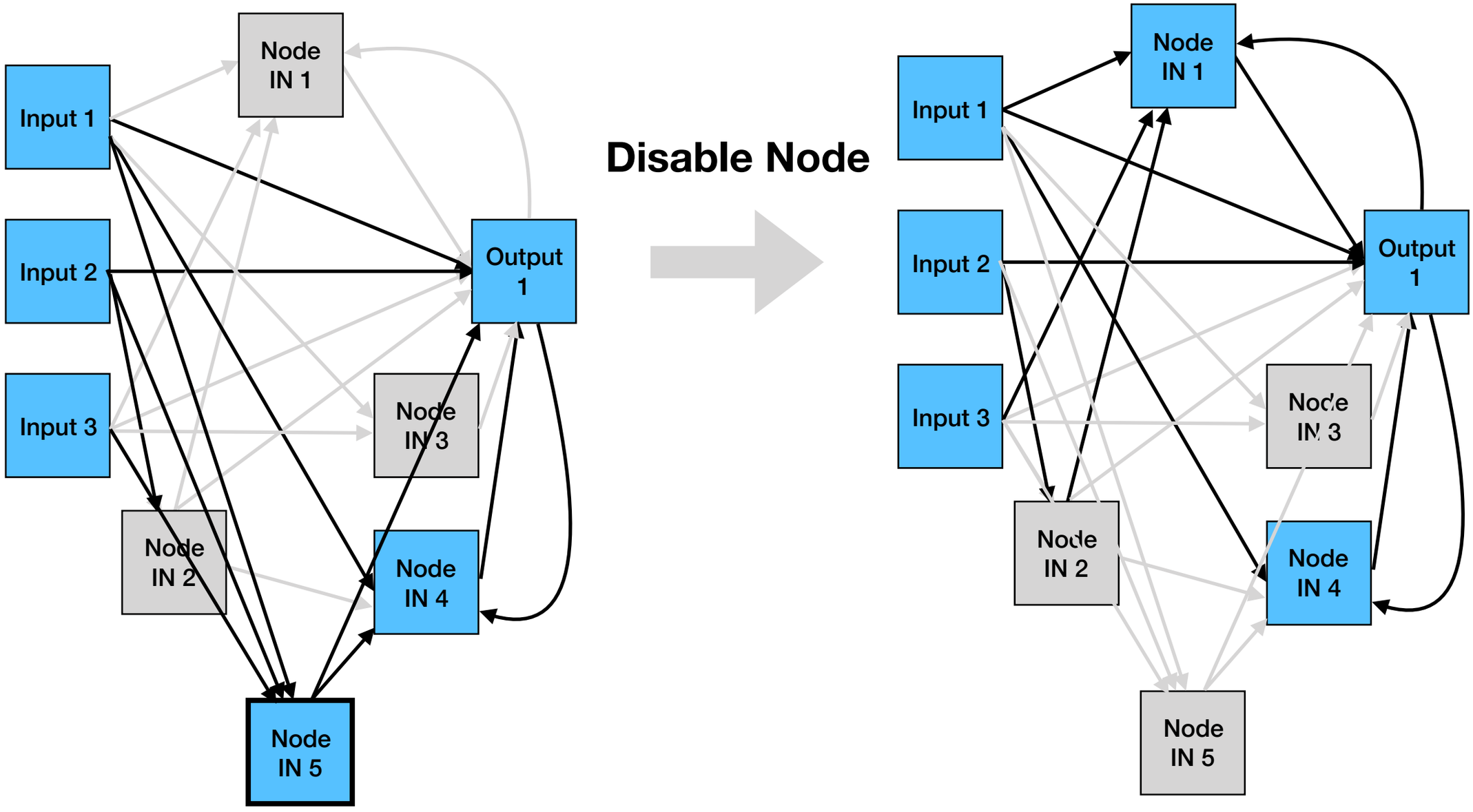
- Node IN 2 and 3 are selected for a merger (input/output edges are dis- abled). Node IN 5 is created with edges between all their inputs/outputs.

Node Mutations: Enable Node



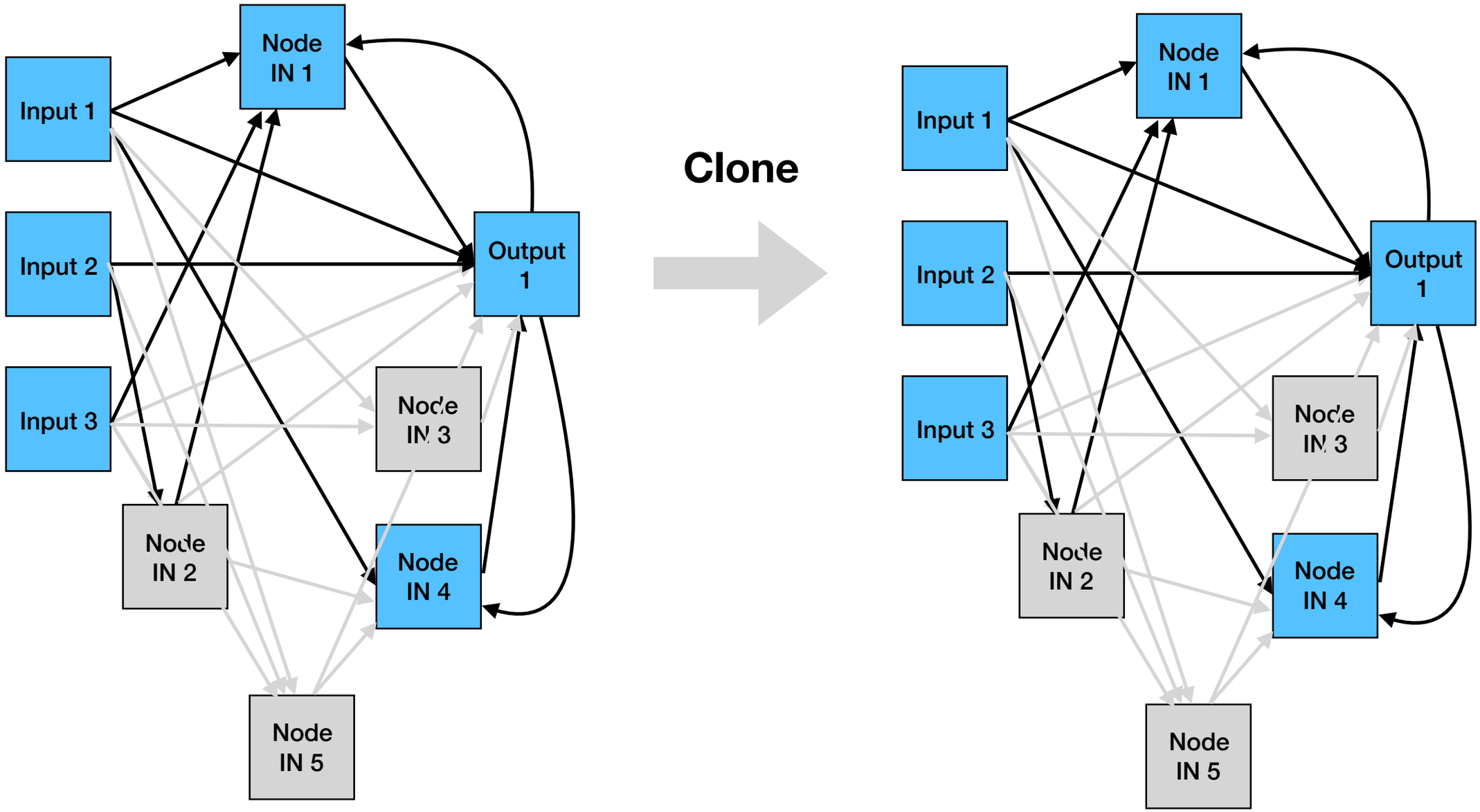
- Node IN 1 is selected to be enabled, along with all its input and output edges.

Node Mutations: Disable Node



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

Clone

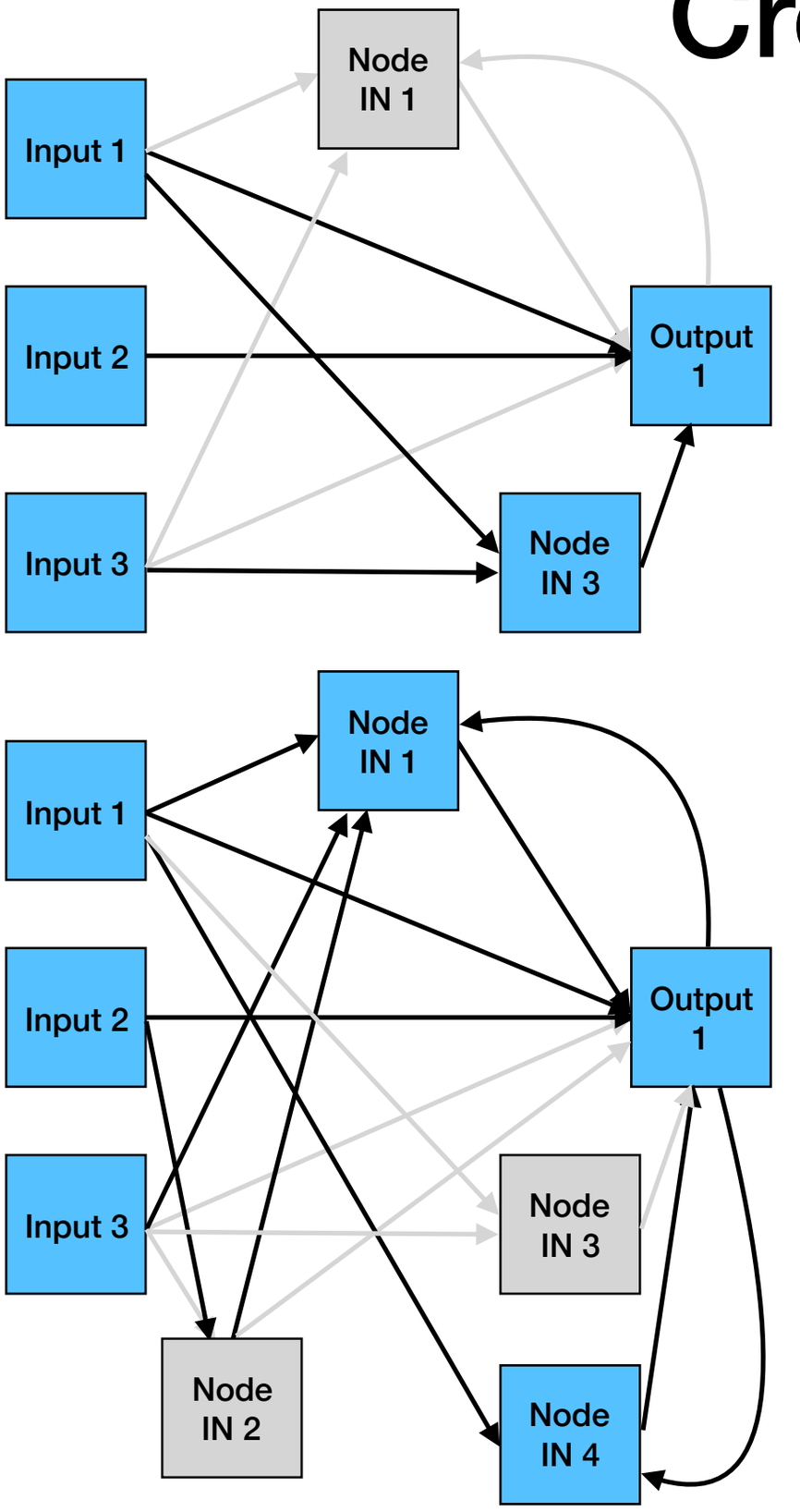


- Clone makes no modifications at all to the parent, allowing it to continue with the back propagation process.

Crossover

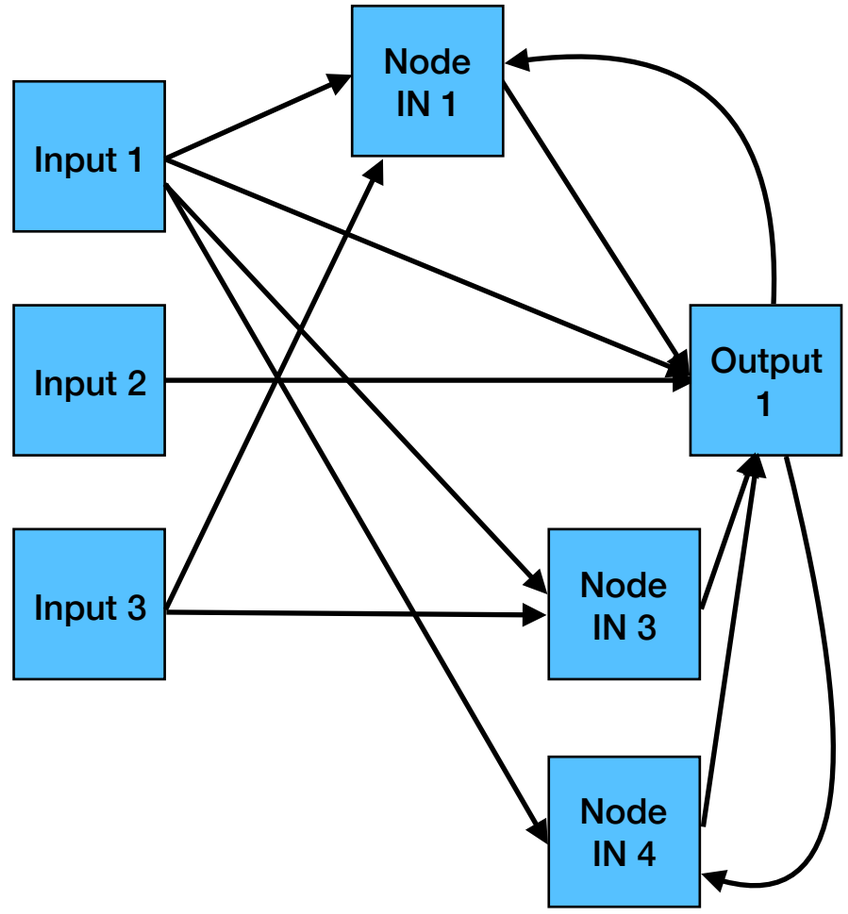
Crossover

Worse Parent



Better Parent

Crossover

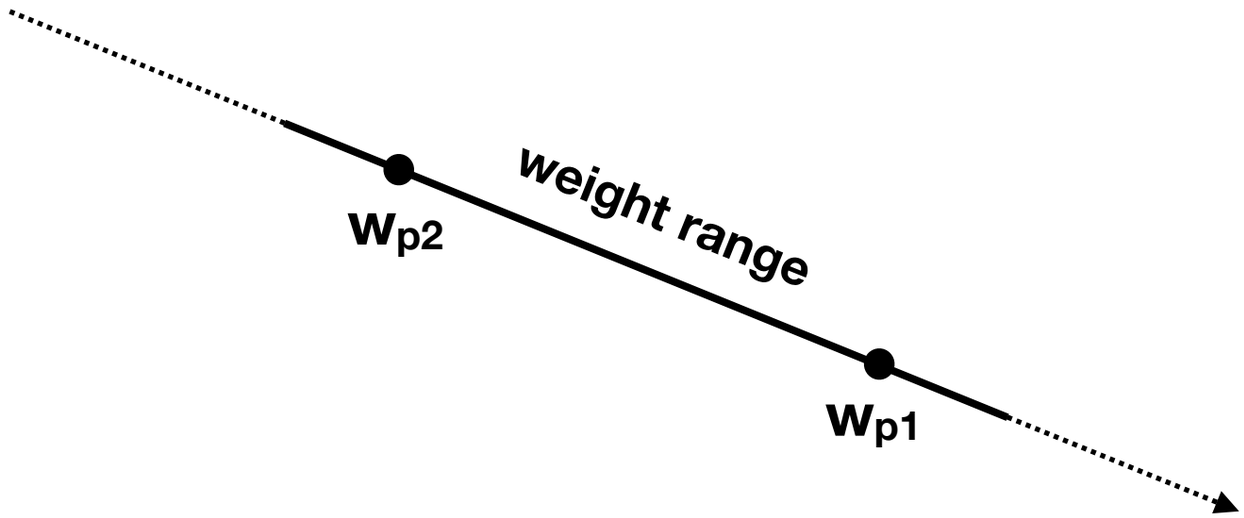


- Crossover creates a child RNN using all reachable nodes and edges from two parents. A node or edge is reachable if there is a path of enabled nodes and edges from an input node to it as well as a path of enabled nodes and edges from it to an output node, i.e., a node or edge is reachable if it actually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

Crossover: 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 parents' 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:

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

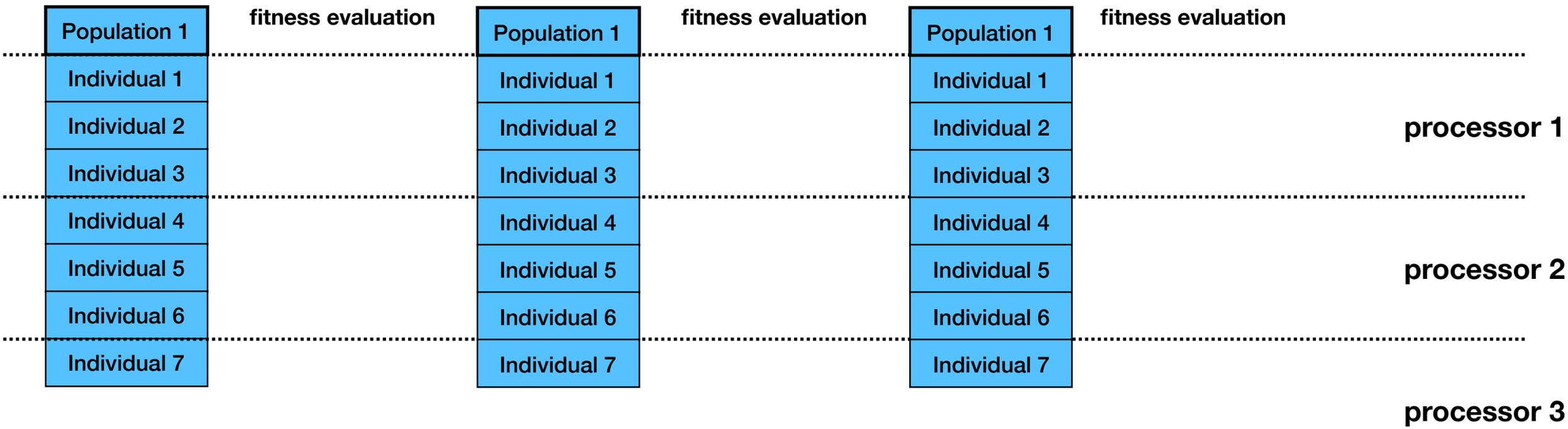
- Where r is a random number -0.5 <= r <= 1.5, where w_{p1} is the weight from the more fit parent, and w_{p2} is the weight from the less fit parent. We can change r's bounds to prefer weights near parent over the other.

Distributed Neuro-Evolution

RIT | Rochester Institute of Technology

CHAI AI Seminar
February 11, 2019

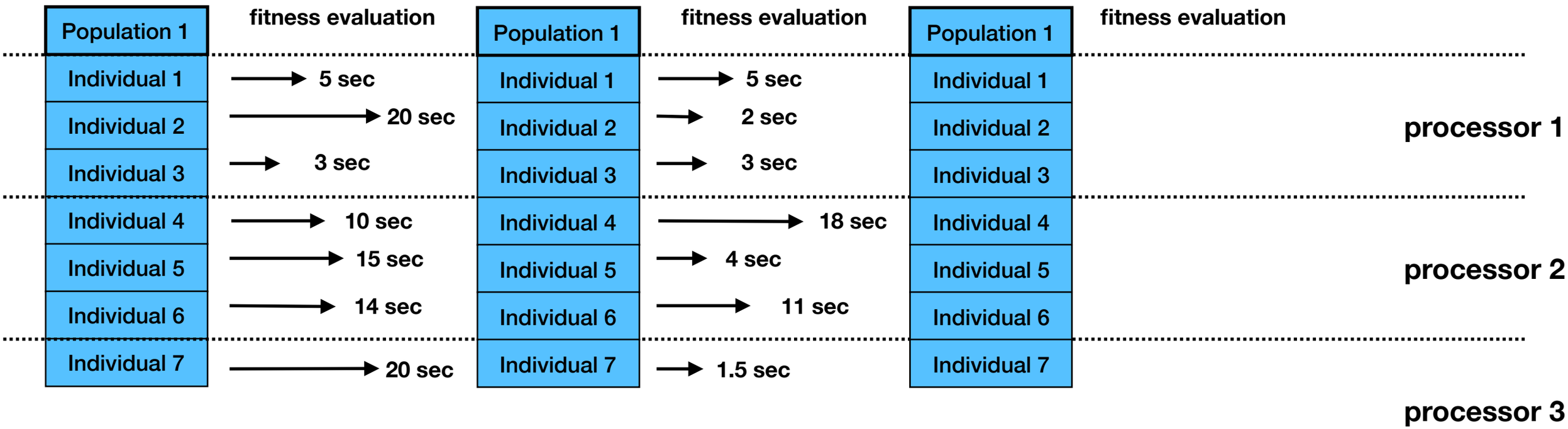
Synchronous/Parallel EAs



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

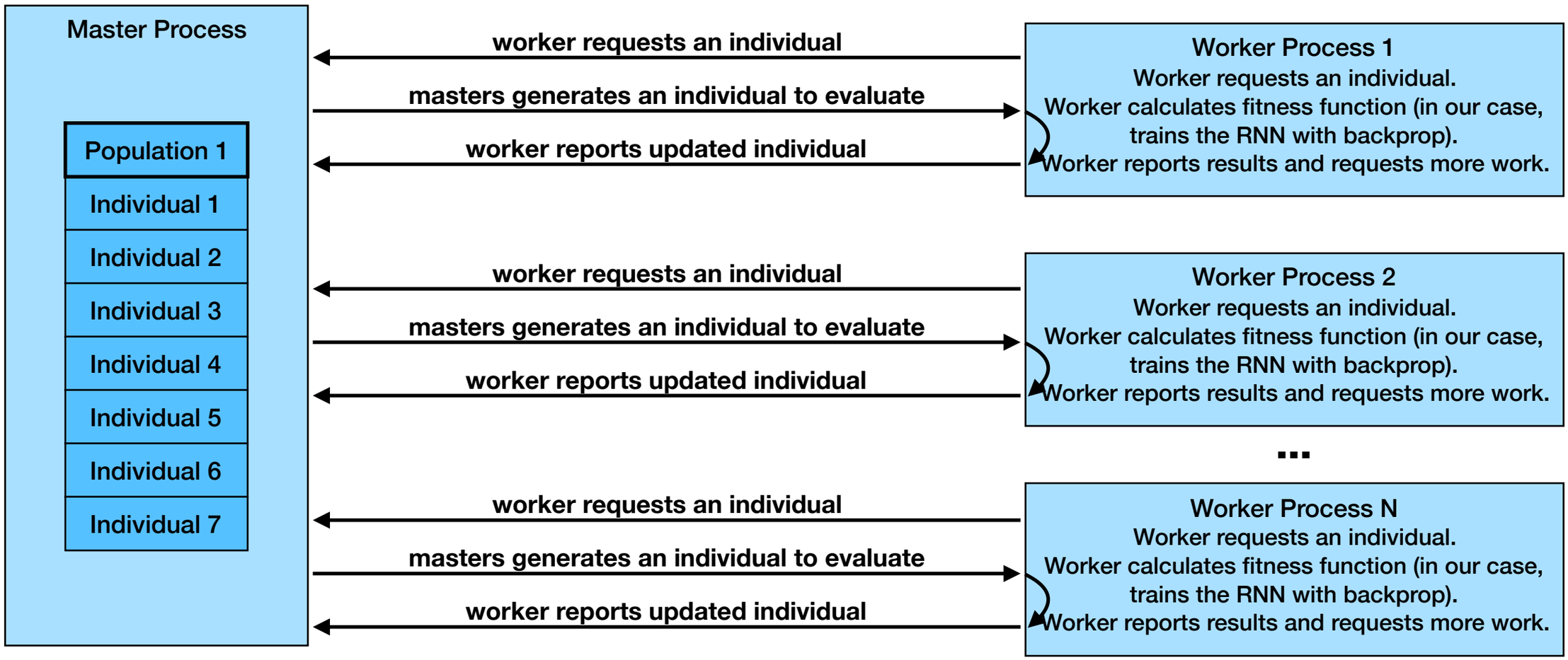
- This has problems in that if the population size is not evenly divisible by the number of processors available there is wasted computation. Also, the population size can't be *less* than the number of processors.

Synchronous/Parallel EAs



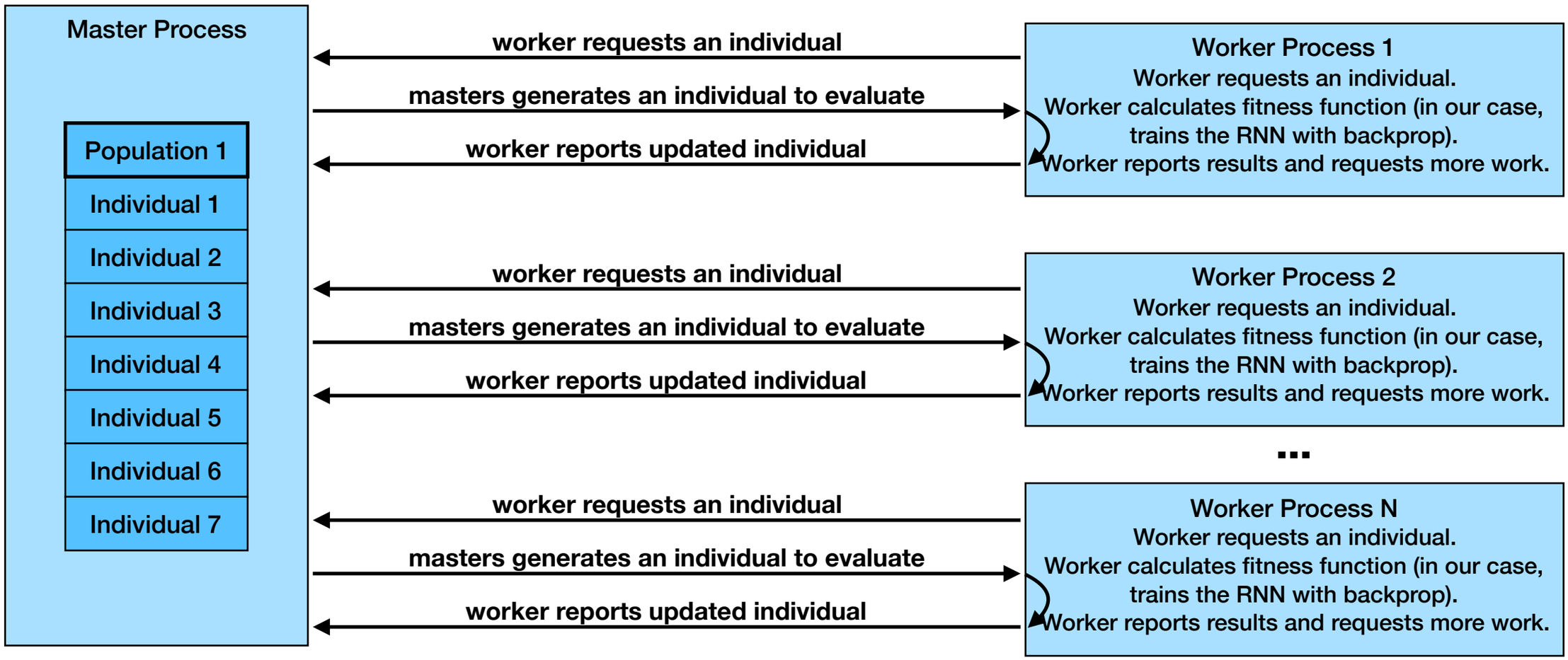
- Things are even more challenging if the fitness evaluation times of the individuals are different or even worse nondeterministic. Lots of waiting and unused cycles.

Asynchronous EAs



- The Master process keeps a "steady state" populuation.
- 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.

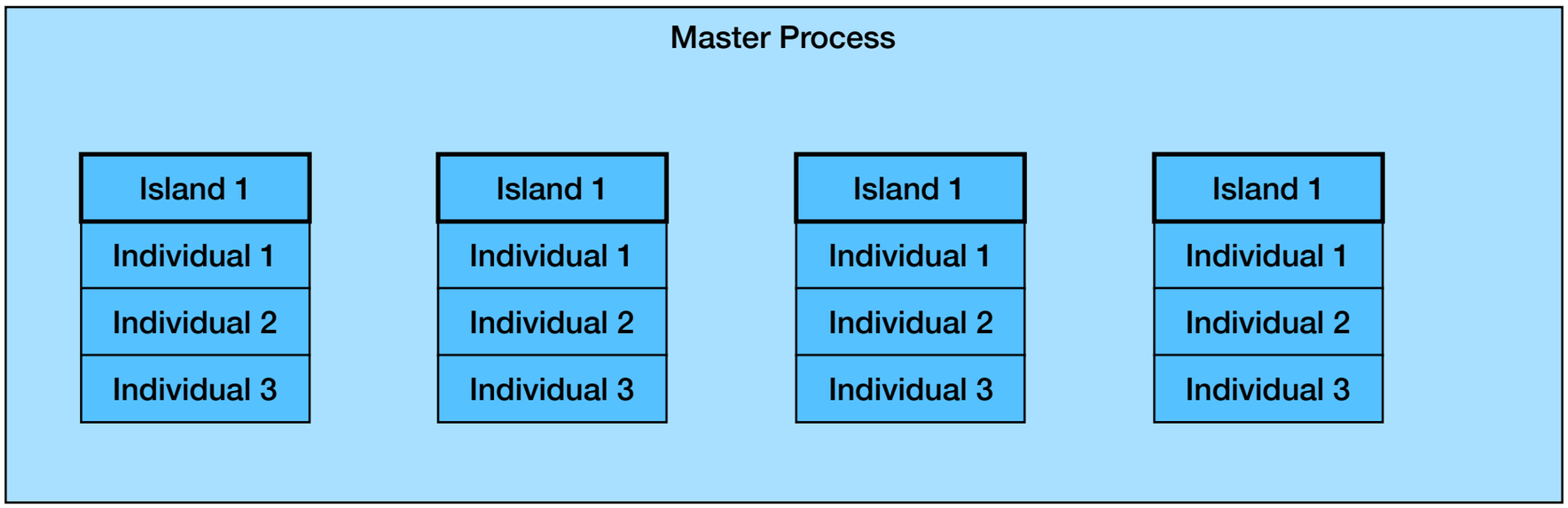
Asynchronous EAs



- 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

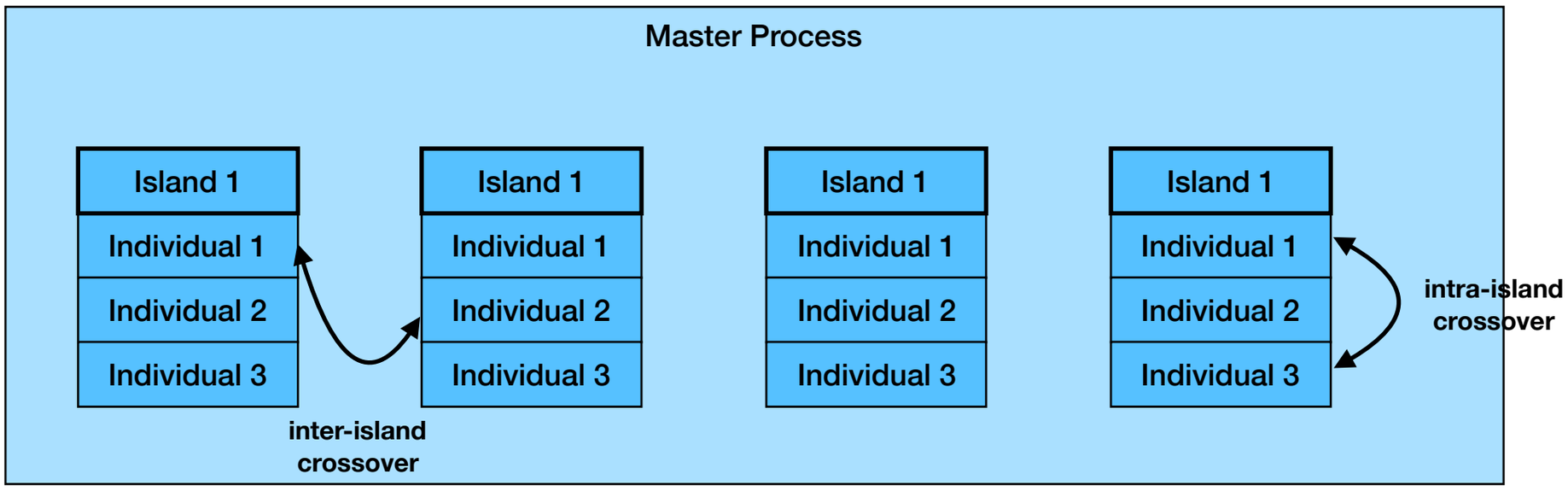


- EXAMM uses islands, which have been shown to potentially provide superlinear speedup to some EAs [2].

- 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).

Simple EXAMM/Asynchronous EA Pseudo-code

Worker Processes:

```
Message msg;
while ((msg = get_msg_from_master()) != TERMINATE_MSG) {
   RNN rnn = msg.get_rnn();
   rnn.backpropagate();
   master.send(rnn);
}
```

Master Process:

```
EXAMM examm = new EXAMM(...)
while (not examm.done) {
    Message msg = get_msg_any_worker()
    switch (msg.type) {
       if (msg.type == RESULT_MSG) {
          examm.insert(msg.get_rnn())
       } else if (msg.type == request) {
          if (done) msg.src.send(TERMINATE_MSG)
          else msg.src.send(examm.create_rnn())
       }
   }
}
```

- Workers repeatedly request RNNs, train them and send the resulting RNNs to the master.
- The master repeatedly receives messages from the workers, inserting RNNs to the population/ island(s) as they arrive and generating new RNNs on work requests.

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.
- EXALT/EXAMM runs utilized different compute notes as determined by RC's SLURM scheduler.

EXALT Experimental Setup

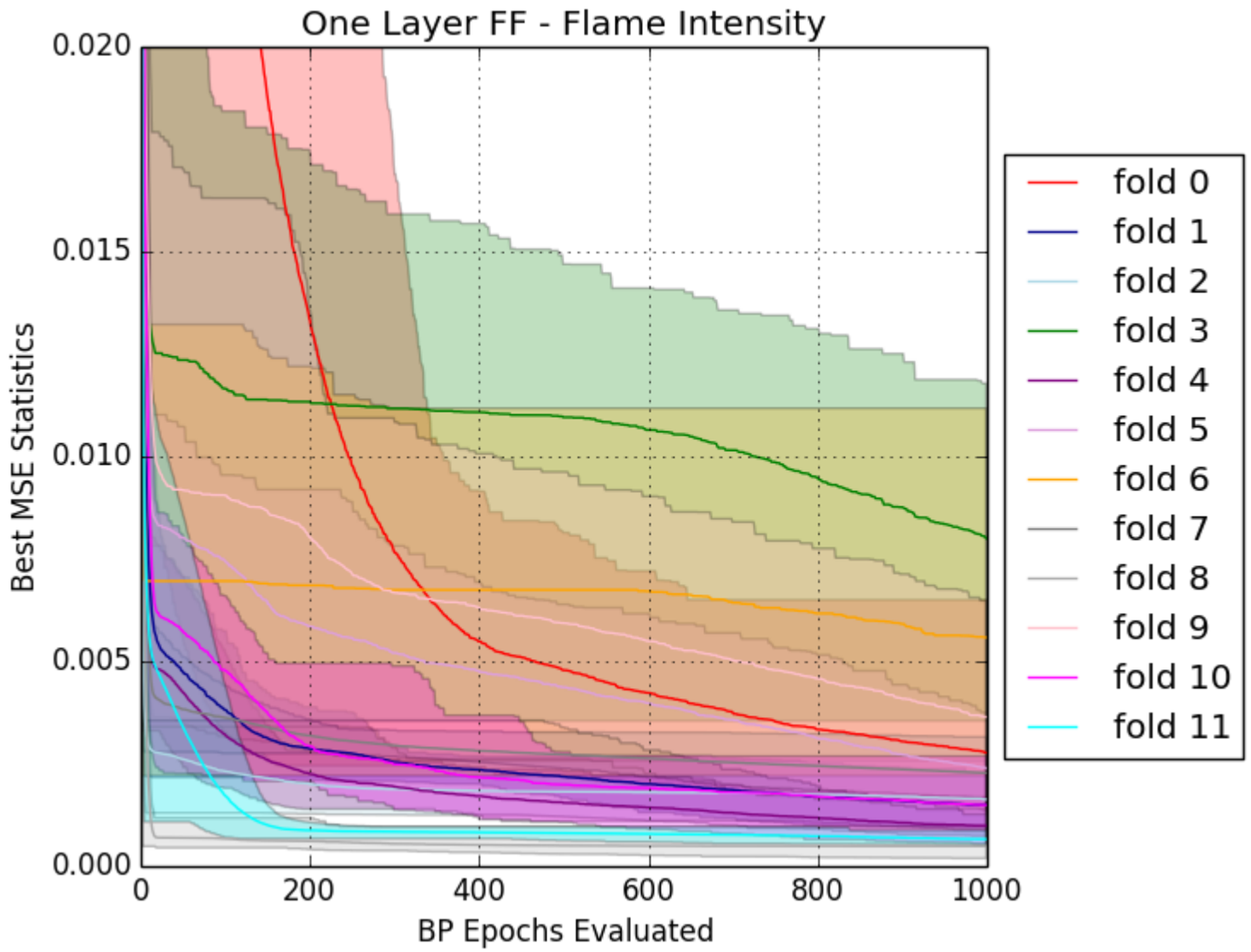
- EXALT compared to traditional RNNs (1-layer FF, 2-layer FF, 1-layer LSTM, 2-layer lSTM, Jordan, Elman) to predict **Flame Intensity**

- K-fold cross validation (1 file per fold), 10 repeats per fold - 720 runs for each of the fixed RNN types.

- K-fold cross validation (1 file per fold), 10 repeats per fold - 120 runs for EXALT.

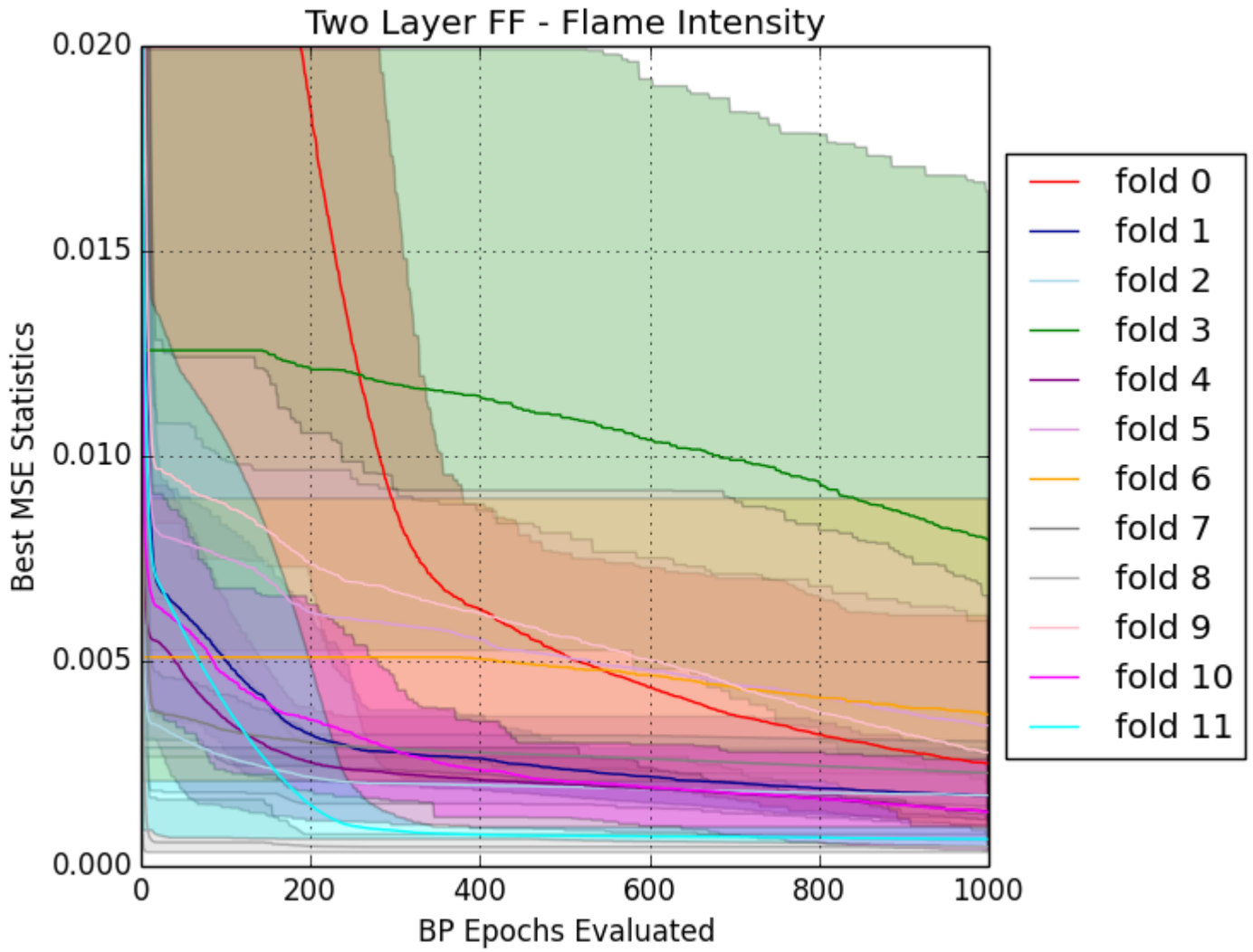
- Fixed RNNs trained for 1000 epochs.
- EXALT trained 2000 RNNs for 10 epochs each, distributed across 20 processes -- compute was expected to be somewhat comparable.

1 Layer Feed Forward



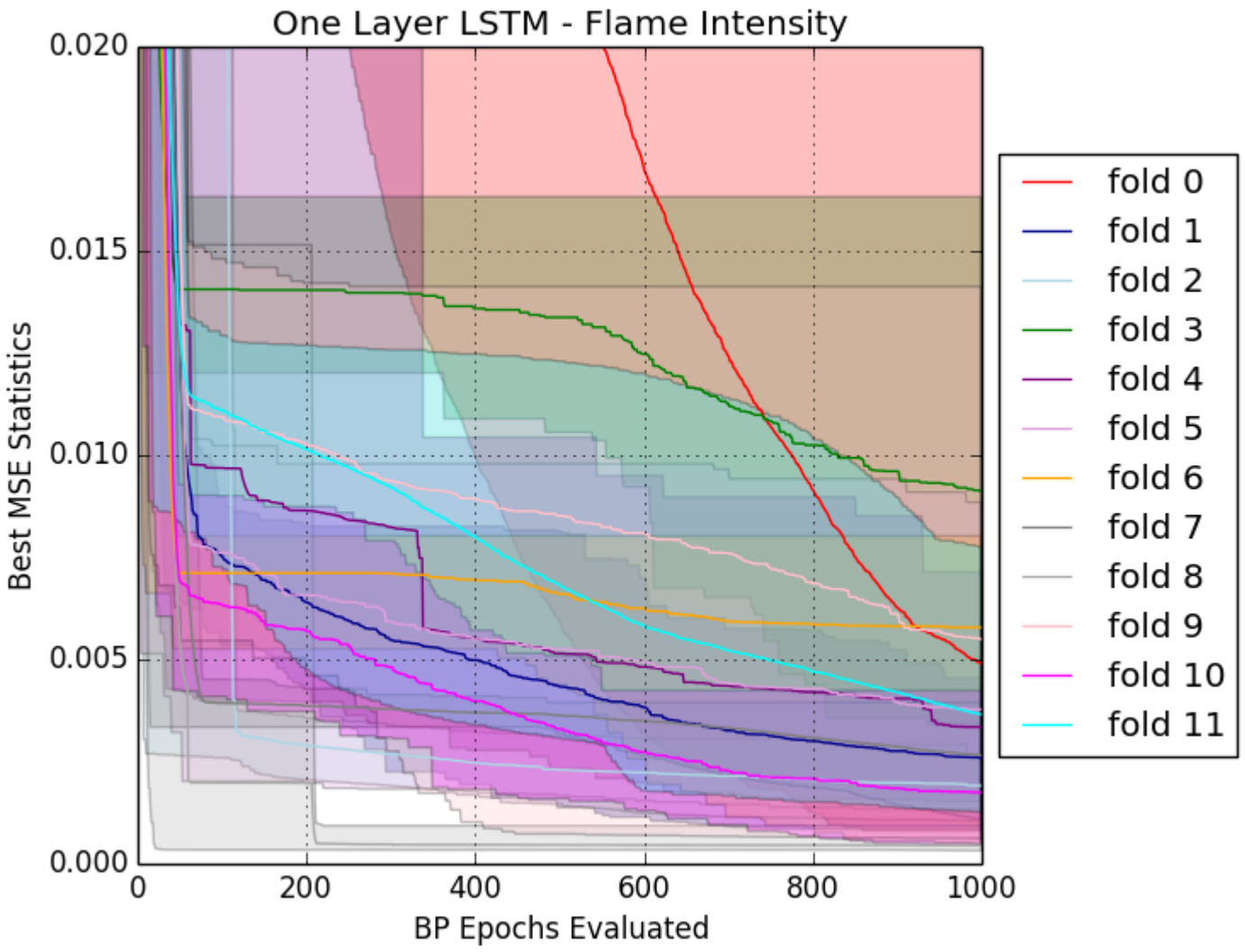
- Min/avg/max mean squared error while training for each fold.

2 Layer Feed Forward



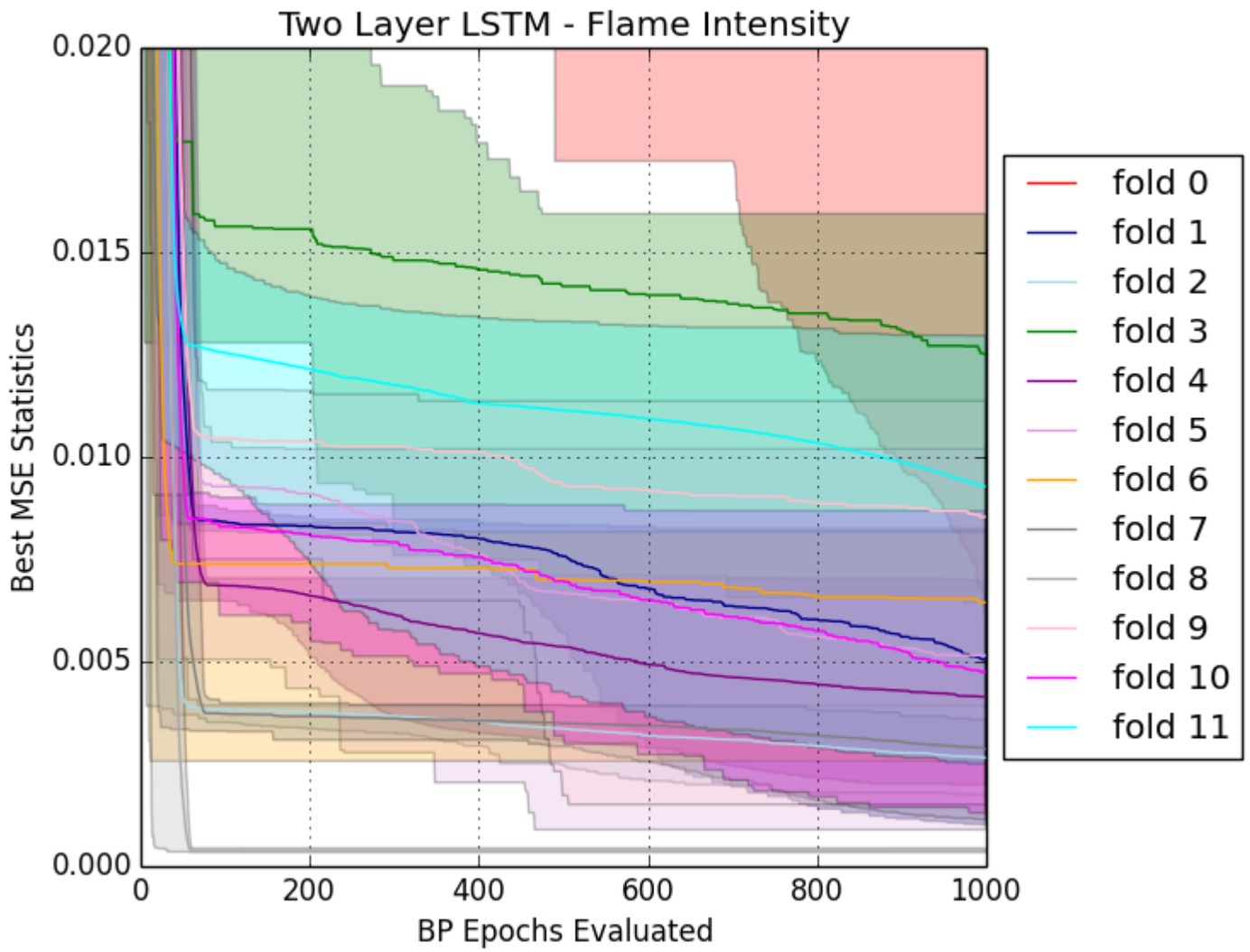
- Min/avg/max mean squared error while training for each fold.

1 Layer LSTM



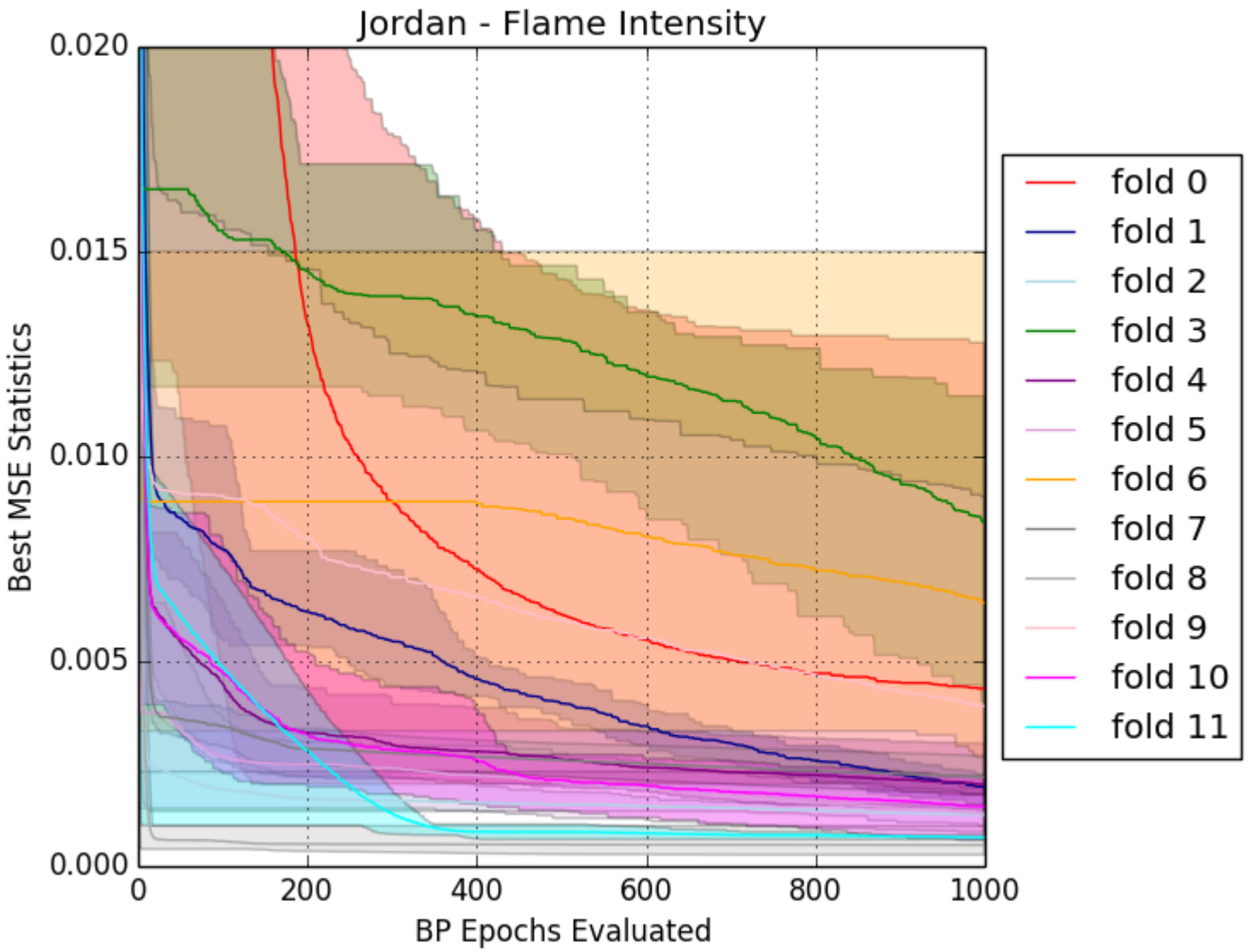
- Min/avg/max mean squared error while training for each fold.

2 Layer LSTM



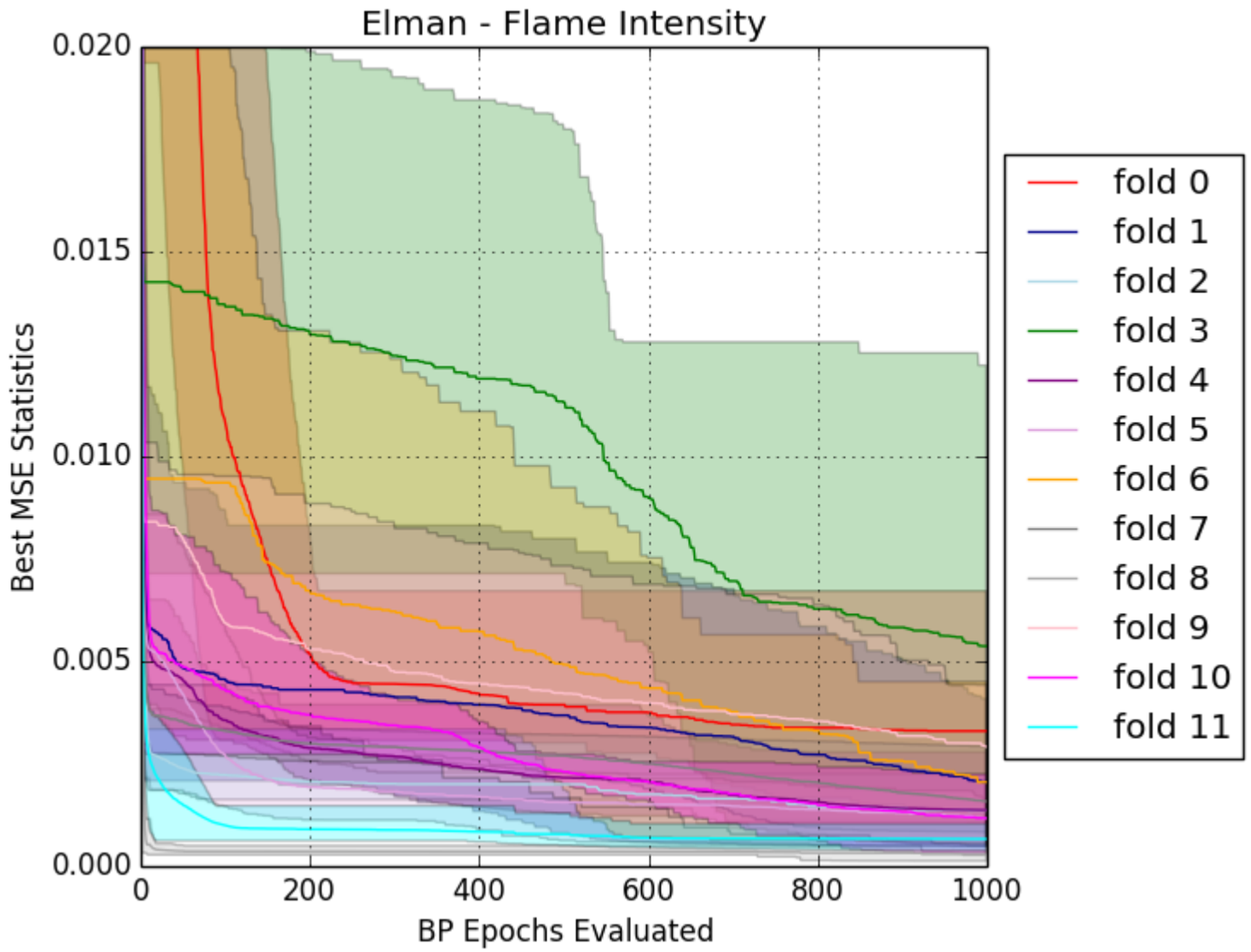
- Min/avg/max mean squared error while training for each fold.

Jordan



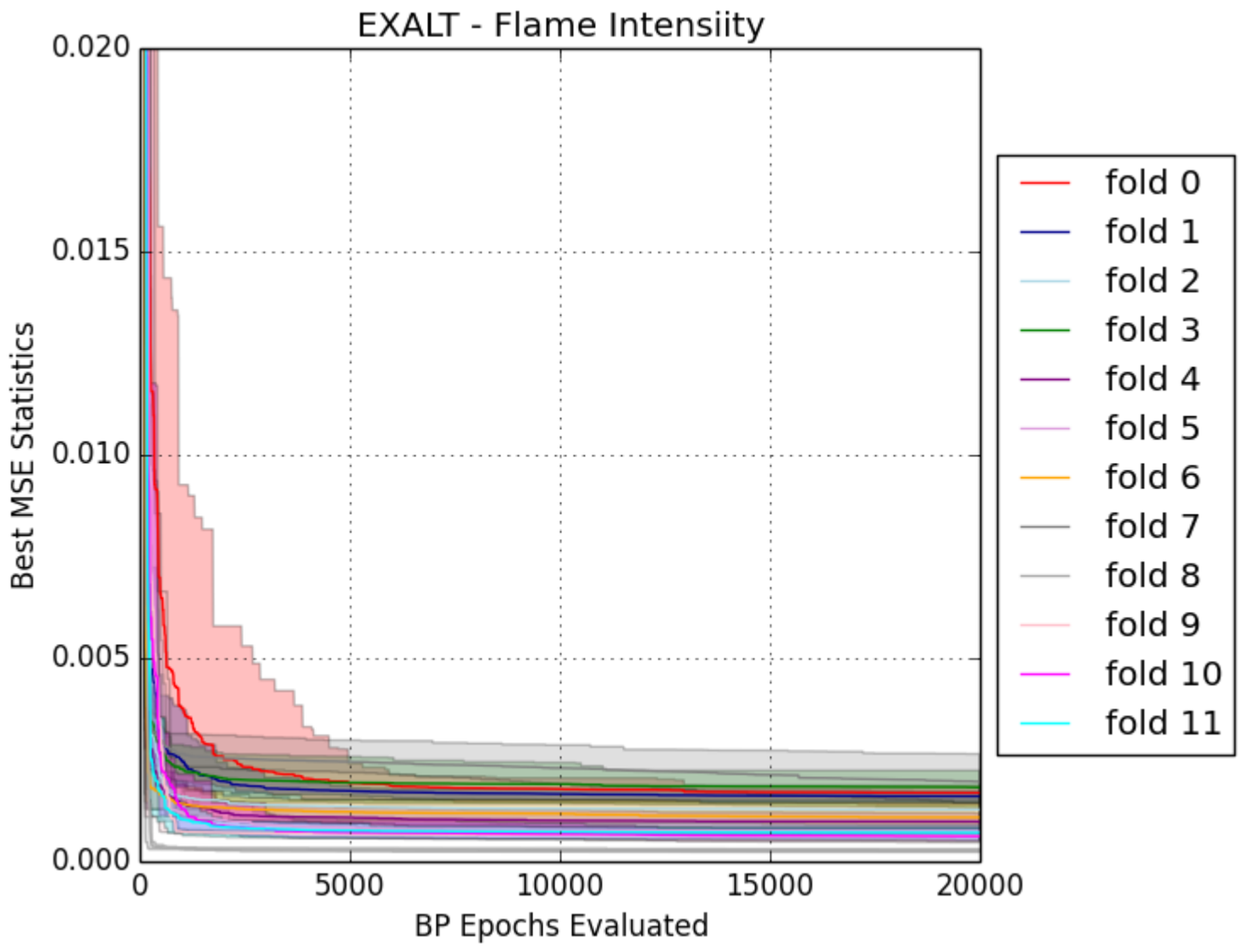
- Min/avg/max mean squared error while training for each fold.

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.**

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 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 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 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 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).

Alex Ororbia*, AbdElRahman ElSaid and Travis Desell*. **Investigating Recurrent Neural Network Memory Structures using Neuro-Evolution.** *arXiv Neural and Evolutionary Computing (cs.NE):* https://arxiv.org/abs/1902.02390

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 -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

- **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 peformed worse in the average and worst cases.

- Much larger search space (6 possible node types).

- **Open Question:** Can we further improve results by dynamically adapting the rates at which memory cells are generated?

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 could 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
- Testing multiple activation functions (tanh mostly used in this work)
- Hyperparameter optimization for RNN training
- Layer-level mutations to speed evolution
- Self-tuning EXAMM

Discussion/Questions?

https://github.com/travisdesell/exact