Evolving Recurrent Neural Networks for Time Series Data Prediction of Coal Plant Parameters

Evostar 2019:

The 22nd International Conference on the Applications of Evolutionary Computation

Travis Desell (tjdvse@rit.edu)
Associate Professor
Graduate Program Director: Data Science
Department of Software Engineering

Other Authors:
AbdElRahman ElSaid (PhD GRA)
Steven Benson, Shuchita Patwardhan, David Stadem (Microbeam Technologies, Inc.)

R·I·T
ROCHESTER INSTITUTE OF TECHNOLOGY

Overview

- What is Neuro-Evolution?
- Background:
  - Recurrent Neural Networks for Time Series Prediction
- EXALT:
  - NEAT Innovations
  - Edge and Node Mutations
  - Crossover
  - Distributed Neuro-Evolution
- Results
  - Performance vs. Traditional
  - EXALT Results
- Future Work
- Questions

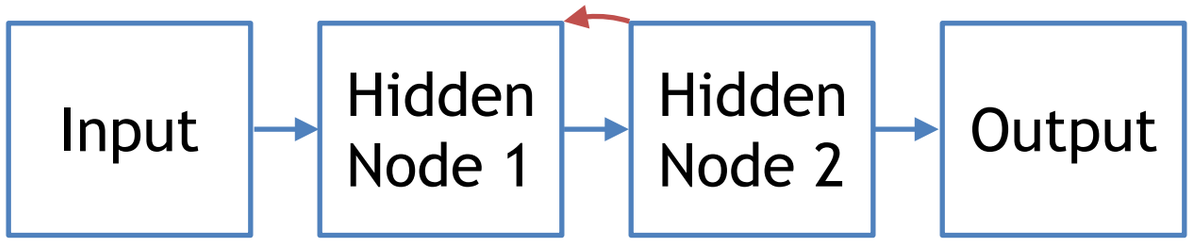
Motivation

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)

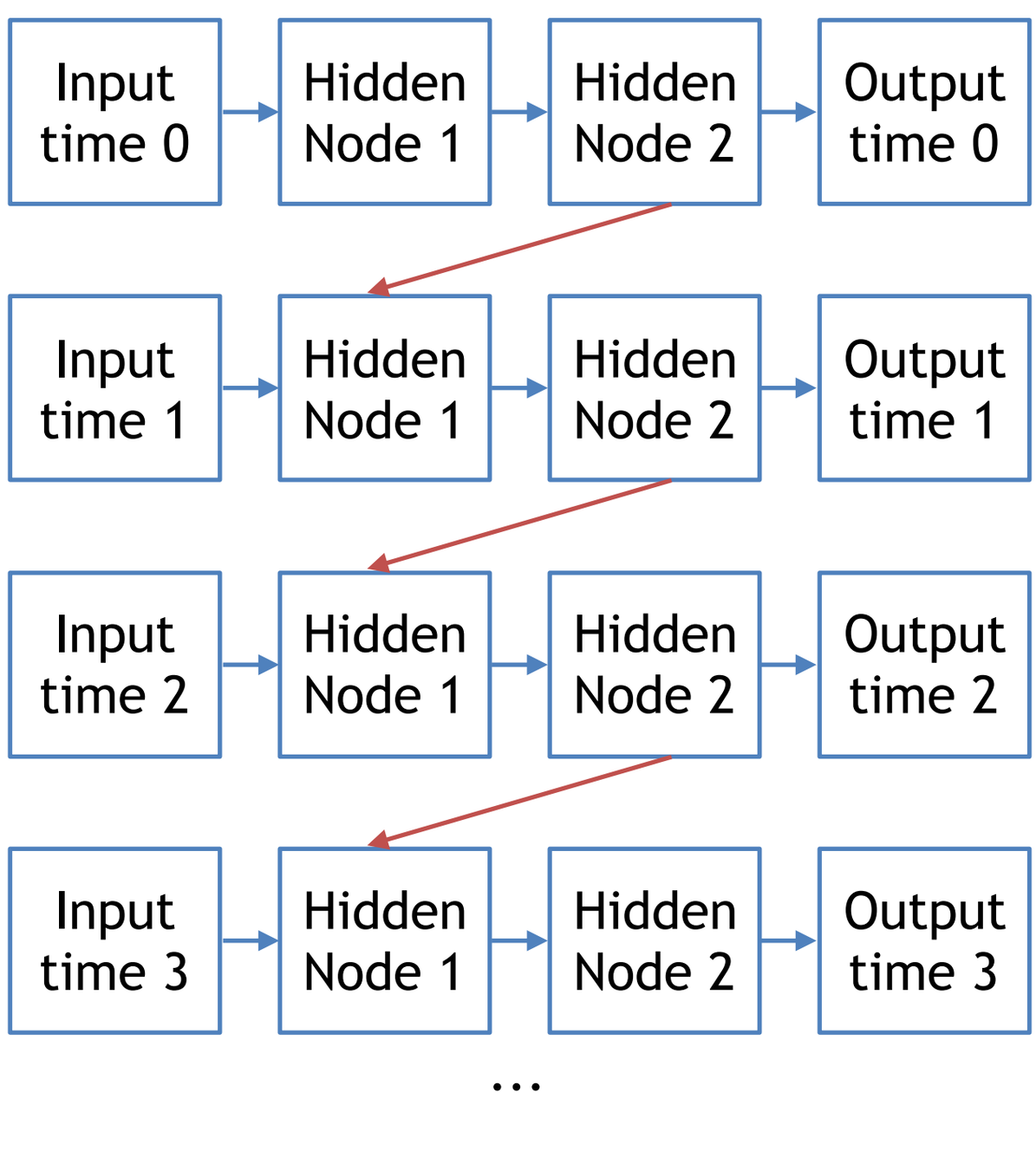
Background

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.*

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

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

EXALT

RIT | Rochester Institute of Technology

EvoStar: EvoApplications
April 26, 2019

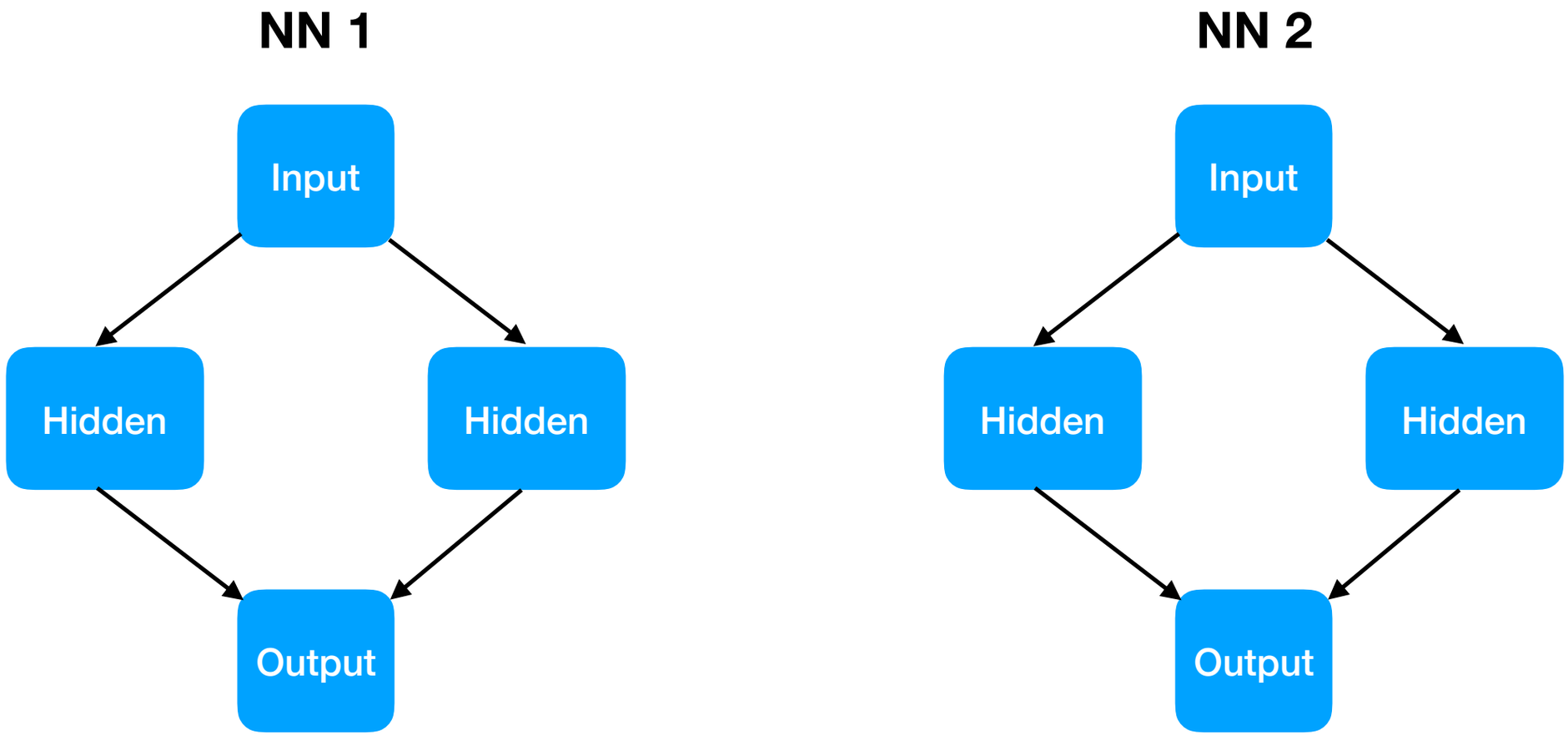
EXALT

- 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.
  - No speciation.
  - Generated RNNs are trained via backpropagation.
  - Weights pre-initialized from parents (Lamarckian evolution).

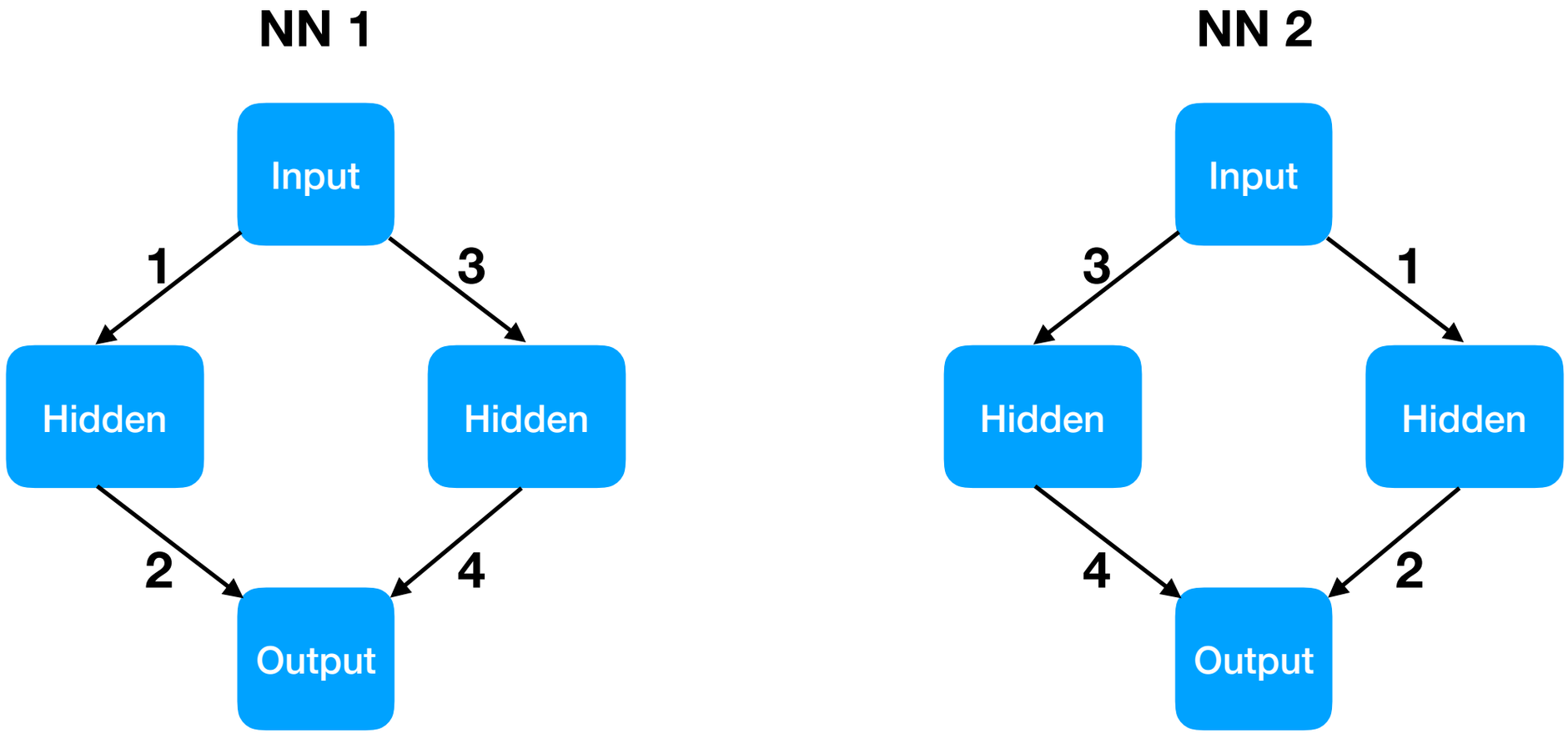
[1] Kenneth Stanley and Risto Miikkulainen. **Evolving neural networks through augmenting topologies.** *Evolutionary computation: 10, 2*. (2002), 99–127.

NEAT Innovation Numbers



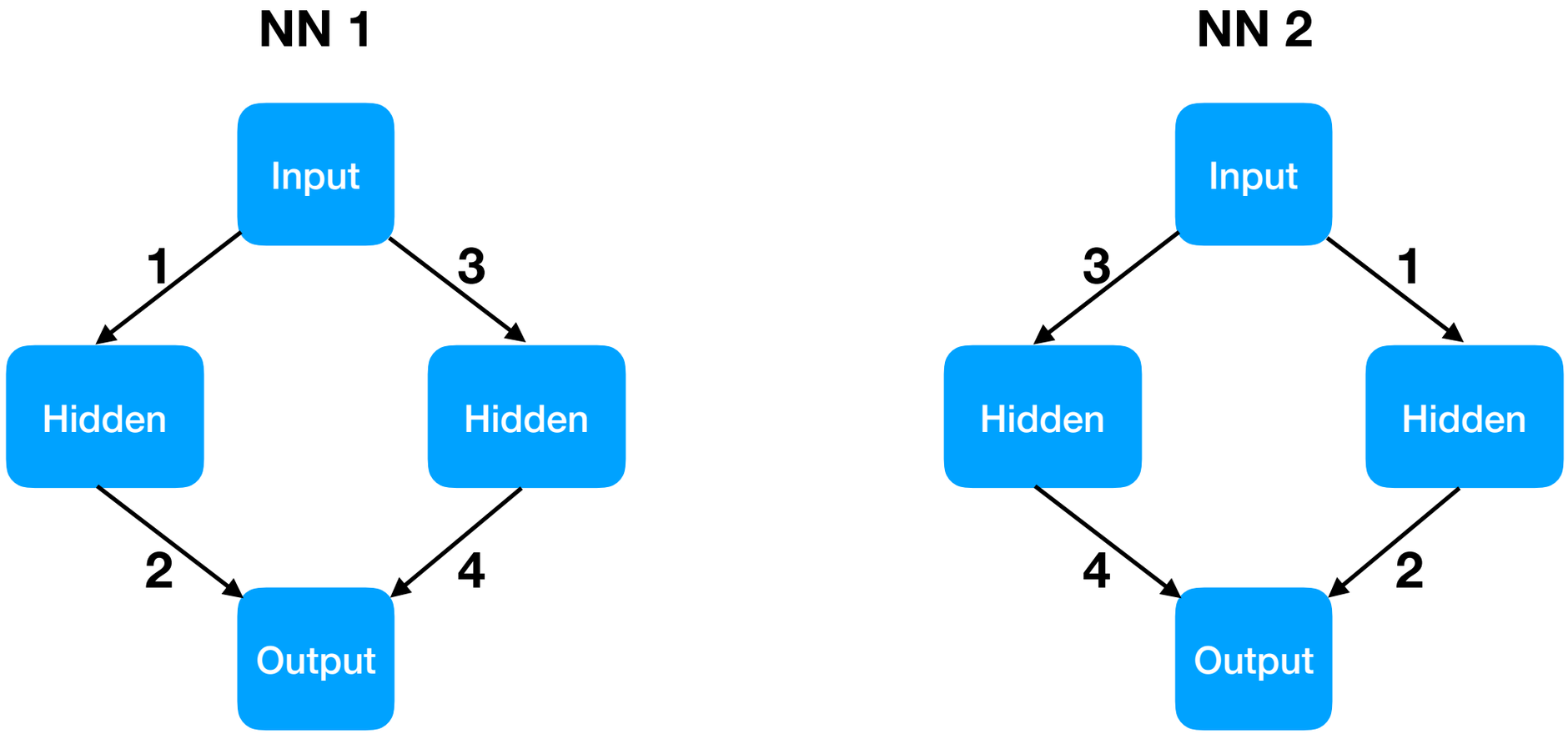
- In neuro-evolution, we need to perform crossover/ recombination between progressively grown neural networks.
- How do we know which edges are the "same" in the above neural networks?

NEAT Innovation Numbers



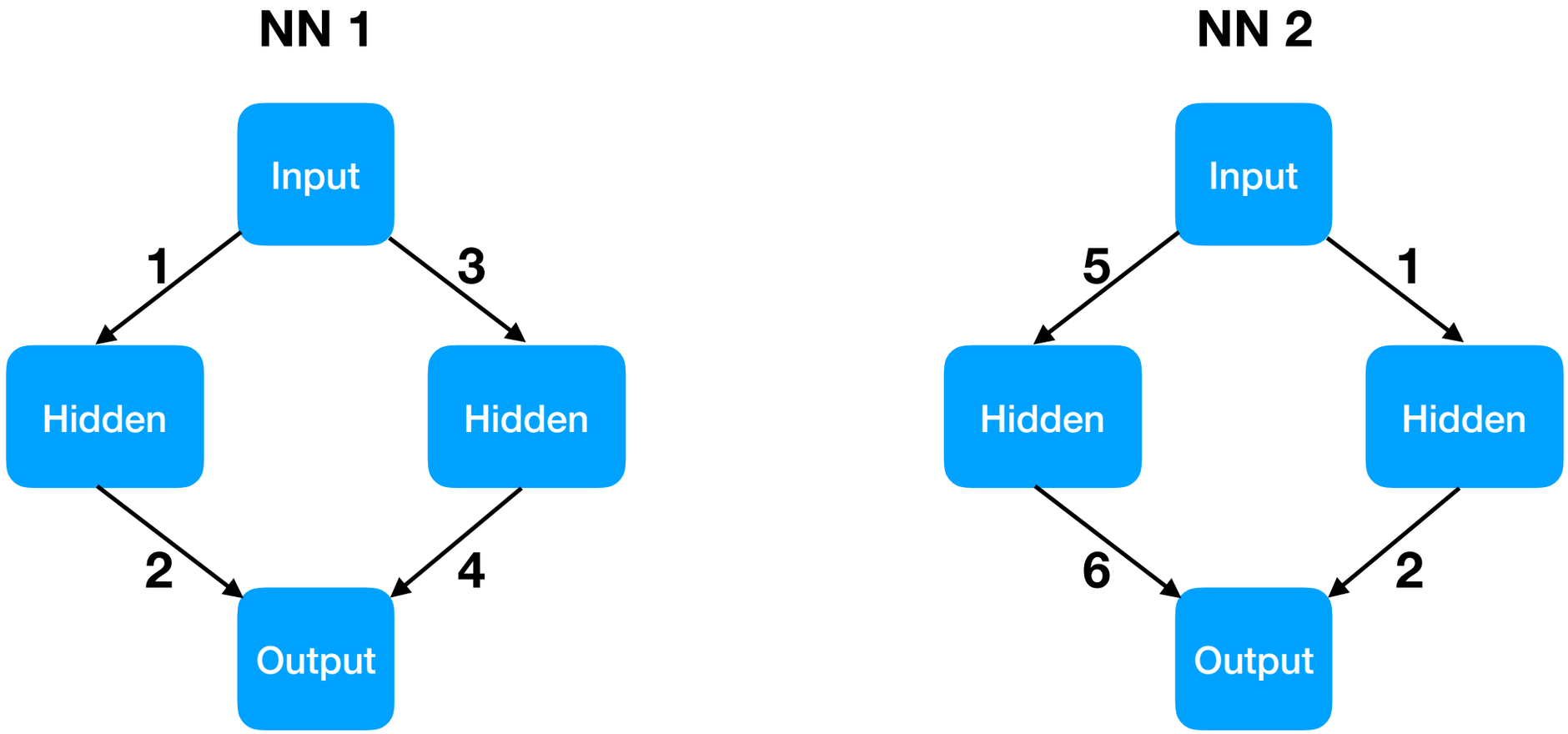
- NEAT assigns a unique "innovation number" to each newly generated edge.
- This allows NN graphs to be compared in linear (assuming edges are sorted according to innovation numbers) time - otherwise NN graphs could be ambiguous and very computationally expensive to compare.

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



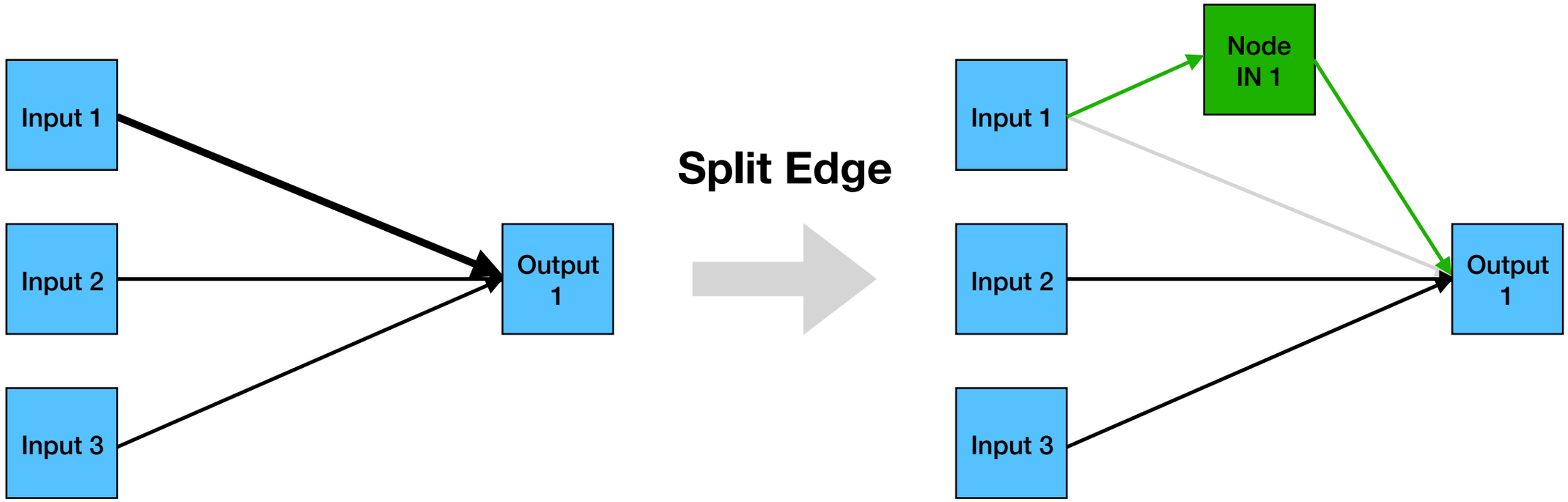
- However, a similar structure may have been generated evolutionarily with "different" edges - in this case they will have different innovation numbers.
- This way we know the edges on the right of NN 2 are the same as those on the left of NN 1, but the other edges occurred through a different evolutionary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

Edge and Node Mutations

RIT | Rochester Institute of Technology

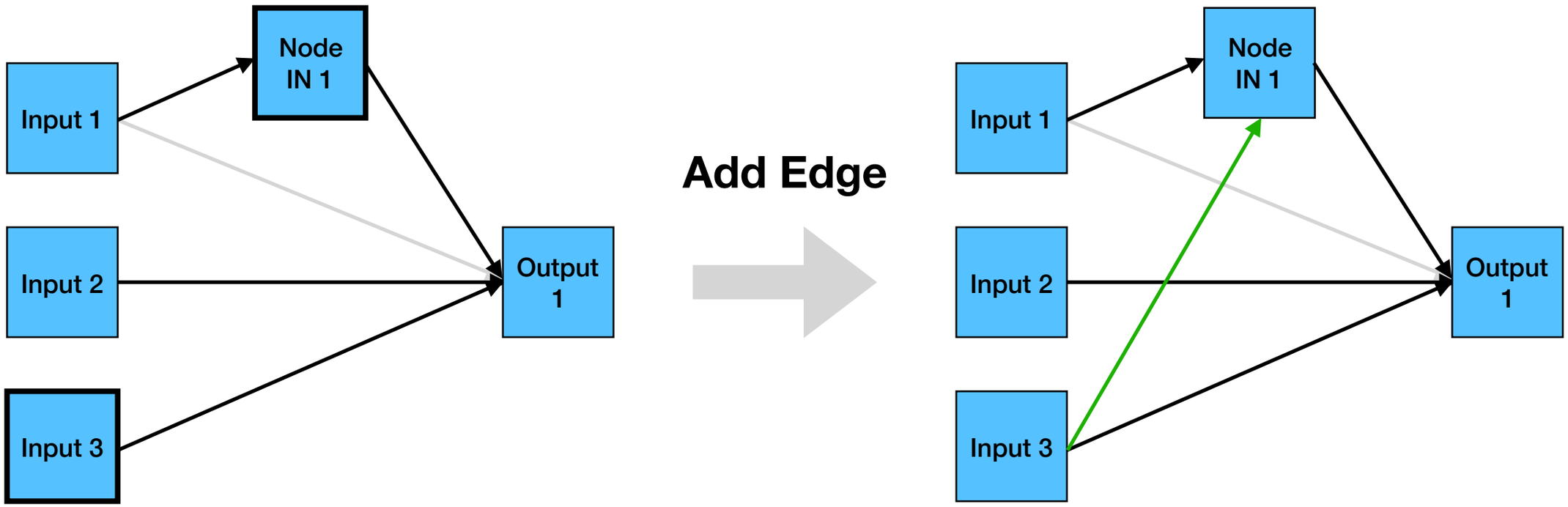
EvoStar: EvoApplications
April 26, 2019

Edge Mutations: Split Edge



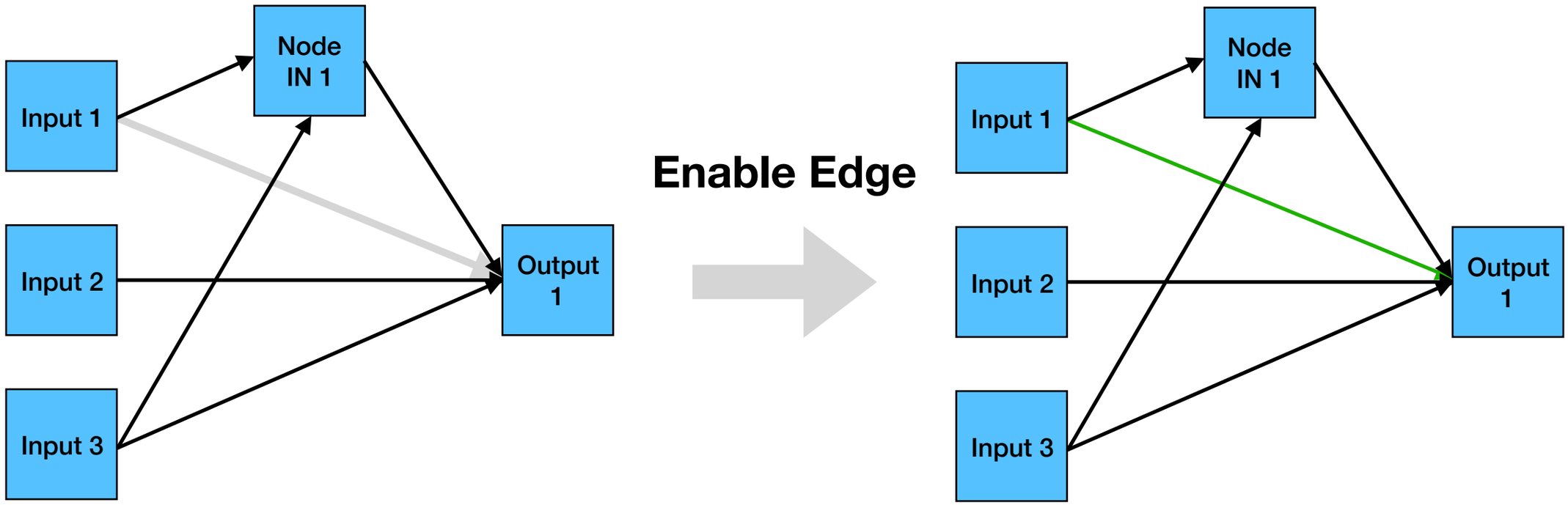
- EXALT always starts with a minimal feed forward network (top left) with input nodes for each input parameter fully connected to output nodes for each output parameter (no hidden nodes).
- The edge between Input 1 and Output 1 is selected to be split. A new node with innovation number (IN) 1 is created.

Edge Mutations: Add Edge



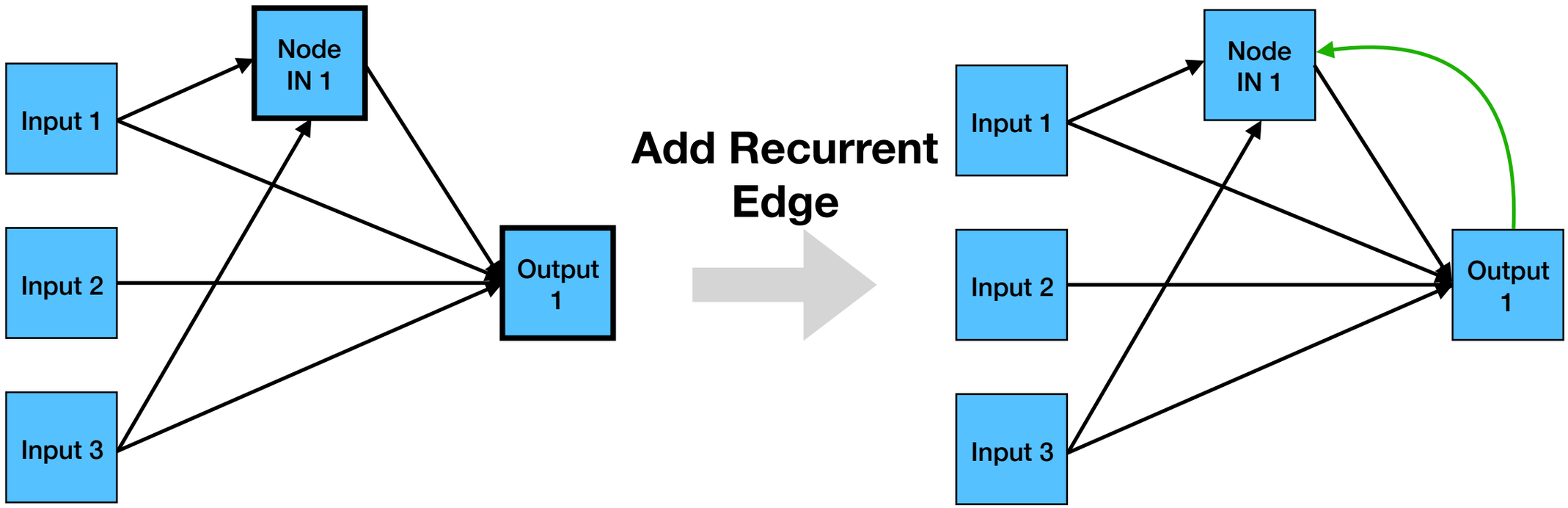
- Input 3 and Node IN 1 are selected to have an edge between them added.

Edge Mutations: Enable Edge



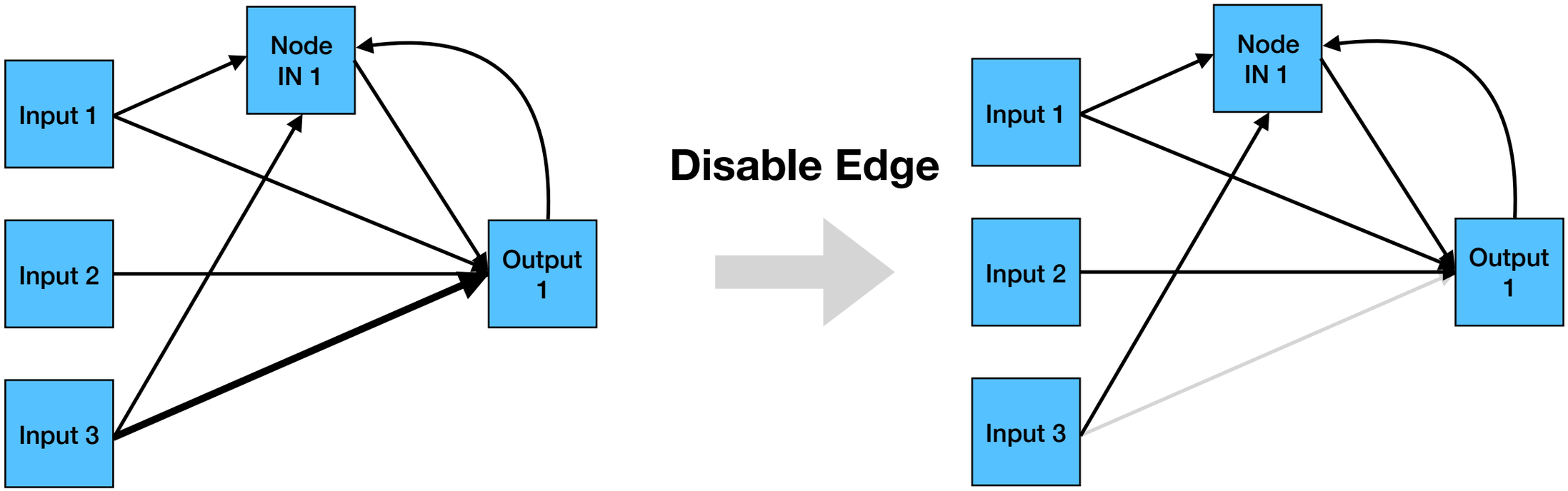
- The edge between Input 3 and Output 1 is enabled.

Edge Mutations: Add Recurrent Edge



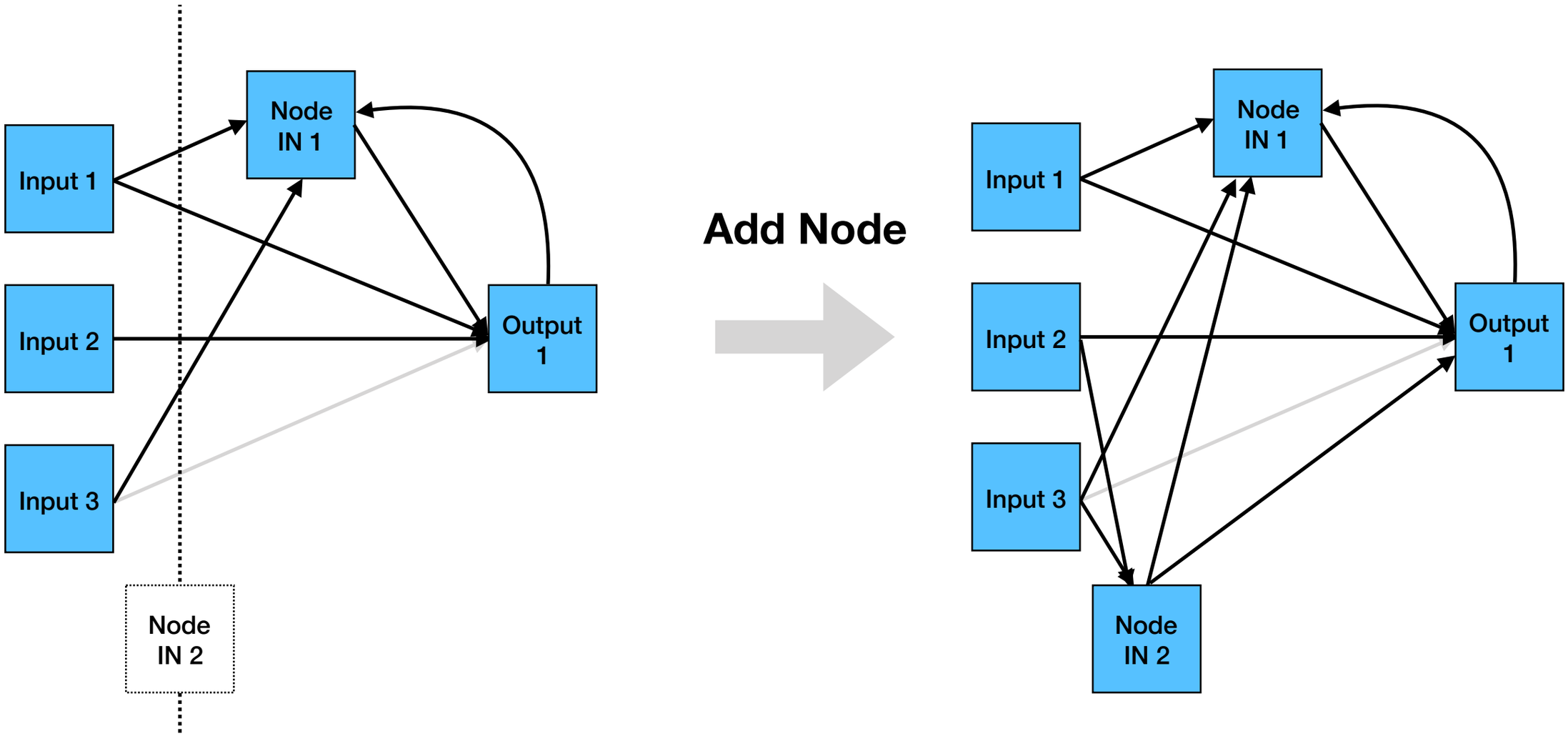
- A recurrent edge is added between Output 1 and Node IN 1.

Edge Mutations: Disable Edge



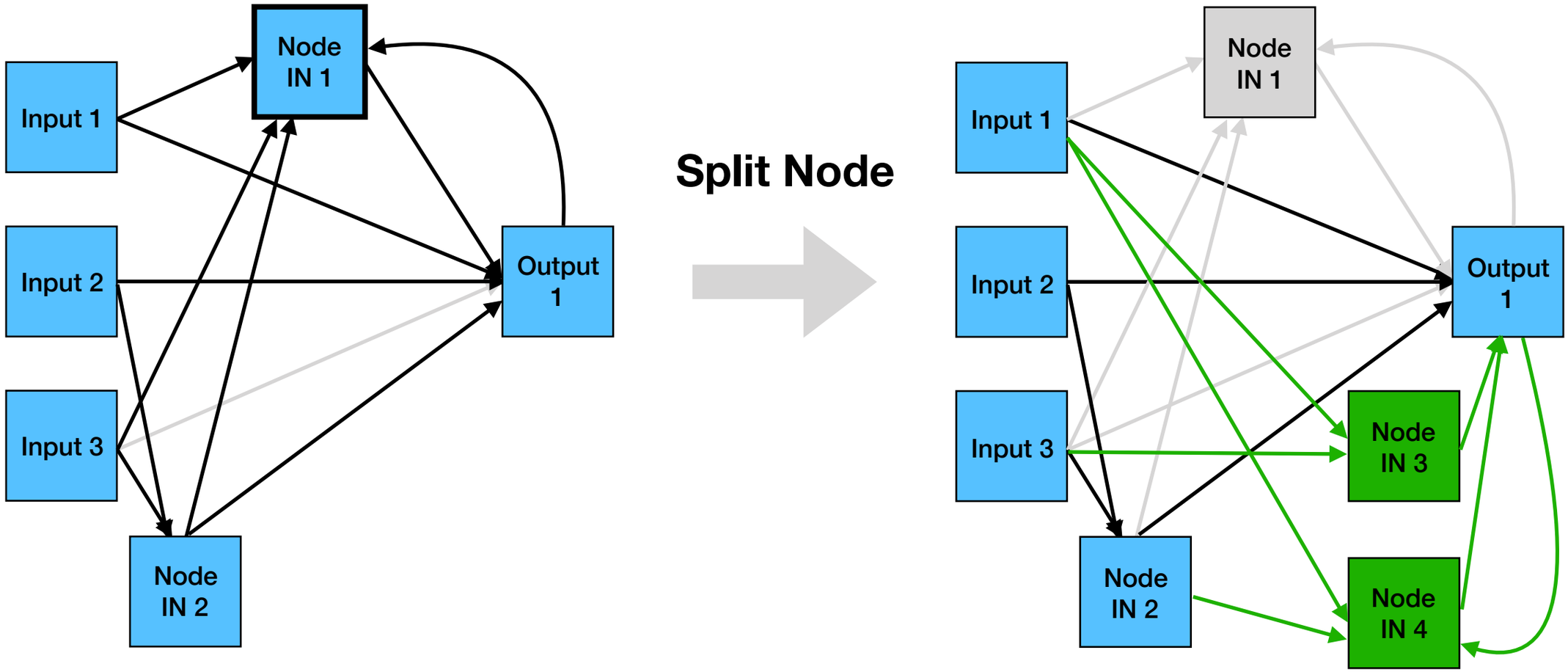
- The edge between Input 3 and Output 1 is disabled.

Node Mutations: Add Node



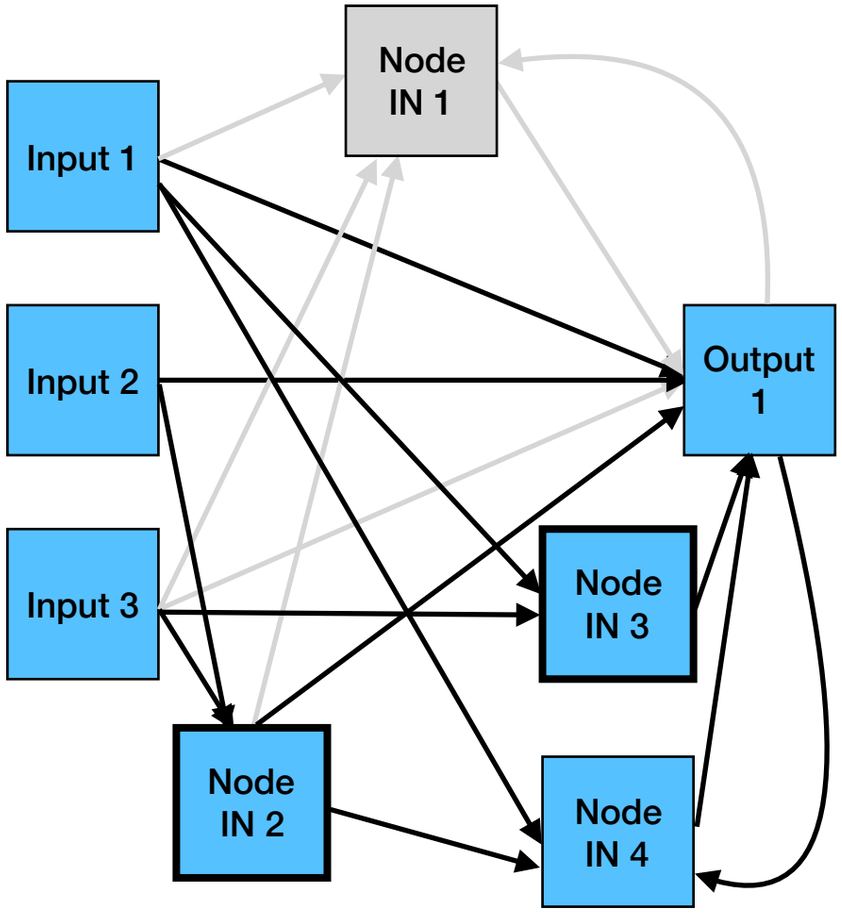
- A node with IN 2 is selected to be added at a depth between the inputs & Node IN 1. Edges are randomly added to Input 2 and 3, and Node IN 1 and Output 1.

Node Mutations: Split Node

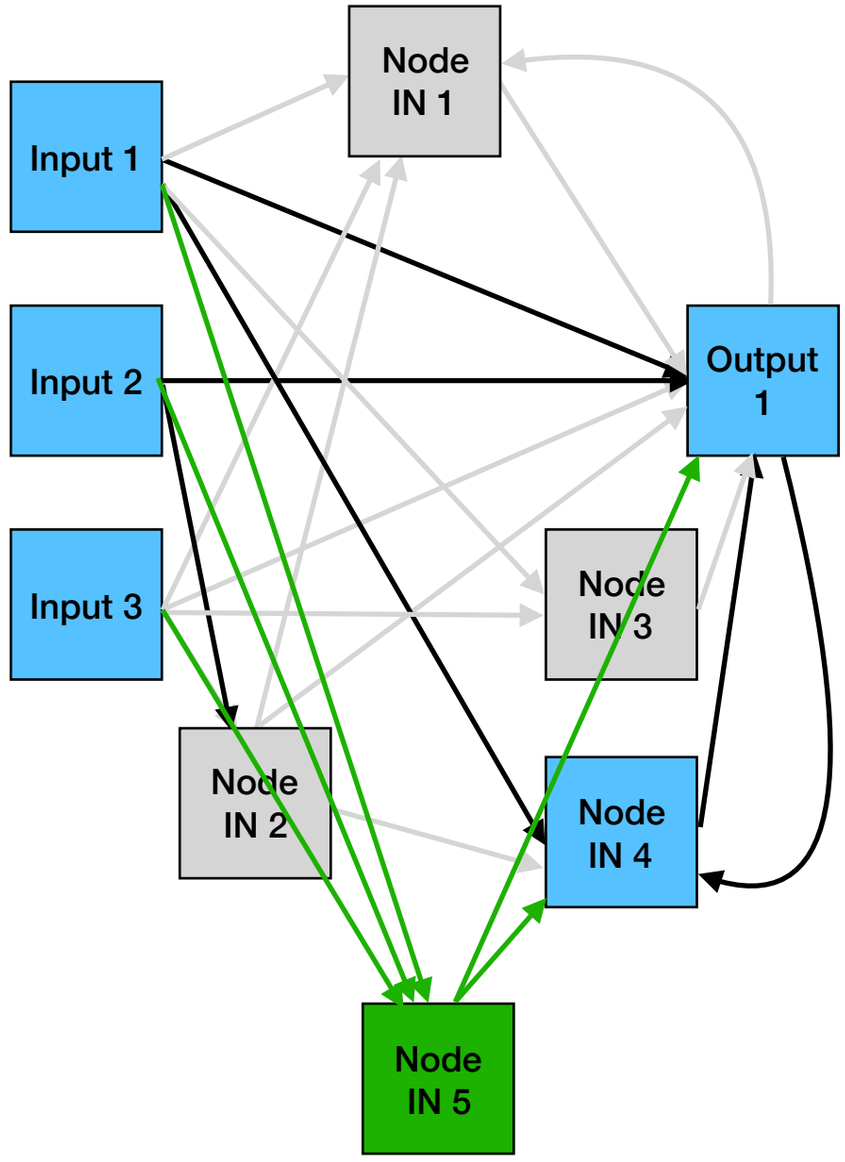


- Node IN 1 is selected to be split. It is disabled with its input/output edges. It is split into Nodes IN 3 and 4, which get half the inputs. Both have an output edge to Output 1 since there was only one output from Node IN 1.

Node Mutations: Merge Node

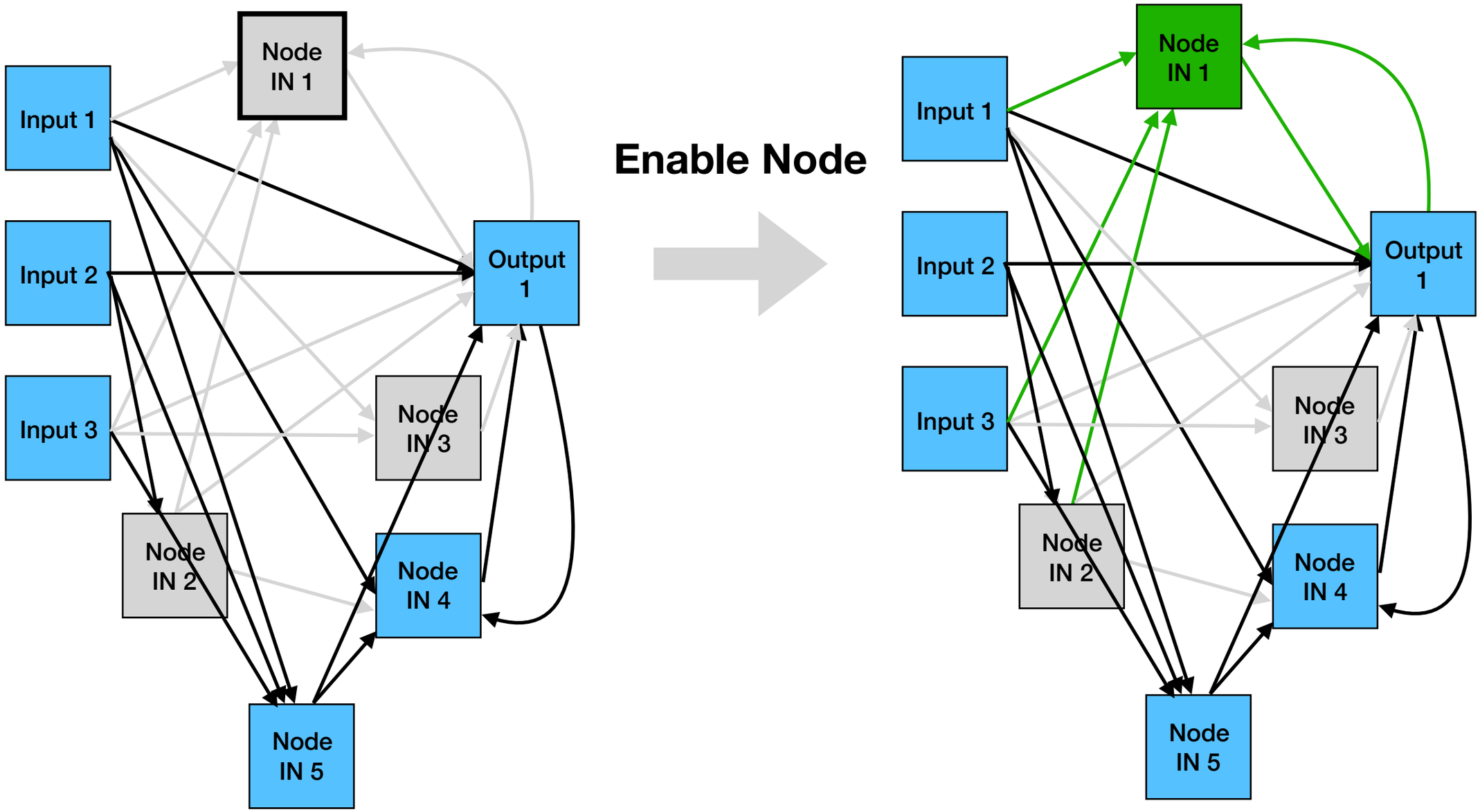


Merge Node



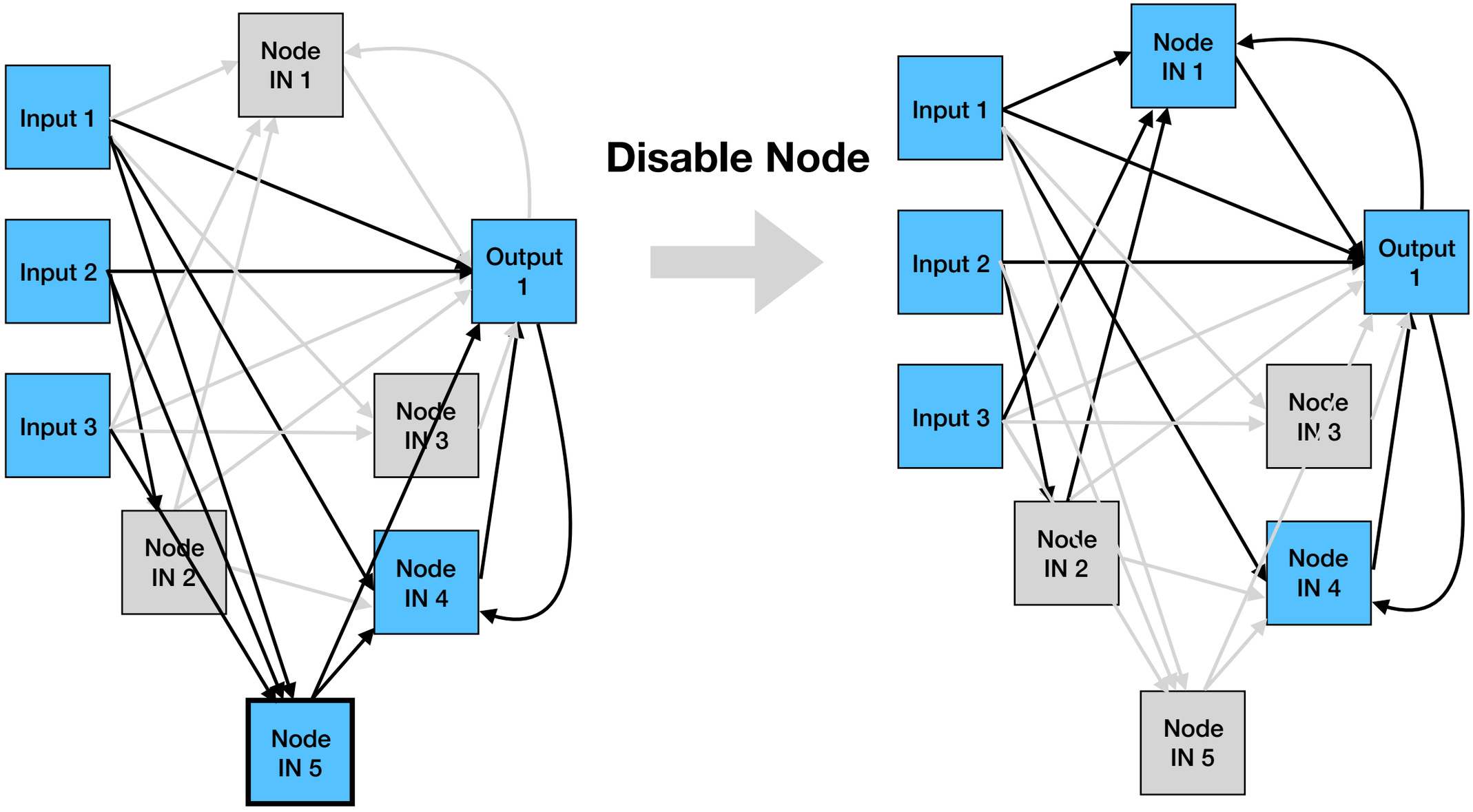
- Node IN 2 and 3 are selected for a merge (input/output edges are disabled). Node IN 5 is created with edges between all their inputs/outputs.

Node Mutations: Enable Node



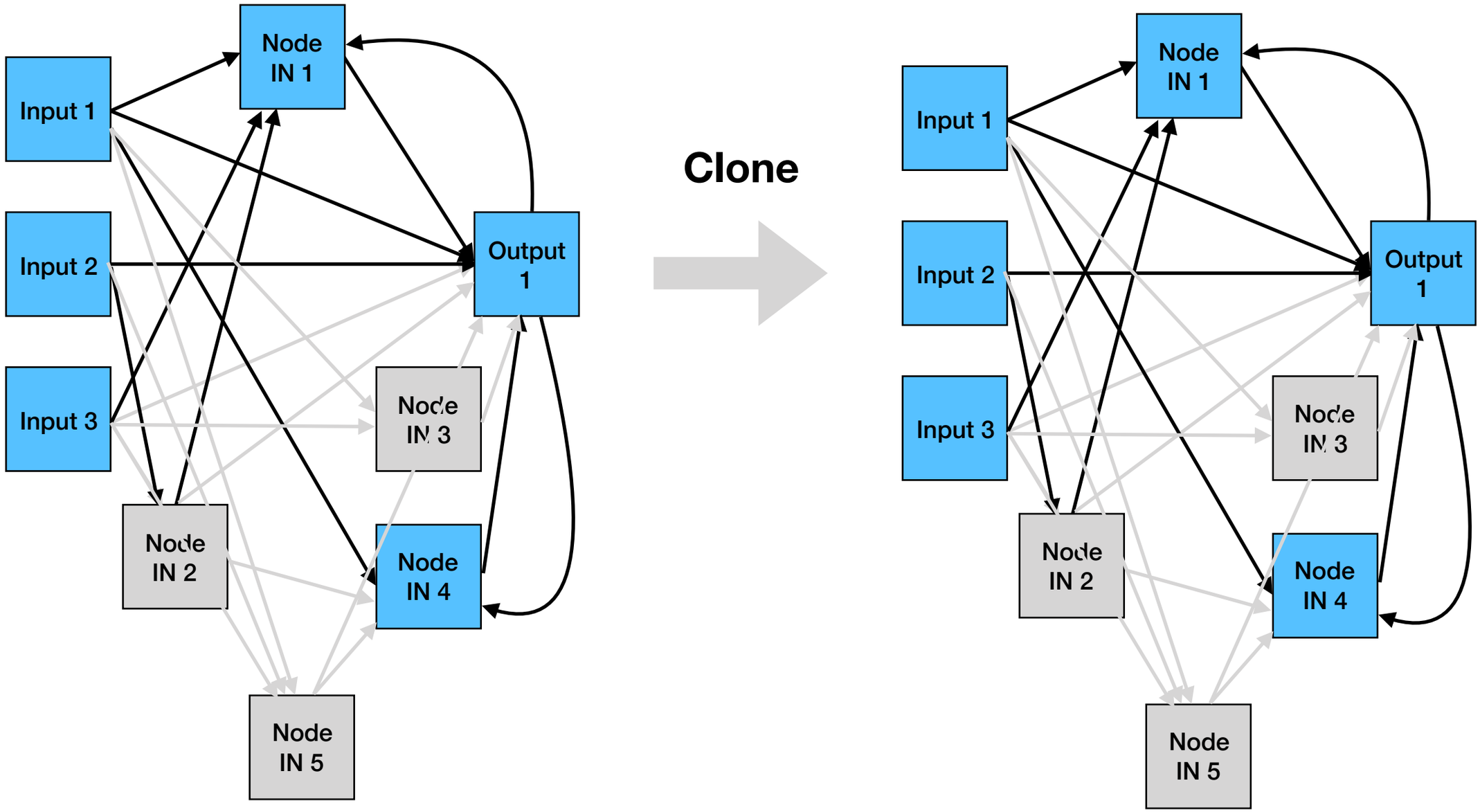
- Node IN 1 is selected to be enabled, along with all its input and output edges.

Node Mutations: Disable Node



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

Clone

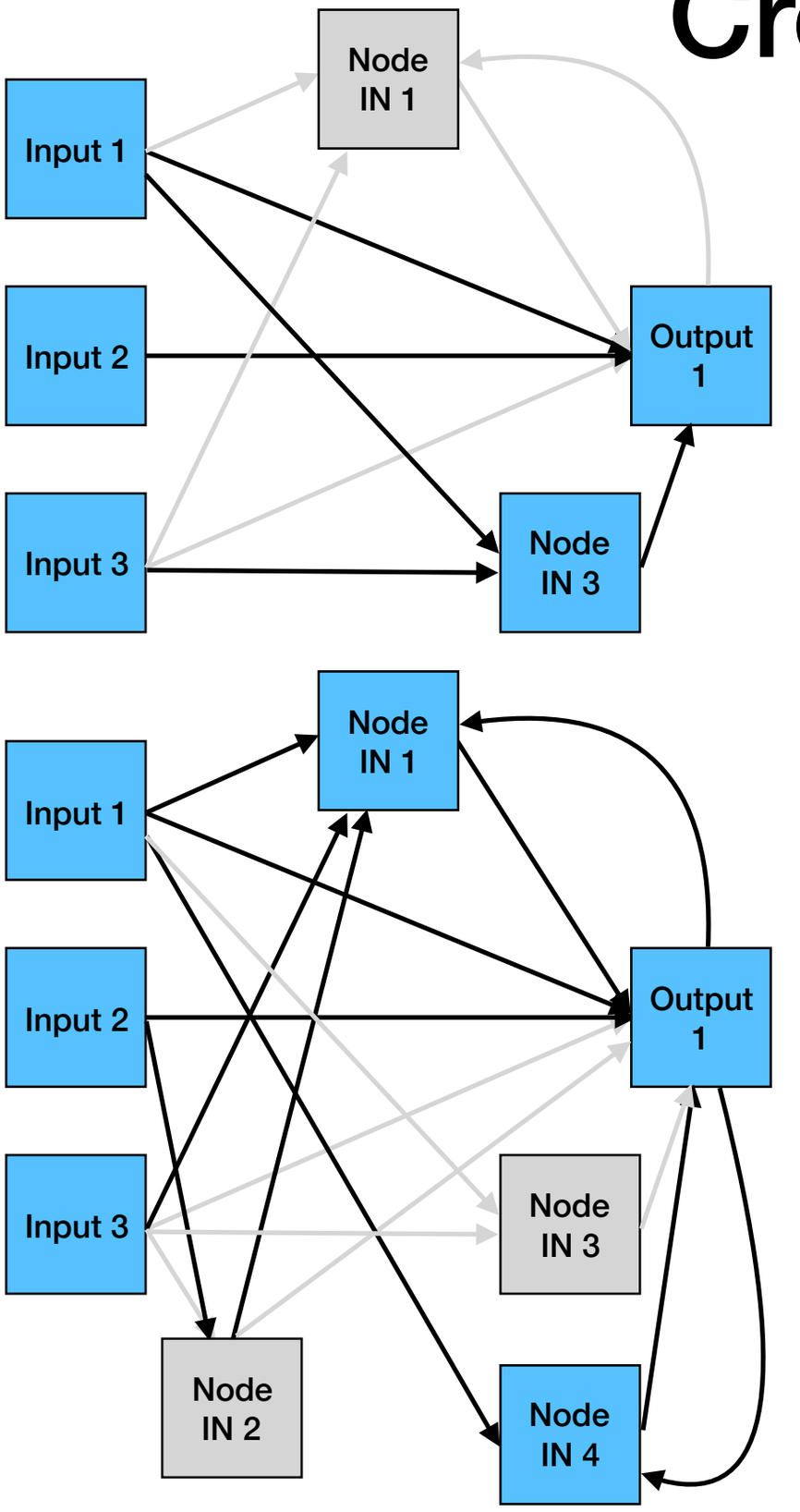


- Clone makes no modifications at all to the parent, allowing it to continue with the back propagation process.

Crossover

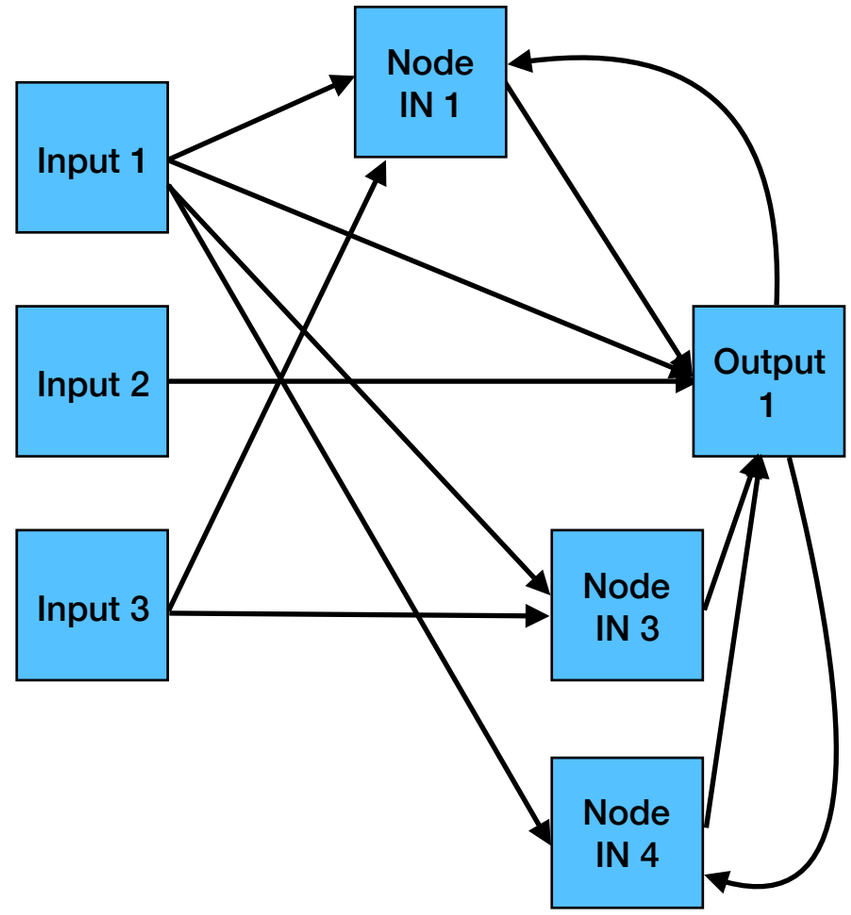
Crossover

Worse Parent



Better Parent

Crossover

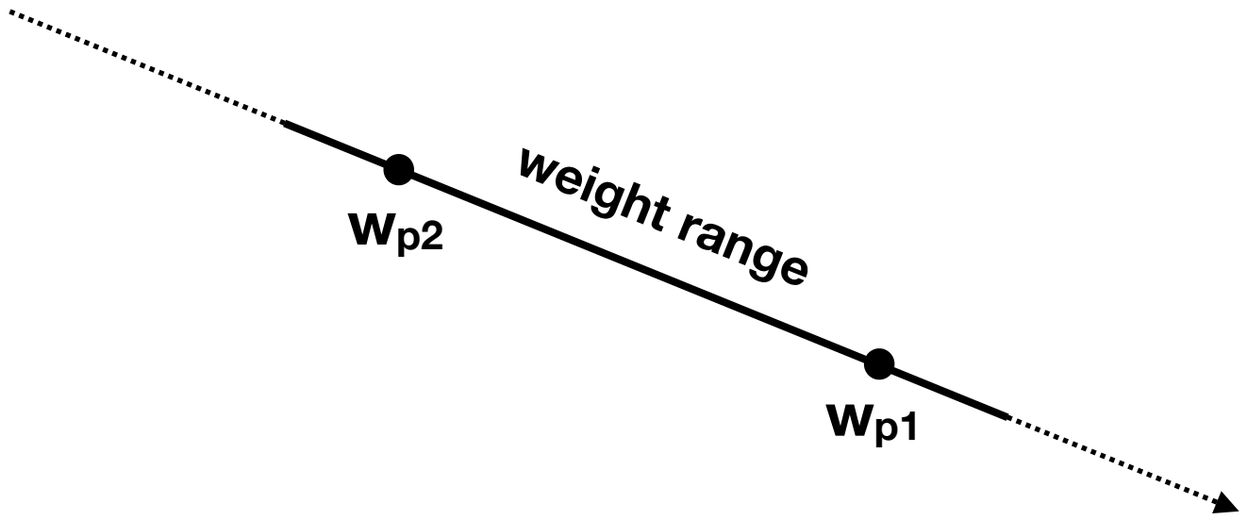


- Crossover creates a child RNN using all reachable nodes and edges from two parents. A node or edge is reachable if there is a path of enabled nodes and edges from an input node to it as well as a path of enabled nodes and edges from it to an output node, i.e., a node or edge is reachable if it actually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

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

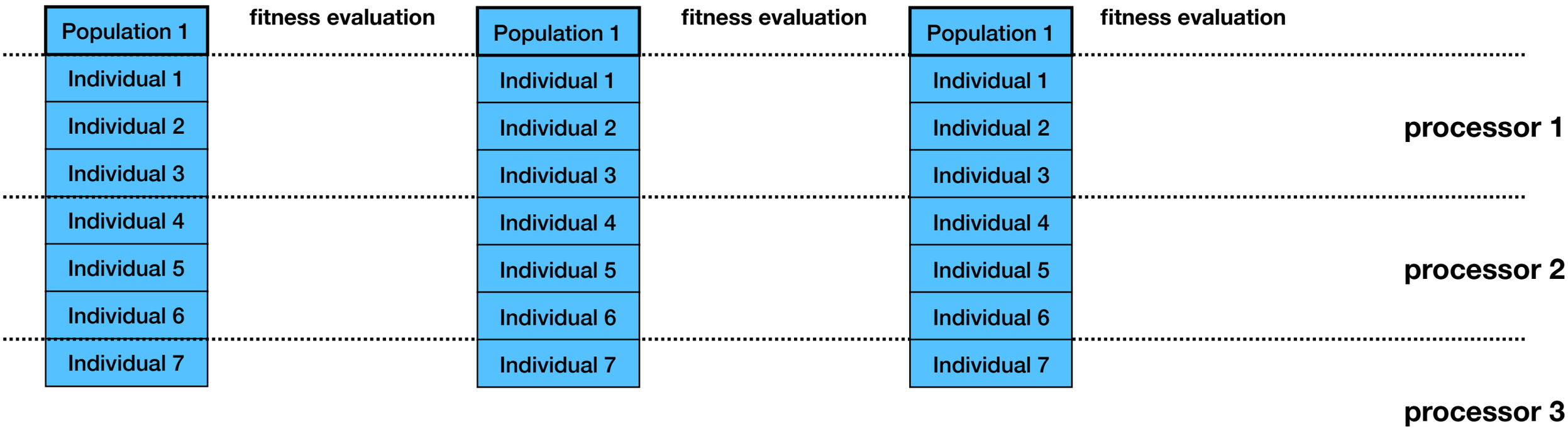
- Where r is a random number -0.5 <= r <= 1.5, where w_{p1} is the weight from the more fit parent, and w_{p2} is the weight from the less fit parent. We can change r's bounds to prefer weights near one parent over the other.

Distributed Neuro-Evolution

RIT | Rochester Institute of Technology

EvoStar: EvoApplications
April 26, 2019

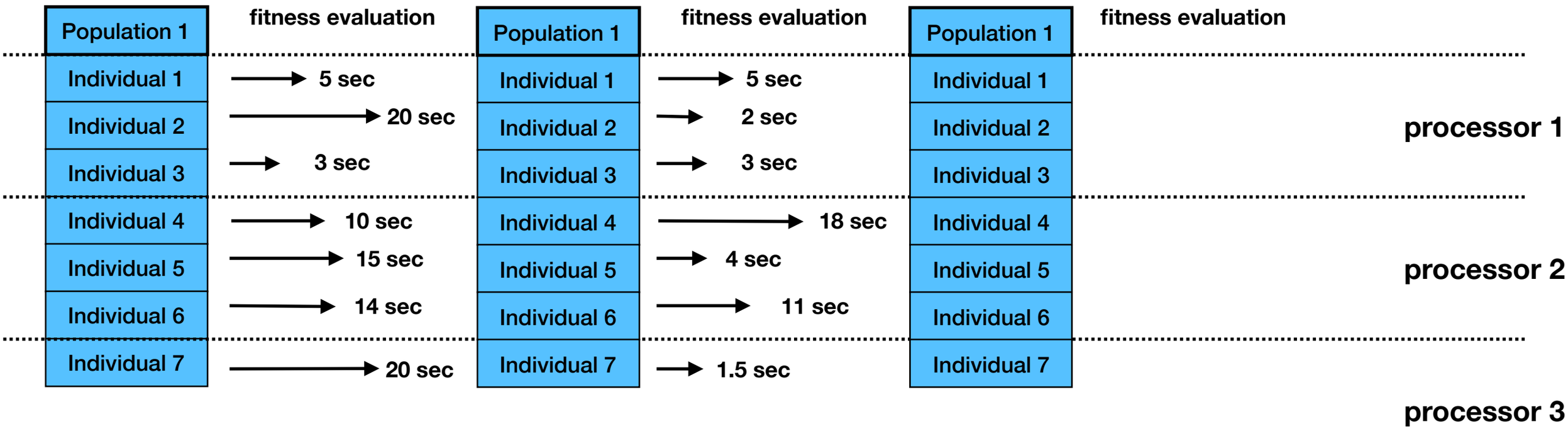
Synchronous/Parallel EAs



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

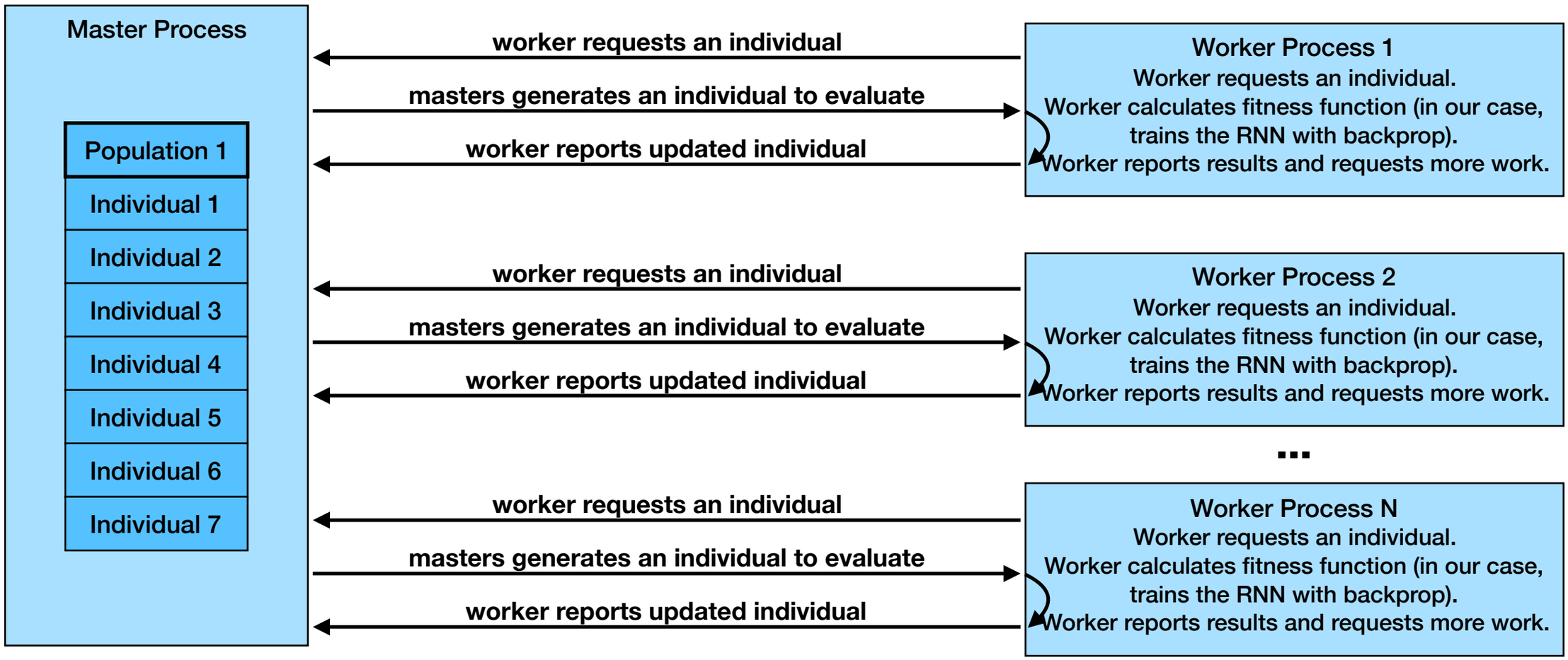
- This has problems in that if the population size is not evenly divisible by the number of processors available there is wasted computation. Also, the population size can't be *less* than the number of processors.

Synchronous/Parallel EAs



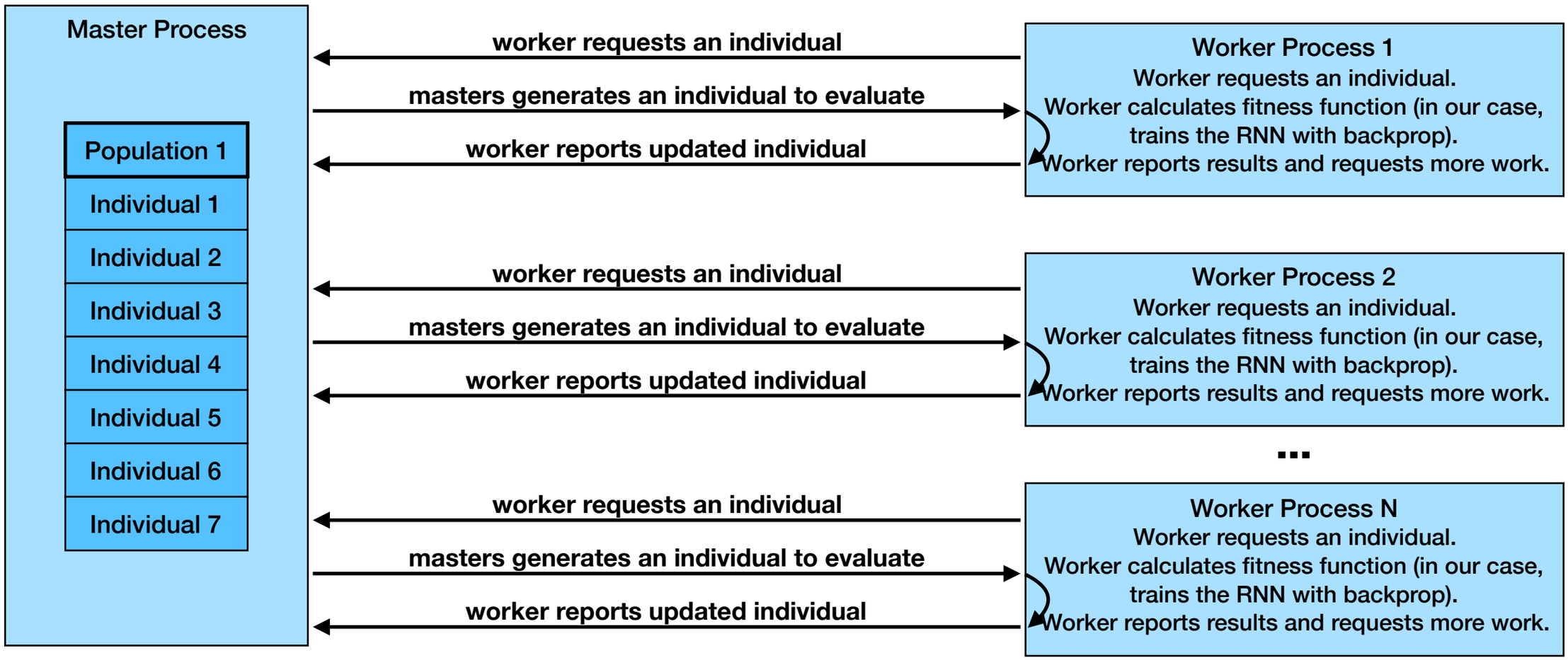
- Things are even more challenging if the fitness evaluation times of the individuals are different or even worse nondeterministic. Lots of waiting and unused cycles.

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.

Data Sets

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 Main Flame Intensity
- Data made public on github repo. Pre-normalized and anonymized.
- 12 coal plant burners from a DOE award with Microbeam Technologies, Inc.
  - 10 days long
  - per minute readings
  - 12 parameters

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 runs utilized different compute notes as determined by RC's SLURM scheduler.

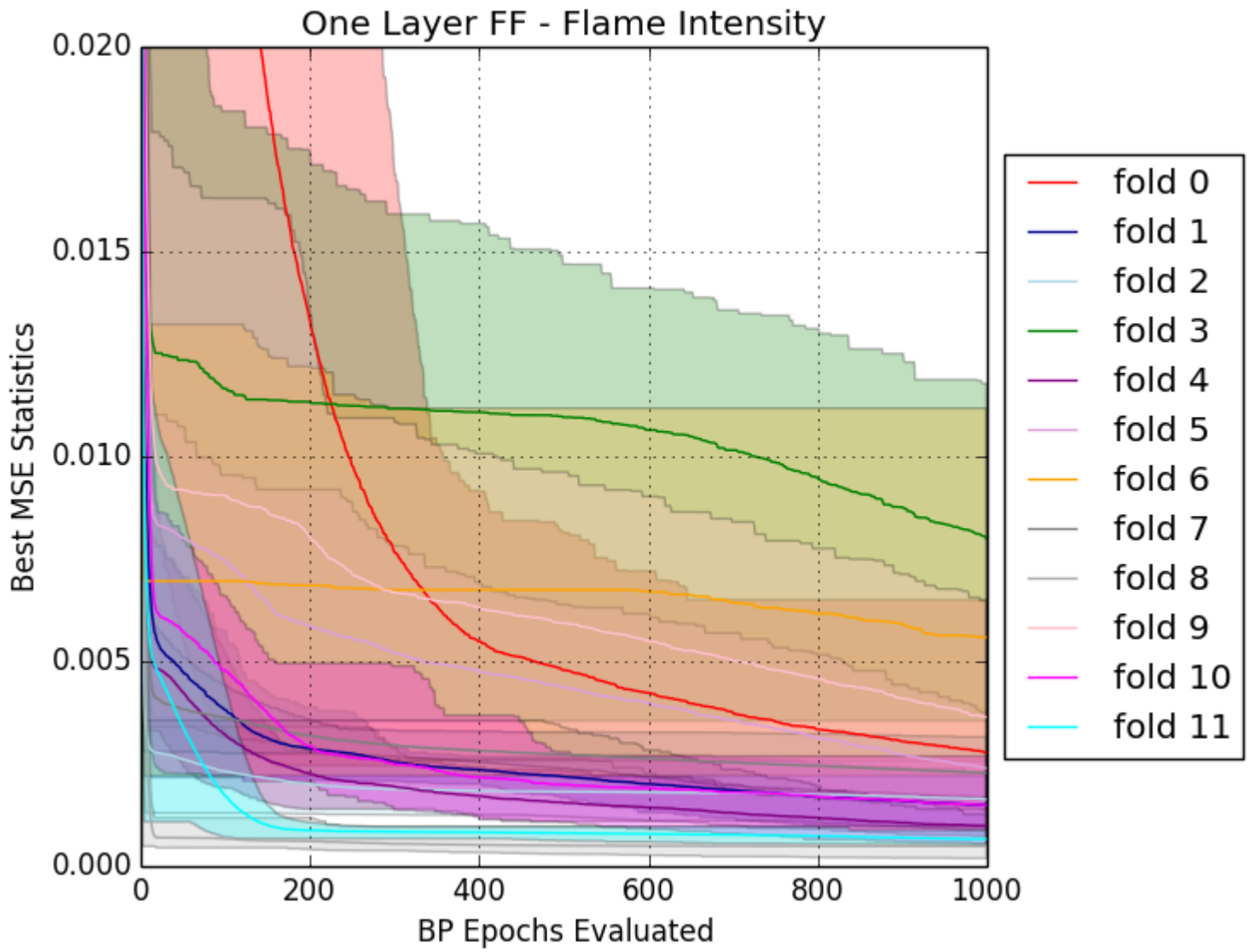
EXALT Experimental Setup

- EXALT compared to traditional RNNs (1-layer FF, 2-layer FF, 1-layer LSTM, 2-layer ISTM, 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.

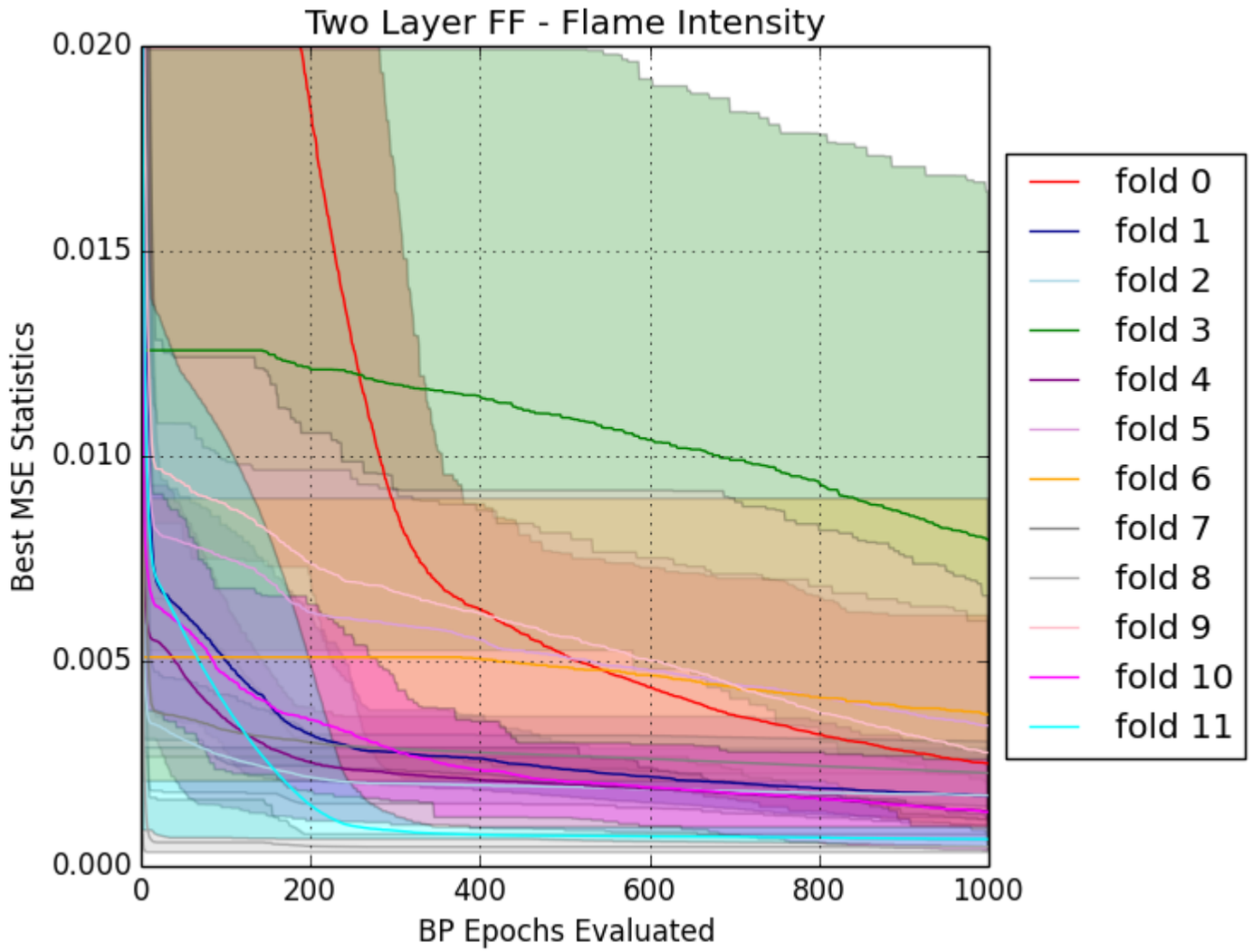
- Fixed RNNs trained for 1000 epochs.
- EXALT trained 2000 RNNs for 10 epochs each, distributed across 20 processes -- compute was expected to be somewhat comparable.
- 12 folds x 10 repeats x 2000 RNNs = 2.4m RNNs trained!

1 Layer Feed Forward



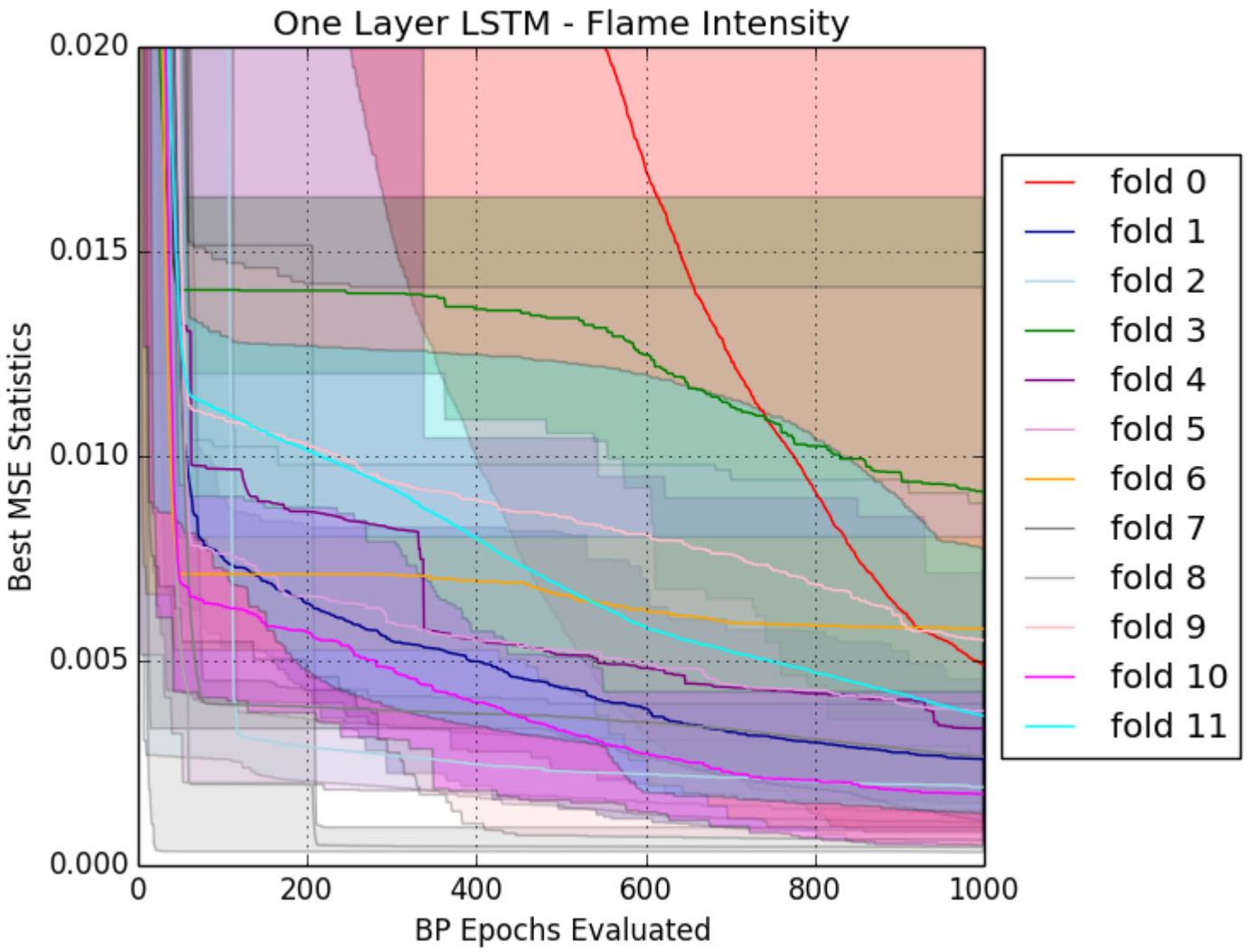
- Min/avg/max mean squared error while training for each fold.

2 Layer Feed Forward



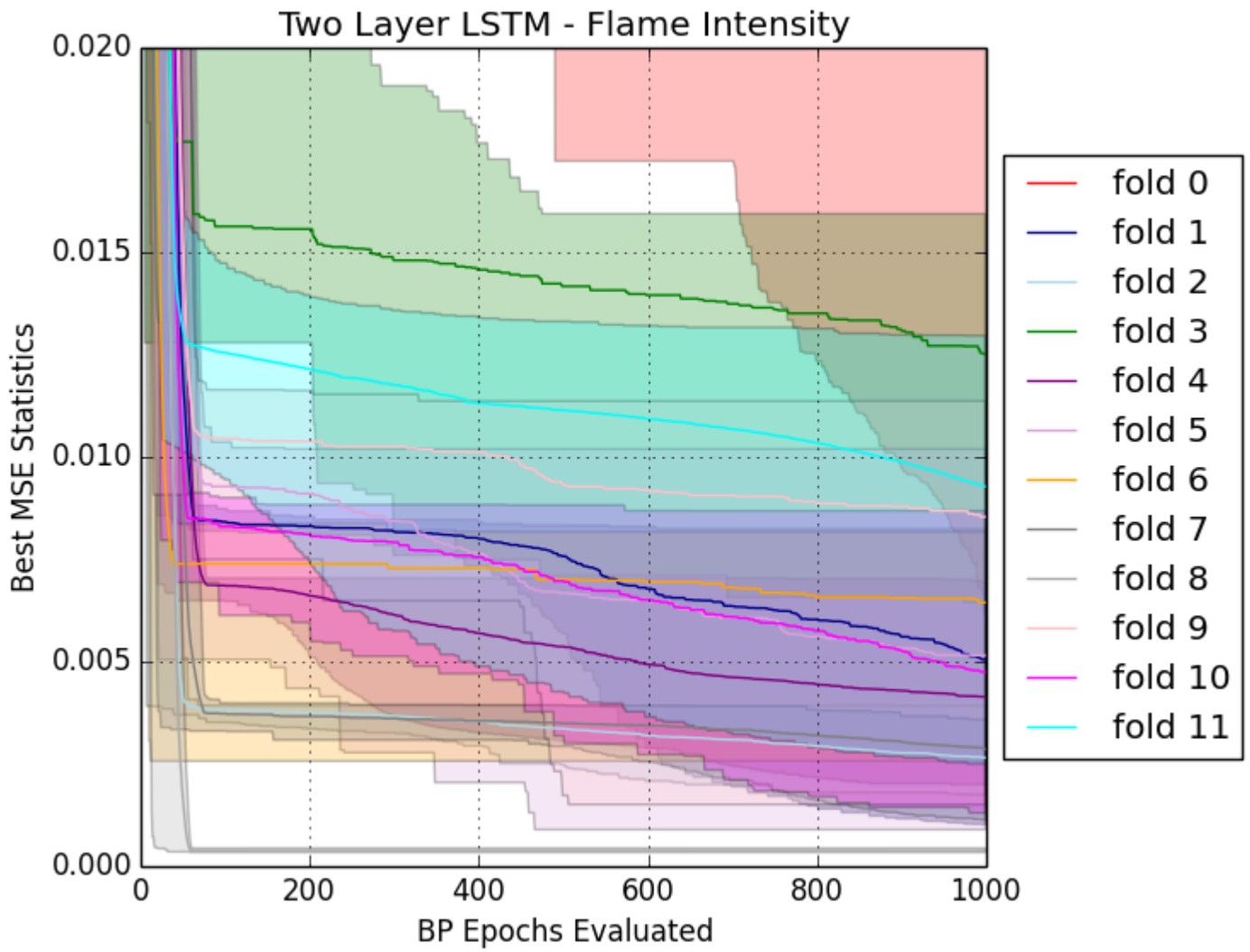
- Min/avg/max mean squared error while training for each fold.

1 Layer LSTM



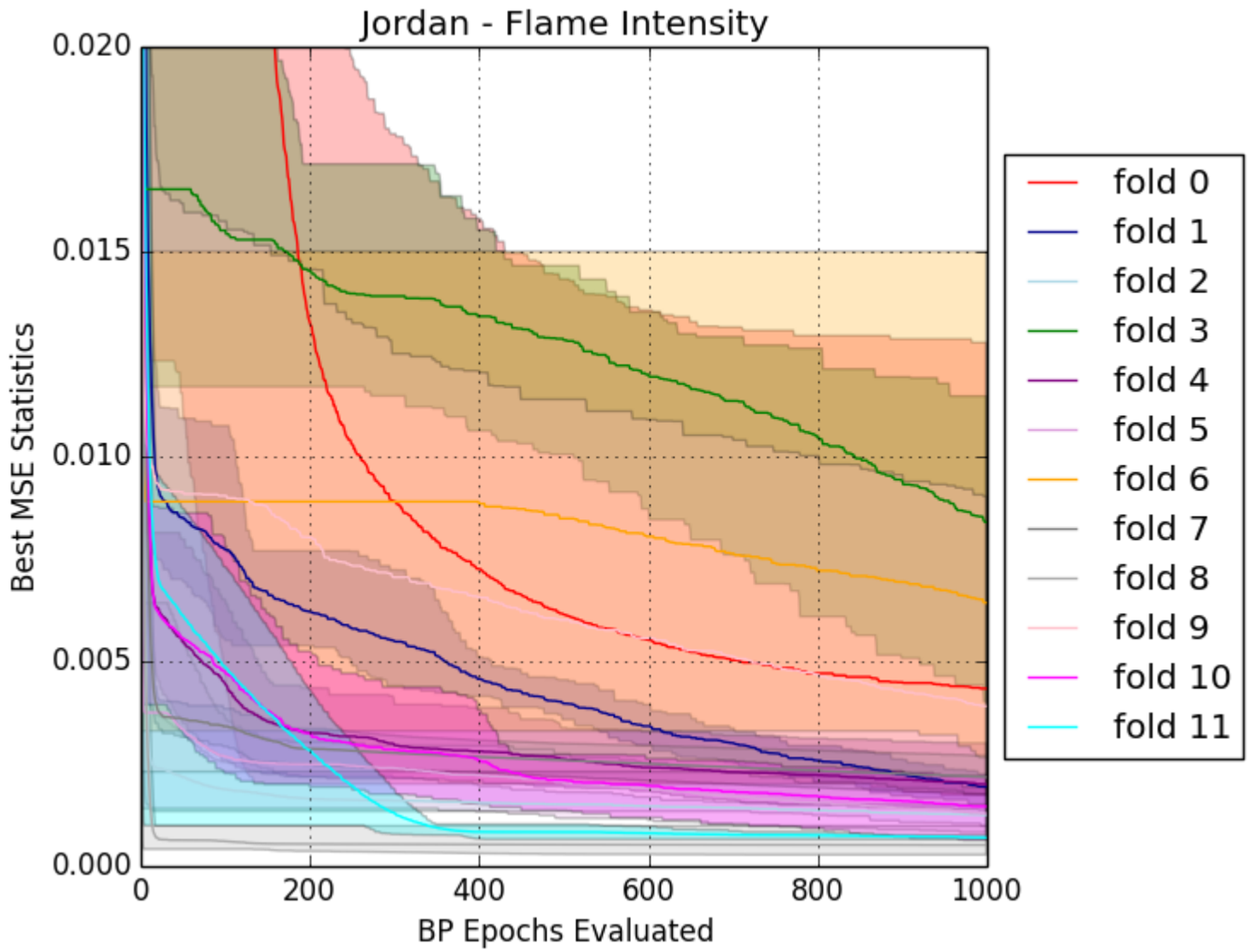
- Min/avg/max mean squared error while training for each fold.

2 Layer LSTM



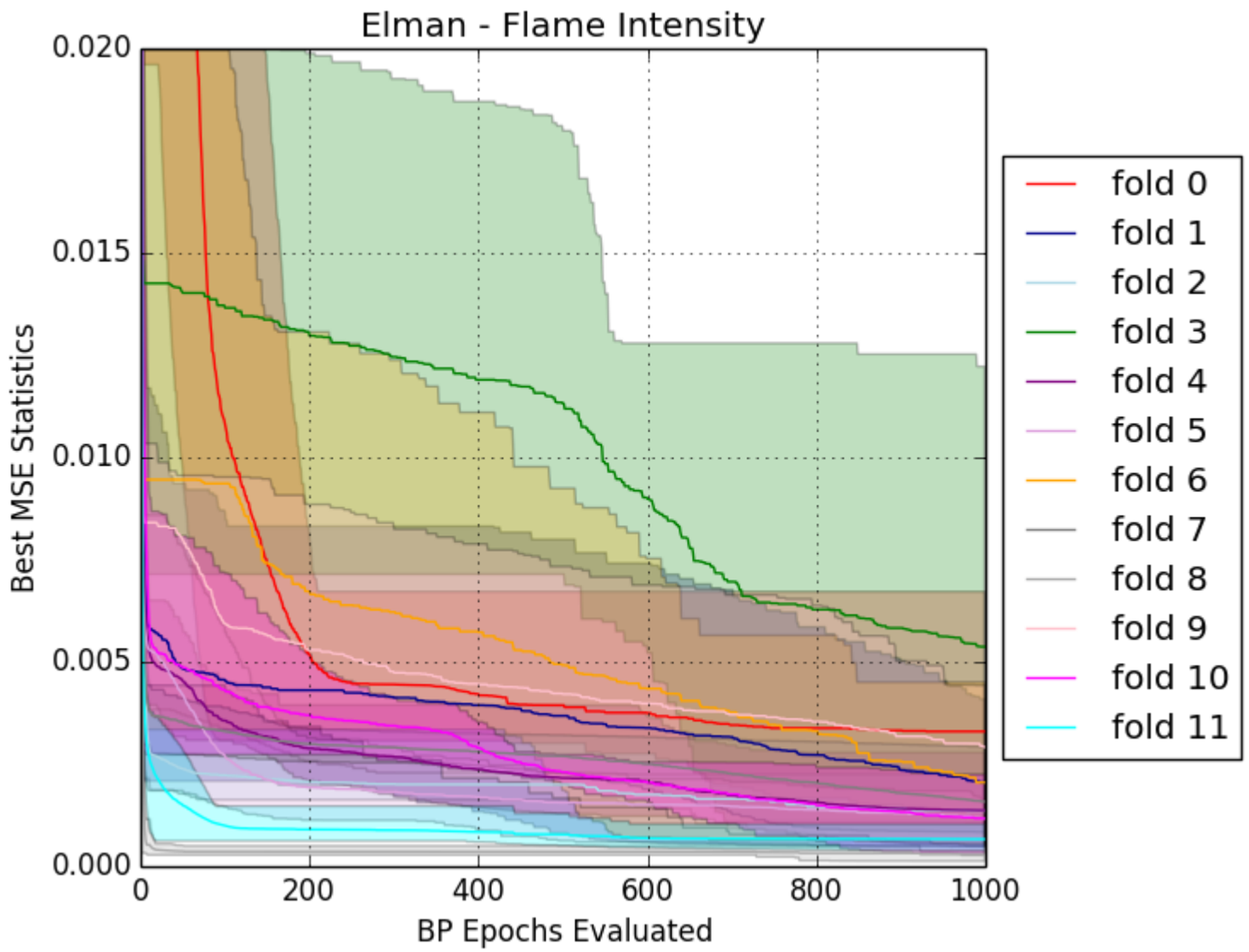
- Min/avg/max mean squared error while training for each fold.

Jordan



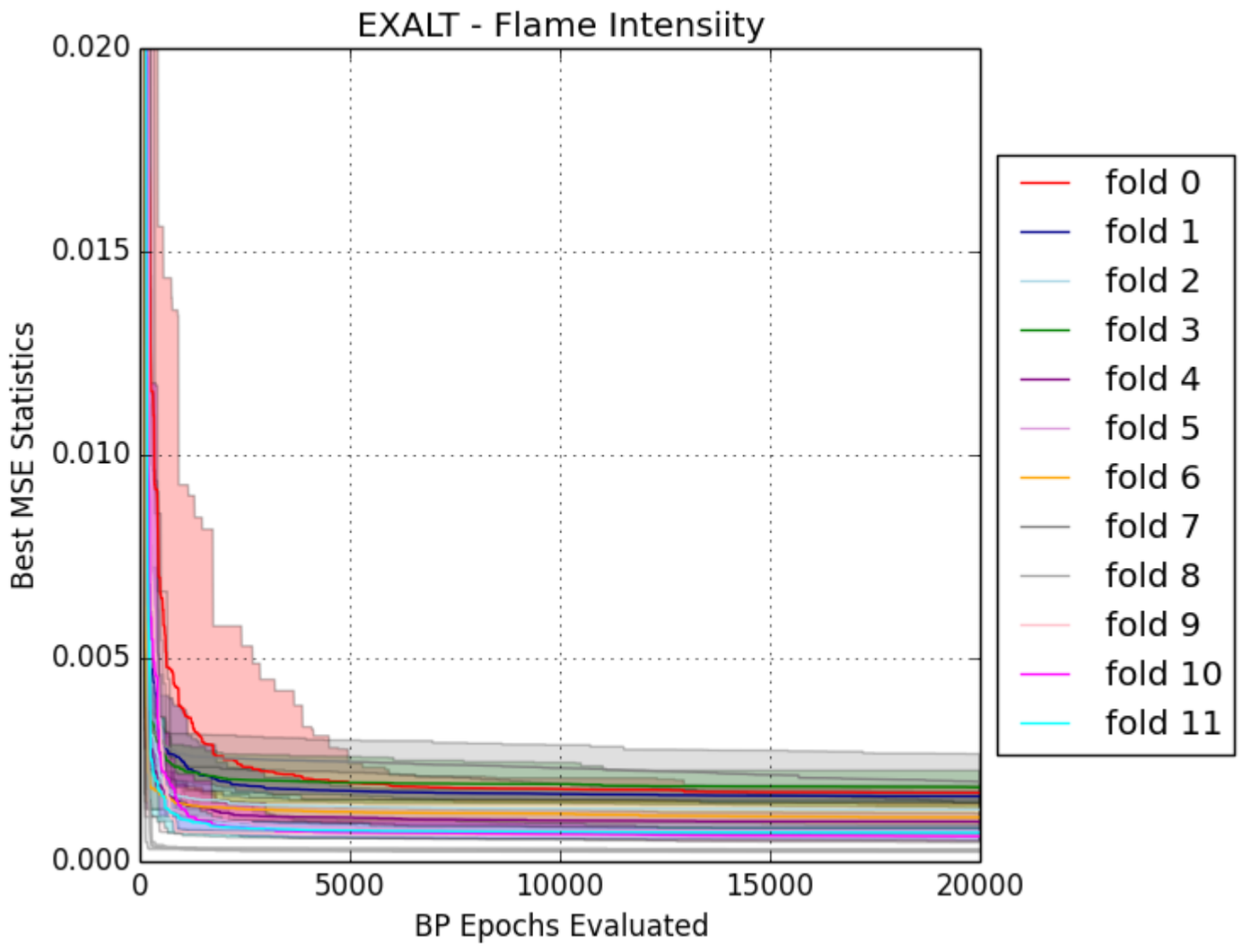
- Min/avg/max mean squared error while training for each fold.

Elman



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

EXALT



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

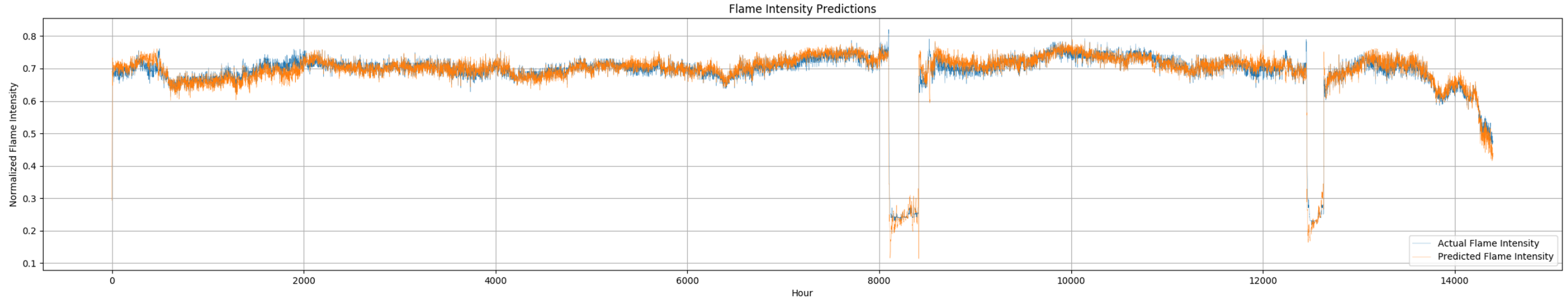
EXALT Results

	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).

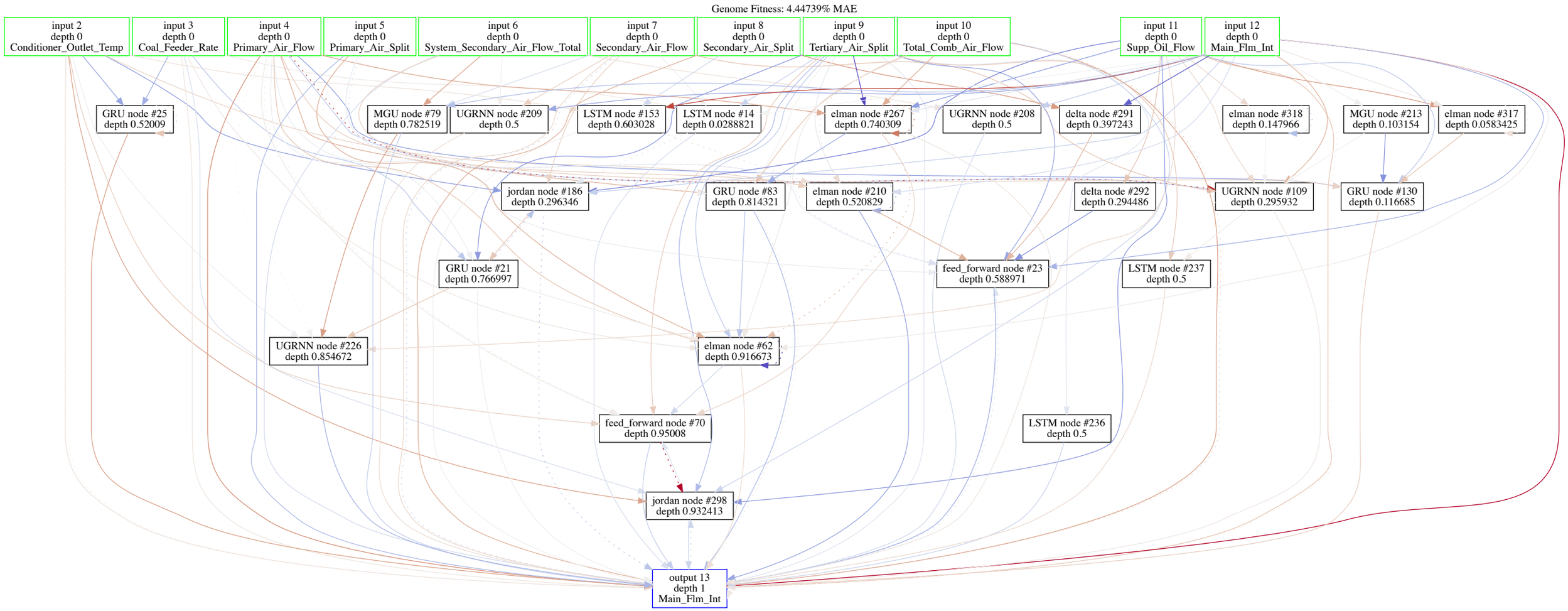
- However, some of the fixed RNNs did find slightly better performance in the best case across all the repeats.

Recent Prediction Example



- RNN evolved on burners 1-10 used to predict unseen data from burner 11.

Recent Evolved Network (EXAMM)



- Example evolved RNN from new version of EXALT, which also allows Jordan, Elman, GRU, MGU, UGRNN, and Delta-RNN neurons in addition to LSTM neurons (see upcoming GECCO paper).

Future Work

- Investigating other cell structures (GRU, MGU, UGRNN, Delta-RNN) and island parallelism for speciation -- results at GECCO! [1]
- Can we optimize hyperparameters concurrently with architecture? Have had some success evolving CNNs this way [2].
- Can we evolving memory cell structures AND architecture simultaneously?
- Testing multiple activation functions (tanh mostly used in this work)
- Layer-level mutations to further speed evolution
- Self-tuning mutation/crossover rates?

[1] Travis Desell. **Developing a Volunteer Computing Project to Evolve Convolutional Neural Networks and Their Hyperparameters.** *The 13th IEEE International Conference on eScience (eScience 2017).* Auckland, New Zealand. October 24-27 2017.

[2] Alex Ororbia, AbdElRahman ElSaid, and Travis Desell. **Investigating Recurrent Neural Network Memory Structures using Neuro-Evolution.** *The Genetic and Evolutionary Computation Conference (GECCO 2019).* Prague, Czech Republic. July 13-17, 2019.

Discussion/Questions?

Source code and Data: https://github.com/travisdesell/exact

Acknowledgements: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Combustion Systems under Award Number #FE0031547.