Controllability, Multiplexing, and Transfer Learning in Networks using Evolutionary Learning

Rise Ooi*
Hokkaido University
rise@rysery.com

C.-H. Huck Yang*
KAUST*, Georgia Tech
chao-han.yang@kaust.edu.sa

Pin-Yu Chen
IBM Research
pin-yu.chen@ibm.com

Vìctor Eguìluz
IFISC (CSIC-UIB)
victor@ifisc.uib-csic.es

Narsis Kiani
Karolinska Institute
narsis.kiani@ki.se

Hector Zenil
Karolinska Institute
hector.zenil@ki.se

David Gomez-Cabrero
Navarrabiomed
david.gomez.cabrero@navarra.es

Jesper Tegnèr
KAUST*
jesper.tegner@kaust.edu.sa

Abstract

Networks are fundamental building blocks for representing data, and computations. Remarkable progress in learning in structurally defined (shallow or deep) networks has recently been achieved. Here we introduce evolutionary exploratory search and learning method of topologically flexible networks under the constraint of producing elementary computational steady-state input-output operations.

Our results include; (1) the identification of networks, over four orders of magnitude, implementing computation of steady-state input-output functions, such as a band-pass filter, a threshold function, and an inverse band-pass function. Next, (2) the learned networks are technically controllable as only a small number of driver nodes are required to move the system to a new state. Furthermore, we find that the fraction of required driver nodes is constant during evolutionary learning, suggesting a stable system design. (3), our framework allows multiplexing of different computations using the same network. For example, using a binary representation of the inputs, the network can readily compute three different input-output functions. Finally, (4) the proposed evolutionary learning demonstrates transfer learning. If the system learns one function A, then learning B requires on average less number of steps as compared to learning B from tabula rasa.

We conclude that the constrained evolutionary learning produces large robust controllable circuits, capable of multiplexing and transfer learning. Our study suggests that network based computations of steady-state functions, representing either cellular modules of cell-to-cell communication networks or internal molecular circuits communicating within a cell, could be a powerful model for biologically inspired computing. This complements conceptualizations such as attractor based models, or reservoir computing.

Introduction

A Turing machine (TM) is a universal mathematical model of computation. Given a table of rules and a tape, TM is essentially a discrete system which can compute any computer algorithm. Furthermore, a function is computable if and only if it can be computed on a TM (Kleene 1971 1952 10th impression 1991). Yet, in addition to what is computed, it is essential to consider how an algorithm is represented, implemented in a specific architecture. Considering learning machines, then a learning algorithm can act on the realized architecture and/or the representation of the computation. Such considerations have resulted in different instantiations of computations. For example, inspired from neural circuits, realized as (deep) neural feed-forward network models of computation (Schmidhuber 2015), the learning generally acts on the connections. Attractor dynamics is another popular conception of neural (Hopfield 1984) computations where the fix-points correspond to memory states or to cell-types as in the case of molecular (gene) based computations (Kauffman 1969).

In this work, we are inspired by the computational processing power in cells, where networks of molecular components collectively sustain robust input signals to the cell resulting in a transformed output signal at the cellular level. In the case of neurons we can represent such a computation as a threshold computation using a ReLU or softmax unit. Yet in general, we do not understand the underlying network of molecular computations occurring within different cells, realizing different input-output transformations. Here we ask if we can search and identify networks that are able to compute different input-output functions. In particular we address whether such system can represent different input-output computations in the same network, whether the computations are controllable in an engineering sense, and if transfer learning can occur under these constraints. To this end we use evolutionary search techniques to evolve networks of interacting elements which collectively compute such a function. We have chosen a vanilla genetic algorithm (GA) (Holland 1992) to illustrate that even one of the simplest evolution algorithms is sufficient to generate defined computational large-scale networks.

Our approach can be viewed as bridging between learning fundamental computational input-output operations and the investigation of the underlying graphical model realizing such a computation without fixating the topology of the graph in advance. Moreover, our work can be related to areas such as transductive and inductive learning with graphs which have recently become a mainstream research focus in artificial intelligence and machine learning for tackling structured yet irregular data inputs (Bronstein et al. 2017; Hamilton, Ying, and Leskovec 2017; Battaglia et al. 2018).

To this end we use evolutionary learning to perform exploratory search for large networks. The generative network models, identified by the evolutionary search, are naturally of broad interest since networks are used as fundamental building blocks for representing data, and computations in biology, social sciences, and communication networks. Given a network there exists a large number of tools for searching, navigating, and characterizing properties of the network which as a rule has strong predictive power in numerous application domains. Our work can be viewed as addressing the putative nature of computations in different networked system. One premise is that by imposing fundamental computational operations (input-output transformations) we can generate underlying ensembles of networks realizing such computations.

Methods

Learning input-output computations in a Network Equipped with internal Dynamics

We search through different network topologies using a GA (described below). Yet, to compute an input-output function from the full network, we need to equip the intrinsic network structure with a set of dynamical equations, defining how a given node interacts with its neighbours. The form of such equations is qualitatively motivated by chemical reactions and molecular regulatory control systems such as genes in a cell. A non-linear threshold function summing input from nearby nodes. For the node dynamics we use

$$\frac{dy_i}{dt} = k_{1i}(f(\sum_{ij} W_{ij} y_j) + I_i) - k_{2i} y_i \tag{1}$$

where

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

For each node i, y represents the activation or expression rate, f is a nonlinear sigmoid function (Equation 2), W is the connection or interaction weight matrix, I is the input matrix prescribed into the system, and both k_1 and k_2 are constant vectors which represent the maximum expression and kinetic degradation rate respectively. Note that the W_{ij} represents the influence from nodes j to nodes i, and for simplicity, we have set all elements of k_1 and k_2 to 1 for all our experiment – essentially removing them from our equation. The motivation originates from the well known hopfield network (Hopfield 1984), where we follow the modifications as described in (Jaeger et al. 2004; Mjolsness, Sharp, and Reinitz 1991; Reinitz and Sharp 1995) and (Vohradsky 2001; Geard and Wiles 2005; Hiscock 2017).

We solve he ODE numerically using the forward Euler's method in which $y_{n+1} = y_n + f(y_n, t_n)\delta t$ and δt is set to 1. A network of such connected units collectively compute a given steady-state input-output function. Equation 1 represents a special version of a recurrent neural network (Hochreiter and Schmidhuber 1997) and the fully connected network structure looks deceptively similar to a Boltzmann Machine (Ackley, Hinton, and Sejnowski 1985) in which the two visible nodes are the input and output nodes respectively while all other hidden nodes (equipped with the above dynamics) are the supporting nodes. However, the links are directed and mutually influencing between nodes. The GA search for the appropriate links in order to implement the desired input-output function from the system, see Figure 1a.

Learning several input-output in the same network

Now that we have the methods to generate large-scale networks with single input node and single output node using GA, we further extend the framework by generating networks such that any single of them can receive multiple injection inputs and produce multiple output functions. We consider the following cases:

1. A network that has exactly 1 node that receives the injection inputs and multiple nodes that can produce any desired output functions. See Figure 1b.
2. A network that has N nodes that receives the injection inputs and N nodes that can produce any desired output functions. See Figure 1c.
3. A network that has N nodes that receives the injection inputs and $2^N - 1$ nodes that can produce any desired output functions. See Figure 1d.
4. A network that has N nodes that receives the injection inputs and exactly 1 node that can produce $2^N - 1$ number of any desired output functions. See Figure 1e.

For the 3. and 4. scenarios, we essentially encode the activation of the input nodes using a binary representation, whereas in the 4. scenario, the single multiplex output node is intrinsically multifunctional.

Design of the cost function The method of training multifunctional networks is similar to the training of single functional network as additional costs of multiple output functions are merged into the total cost function. There is a caveat, however, that merging all the costs into a single number has the detrimental effects of information loss and thus the choice of total cost function is very crucial. We have examined all three classical Pythagorean means – the arithmetic mean, the geometric mean, and the harmonic mean – and their variations and many of them provide unsatisfactory results. For all experiments described in this paper, we have used the mean squared error (MSE) or the $L2$-norm to merge the respective costs of each desired output function in the multifunctional settings. This is extremely suitable for our case as it aggravates the impact of the outliers, forcing the algorithm to put priority to search for a result for the output nodes that have yet to have a functional output. They can be further optimized later after all output nodes found their appropriate functions, leading to a very dynamical training process.

Genetic Algorithm

Following (Such et al. 2017), we have purposefully used the simplest genetic algorithms (GA) in almost all our experiments – unless specified – to set a baseline for the performance of using gradient-free evolutionary algorithms. The idea of GA is that there is a population P with N number of

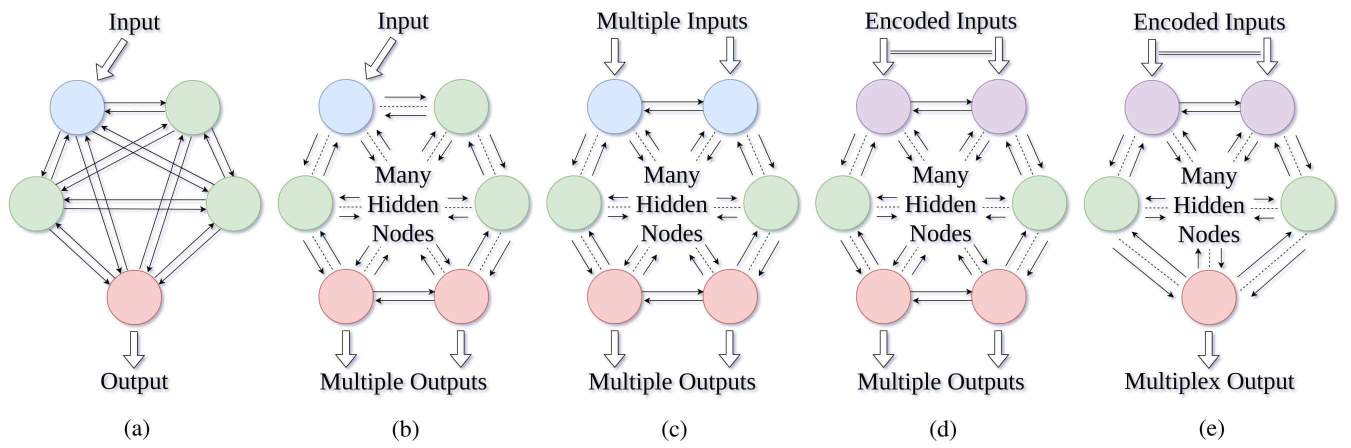


Figure 1: Representation of directed functional network where blue nodes are the visible input nodes, red nodes are the visible output nodes, green nodes are the hidden nodes, and purple nodes are the binary encoded input nodes. 1a shows a representation of a standard functional network with 5 nodes. It has only an input node an output node. 1b shows another representation with a single input and multiple output nodes. The dotted lines imply there are may be more hidden green nodes not drawn in the figure. 1c shows another representation with a multiple input and output nodes. 1d shows a another representation with multiple binary encoded input nodes with multiple output nodes. Finally, 1e shows another representation with multiple binary encoded inputs and a single multiplex output node.

individuals. For each generation G, the individuals are evaluated for their fitness score f according to a fitness function F. Those with the higher fitness score have a higher probability to survive while those with the lower fitness score would probably not make it to the next generation, and this is called the selection process. Finally, those that are determined to survive can "mate" or crossover and randomly mutated within the survivors group and repopulate the population. To simplify we do not use crossover process for our algorithm as crossover can be undesirable and negatively affect the performance of GA since effective structures identified by the GA can be broken by crossover (Stanley and Miikkulainen 2002). The GA algorithm is repeated until a stopping criteria is met – either because the GA has successfully converged (f ¡ certain convergence criteria C) and reached a solution or if it has failed and reached the maximum allowed number of generations.

Evaluation: Except for the first ever generation (G_0) in which the whole population P is randomly generated with function, evaluation is being done for the newly repopulated and mutated population of each generation ($G_{g>0}$) against our desired function. Concretely, if we determine B as the batch size of the function, we generate B number of points in which each point is the output level at an input level point. Then the fitness function F simply calculate the cost, that is the $L1$ or $L2$-norm between the output level points of the currently simulated function and the desired function. F outputs fitness score f, and our objective is to minimize it.

Selection: For all the experiments in this paper – unless specified – we have used a simple truncate-and-clone with elitism method. That is, if we have N number of individuals in a population and we set a truncation number as T. Then, the top $\frac{N}{T}$ number of individuals with the highest fitness score of the generation are deemed to survive, while the $\frac{(T-1)N}{T}$ rest of the population with the lower fitness score are removed from the algorithm. The top $\frac{N}{T}$ individuals are then cloned $\frac{(T-1)N}{T}$ times to repopulate the population.

Mutation: After repopulation, we now again have N number of individuals in the population and T duplicates of $\frac{N}{T}$ individuals. Let S_0 be the original survivors and S_t for each $\frac{N}{T}$ group where $t = 0..T$. For each S_t group, each weight of the individuals will be mutated according to $t\delta$ of the current θ strength degree: $\theta'_t = \psi(\theta, \delta, t) = \theta \pm t\delta\theta$ whereby δ is simply a small constant determining the mutation rate. With one exception, the true elite – the original best performing individual – will not be mutated. In other words, the S_t group with higher value of t will have greater mutation rate to sparse out the search.

Note that to calculate the fitness or cost between current function and target function, we first solve the ODE to obtain the simulated function.

Random Search In addition to GA described above, we also turned off the selection for GA, making it essentially a random search (RS) algorithm to check against our GA. We compared GA and RS to the previous work of gradient descent-based motif generation method.

Controllability

To formally validate the effectiveness of the generated large-scale networks and multifunctional networks with respect to their controllability – that is to discover the input settings where the network can be driven from any initial state to any desired final state within finite time. We used two widely used schemes such as inspecting the degree distribution of a

Algorithm 1 Network Genetic Algorithm

```
Input: max generations $G$, points batch size $B$, population size $N$, truncation number $T$, fitness function $F$, mutation rate $\delta$, convergence criteria $C$, selection function $\Upsilon$, mutation function $\Psi$, random initialization function $\Re$.
$\Re(P)$;                        // Randomly initialize $N$ networks
for $g = 1, 2, ..., G$ generations do
    for $n = 1, 2, ..., N$ individuals do
        $f_n = F(P_n, B)$;       // Simulate ODE and compute $f$
    end for
    if $f_{min} < C$ then
        return $P_0$ (Elite)     // Convergence successful
    else
        $P = \Upsilon(P, f, T, B)$   // See Selection above
        $P = \Psi(P, \delta)$        // See Mutation above
    end if
end for
```

whole linear – or linearized nonlinear – time-invariant complex network. (Liu, Slotine, and Barabási 2011), referred to as a Linear Nodal Dynamics Method. The technique captures the dynamical process occurring on edges of a directed complex network. (Nepusz and Vicsek 2012), referred to as the Switchboard Dynamics Method.

Transfer Learning

We used the required number of GA generations as a metric for the efficiency of the learning. Transfer learning occurs then if the number of generations to train a steady-state input-output function is lower if the training is initiated from another steady-state function as compared to being initiated from tabula rasa. That is, our hypothesis is that if a network is trained to output A from blank state, it may already have some structural properties facilitating the learning of another function B. We allow the intermediary A to be a triangle-like "peak" function (See Figure 2b) and B to be the band-pass function. We first generate 200 networks to output B from tabula rasa and record the required generations. Then, we will generate another 200 networks to output A, and then to output B, and record the required generations.

RESULTS

Computation of steady-state input-output functions with large networks

Recent gradient descent-based methods have low probability in generating large-scale networks ($N_{max} = 18$) (Yang et al. 2018). However, we were able to generate networks with 500, 1000, 2000, 3000 nodes, each network targeting a band-pass function. For the 3000 nodes, see Figure 2a. It was somewhat unexpected that it was feasible to find large networks which could robustly compute a defined input-output function despite the large degree of freedom within the network. All these networks was generated on a 56-cores desktop computer in less than 4 hours. We remark, that these results are generated with the simplest evolutionary algorithm. The node count can arguably be increased indefinitely and appears only to be limited by the memory. We are currently exploring what could be achieved using modern evolutionary algorithms (Lehman and Stanley 2011; Mouret and Clune 2015; Cully and Demiris 2017) and data storing techniques (Stanley, D'Ambrosio, and Gauci 2009; Ida et al. 2014; Iwashita et al. 2017). Such distributed computing can most likely increase the size of the networks to be explored by an additional couple of magnitudes.

Multiplexing of different input-output functions using the same network

Next we ask whether we can evolve multi-functional networks which can compute more than one single input-output function. Moreover, we investigated 4 particular input-output configurations as described in the methods section. For these experiments, the 3 output functions we tested were the band-pass (French Flag), valley (reversed French Flag), and threshold (Binary Step). We have arbitrarily chosen these functions and we emphasize, any desired function can be approximated with enough nodes. We have particularly chosen the French Flag because of its significance and difficulty to generate compared with easier functions like Linear.

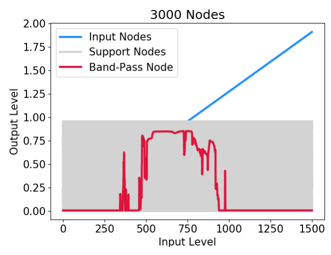
Multiple input-outputs scenario Figure 2c illustrate the case where a single input can control 3 different outputs while figure 2d shows 3 inputs controlling 3 different outputs. Here the GA learned the three different desired functions respectively. We have also looked deeper into the latter 3-in-3-out configuration as it is similar to the former 1-in-3-out configuration. After inspection of the generated network structure, we confirmed our suspicion that the other two input nodes were dummies and only one input node was lifting all the heavy work. For more details, please refer to Supplementary Information Figure 4.

Encoded inputs and multiplexing scenario For the following two experiments, we have essentially "encoded" the input nodes in binary format. That is, if both input nodes are not being injected with stimuli, it will be "00" which the least significant number is on the right side. If only the first input node is active, then it should be "01"; if only the second node is active, then it is "10"; and finally if both input nodes are being pumped with injections, then it is "11".

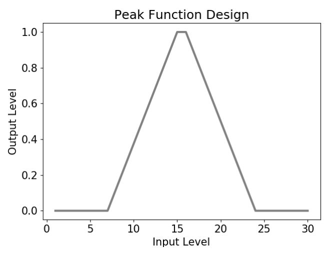
Figure 2f and 2e shows the "01" encoding generating a band-pass function for the first output node; Figure 2g shows the "10" encoding generating a valley function for the second output node; Figure 2h shows the "11" encoding generating a threshold function for the third output node.

And finally, the last experiment uses the same binary encodings but to output the 3 functions on a single multiplex output node. Figure 2j, figure 2j, and figure 2j shows the "01", "10", and "11" encodings successfully generated three functions respectively on the same output node with intrinsic multi-functional property.

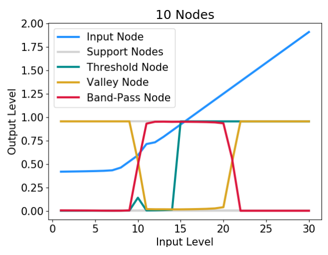
Multiplexing with more nodes The number of nodes are chosen arbitrarily for the 4 configurations described above, we have also tested out with 1000 nodes for all 4 configurations and their analysis results are included in Table 1.



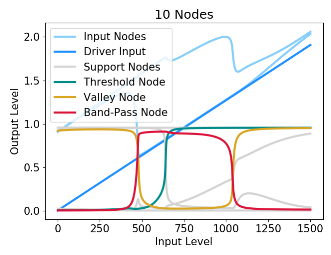
(a) High Nodes Band-Pass



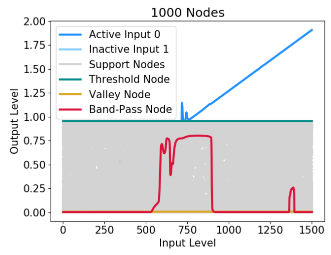
(b) Peak Function



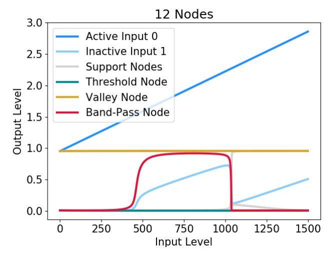
(c) 1-in-3-out



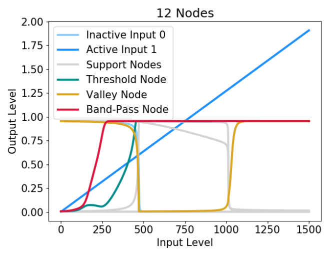
(d) 3-in-3-out



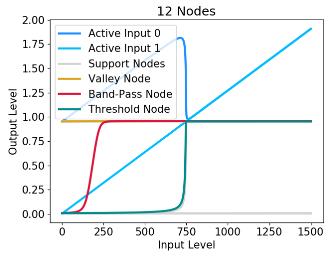
(e) 2-en-in-3-out-bp-1000 (01)



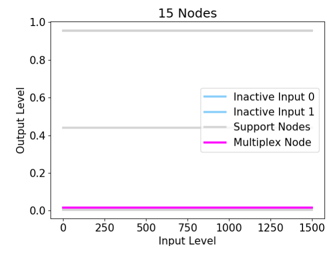
(f) 2-en-in-3-out-bandpass (01)



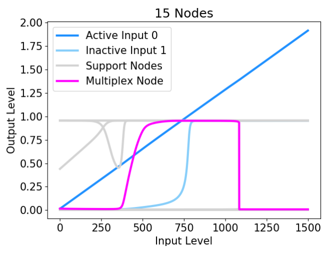
(g) 2-en-in-3-out-valley (10)



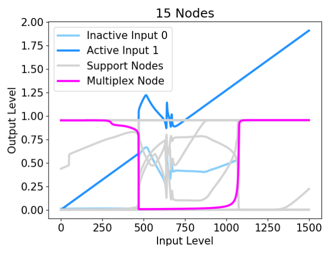
(h) 2-en-in-3-out-threshold (11)



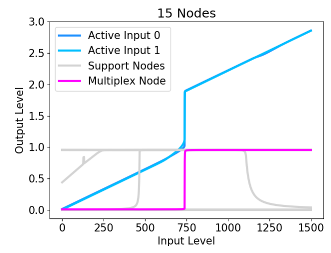
(i) 2-en-in-1-out-inactive (00)



(j) 2-en-in-1-out-bandpass (01)



(k) 2-en-in-1-out-valley (10)



(l) 2-en-in-1-out-threshold (11)

Figure 2: Graphs representing the simulated activity levels within the networks plotted as the output level versus input level. 2a shows a 3000 nodes network outputting the band-pass function. 2c and 2d shows the simulated expressions of multiple inputs and outputs. 2f, 2g, and 2h show the simulated activity levels of 3 different outputs – band-pass, valley, and threshold – produced by 3 different binary encodings respectively. 2e is the (01) encoding, 1000 nodes version of the same configuration. Similarly, 2i, 2j, 2k, and 2l show the 4 simulated activity levels produced by 4 different binary encodings but on a single multiplex output node. Figure 2b shows the peak function design used in the transfer learning experiment.

Random Search

To provide a reference to the GA search, we compared the results using a Random Search (RS) to search for any solutions which the networks can generate an useful function. Interestingly, RS fails to converge to generate any function in all sizes from 3 nodes up to 100 nodes. We have tried using RS to search for networks that can generate other, possible more tractable functions such as a sigmoid but was to no avail. And since gradient-descent based method fails at high node count, our Network GA serves as the new baseline for all future work.

The learned networks are controllable

Here as ask if these complex large networks are amenable to an engineered control or not. To this end we used the framework of the established controllability analysis. This was applied to the generated networks using what is referred to as the Linear Nodal Dynamics Method and Switchboard Method to compute controllability, as described in the Methods Section. Table 1 summarizes all the results from the controllability analysis experiment. The number of required driver nodes using Linear Nodal Dynamics Method turned out to always be 1. In contrast, using the Switchboard Dynamics Method, revealed an increased number of required driver nodes as the node count increases. Approximately less than half of the nodes were needed in order to control the network. The average number of required driver nodes in both cases as the network evolves, however, were approximately constant in both cases.

Transfer Learning between different steady-state input-output functions

Finally, we conducted an experiment to explore whether the required GA generations for the networks to train to output a steady-state input-output function will be lower if done from a trained state instead of from tabula rasa. We first compute the required generations to achieve the band-pass function from tabula rasa, band-pass from an intermediate ”peak” state (Figure 2b), and also peak from tabula rasa.

As GA generations can differ wildly between different epochs we have averaged them out over 200 networks for

Table 1: Results from the Controllability Analysis Experiment. *GD* (Gradient Descent) and *GD-Pruned* rows show the analysis done on networks generated using gradient descent – with and without pruning (Hiscock 2017). Note that the maximum node count available for *GD* is 18 (Yang et al. 2018). The following results originates from our GA method. $Edges_{max}$ show the total combinations available or possible for the nodes to connect to each other and themselves, which are naturally the square of N *Nodes*. The number of *Nodes* are also the number of *Vertices* which is not shown here. E shows the number of "edges" or useful connections in the network that are dense and not close to zero (We set "close to zero" as value between -1 and 1). e shows the number of edges in relative fraction, which is E over *Combinations*. $\overline{e}$ shows the average of e throughout the evolving process. N_L is the number of driver nodes detected according to the Linear Nodal Dynamics Method while N_S is the number of driver nodes detected according to the Switchboard Dynamics Method. n_L and n_S are the relative fractions of the number of driver nodes over total number of nodes of both methods respectively. $\overline{n_L}$ and $\overline{n_S}$ show the average of n_L and n_S throughout the evolving process respectively. Analysis result done on networks generated by previous gradient descent-based work has no average of described relative fractions throughout evolving process as they are only done on the final generated networks.

Experiment	**Nodes**	**Edges**$_{max}$	E	e	$\overline{e}$	N_L	n_L	$\overline{n_L}$	N_S	n_S	$\overline{n_S}$
GD	3	9	9	1	-	1	0.333	-	1	0.333	-
GD	10	100	95	0.95	-	1	0.1	-	2	0.2	-
GD	18	324	311	0.96	-	1	0.055	-	3	0.167	-
GD-Pruned	3	9	7	0.78	-	1	0.333	-	1	0.333	-
GD-Pruned	10	100	32	0.32	-	4	0.4	-	4	0.4	-
GD-Pruned	18	324	17	0.052	-	13	0.722	-	5	0.278	-
GA 1-in-1-out	3	9	9	1	1	1	0.333	0.333	1	0.333	0.333
GA 1-in-1-out	10	100	88	0.902	0.87	1	0.1	0.1	3	0.3	0.3
GA 1-in-1-out	100	10,000	9,140	0.914	0.917	1	0.01	0.01	46	0.46	0.43
GA 1-in-1-out	200	40,000	36,432	0.911	0.915	1	0.005	0.005	89	0.445	0.444
GA 1-in-1-out	500	250,000	229,439	0.918	0.921	1	0.002	0.002	239	0.478	0.476
GA 1-in-1-out	1000	1,000,000	916,884	0.917	0.92	1	0.001	0.001	481	0.481	0.488
GA 1-in-1-out	2000	4,000,000	3,733,207	0.933	0.932	1	0.0005	0.0005	989	0.495	0.491
GA 1-in-1-out	3000	9,000,000	8,400,692	0.933	0.933	1	0.0003	0.0003	1441	0.48	0.486
GA 1-in-3-out	10	100	99	0.99	0.99	1	0.1	0.1	1	0.1	0.1
GA 3-in-3-out	10	100	97	0.97	0.975	1	0.1	0.1	1	0.1	0.1
GA 2-en-in-3-out	12	144	142	0.98	0.983	1	0.083	0.083	2	0.167	0.167
GA 2-en-in-1-out	15	225	218	0.969	0.97	1	0.067	0.067	3	0.2	0.2
GA 1-in-3-out	1000	1,000,000	926,372	0.926	0.929	1	0.1	0.1	474	0.474	0.476
GA 3-in-3-out	1000	1,000,000	925,403	0.925	0.927	1	0.1	0.1	473	0.473	0.472
GA 2-en-in-3-out	1000	1,000,000	917,637	0.918	0.919	1	0.001	0.001	480	0.48	0.485
GA 2-en-in-1-out	1000	1,000,000	930,888	0.931	0.931	1	0.001	0.001	475	0.475	0.48

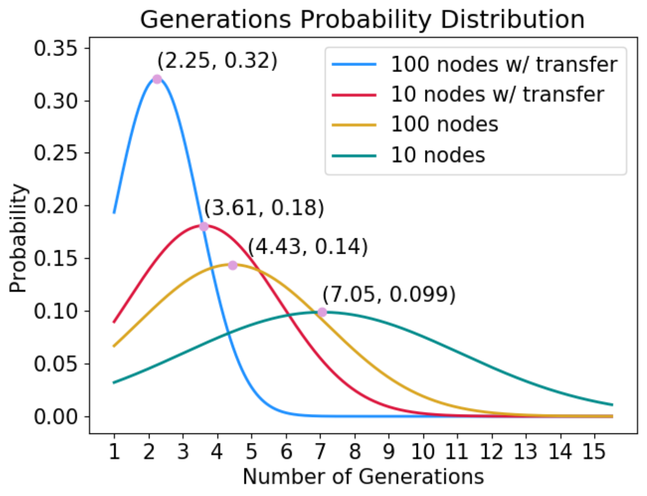


Figure 3: Figure shows the probability distribution of the number of generations required for the genetic algorithm to learn to output the band-pass function.

each configuration. Figure 3 shows the probability distribution of the number of generations required for the genetic algorithm to learn to output the band-pass function. For simplicity, the results from two separate processes – blank to peak, peak to band-pass – are merged into one. Interestingly, the generations needed to reach band-pass from peak is lower than to train a band-pass from tabula rasa (not shown in figure), inferring transfer learning. Even more interestingly, total generations needed to reach band-pass from tabula rasa if peak is also lower than to train a band-pass straight from tabula rasa. Also interestingly, it is "faster" in higher node count.

Discussion

The work presented here is clearly motivated by a perspective of how computing could occur in living systems. Our approach hinges on the notion to encapsulate a fine structure, i.e. network structure and dynamics, which in turns performs well defined input-output transformations of external signals via the encapsulation. Using evolutionary algorithms, we find here that such systems can readily evolve and be learned. Our embedded computational network architecture can be interpreted either at the levels of cells (internal network of molecules such as genes, proteins, metabolites) or alternatively as group or module of cells effective performing input-output transformation of external signals relative to the module. Naturally, this does not exclude a nested architecture ranging from cells encapsulating molecular networks to interacting modules composed of cells.

Historically, there have been numerous studies investigating internally stable states in networks of interacting elements equipped with some dynamics. This includes pure spin glass models (Castellana and Bialek 2014) or those inspired by such models, i.e. attractor networks (e.g. Hopfield)(Hopfield 1984), networks of McCulloch-Pitts neurons (McCulloch and Pitts 1943), or Boolean networks representing either molecular circuits (Kauffman 1969) or interacting cells. This important line of work essentially frames the problem such that; given a network architecture (e.g. symmetric connectivity matrix, statistical connectivity constraints) and rules for local computation (e.g. Boolean rules, threshold dynamics, summation of inputs) then what are the stable states, transitions, and dynamics within the system. Deep neural networks have successfully been trained as efficient statistical classifiers and we know that even shallow neural networks are universal in a Turing sense.

Now, in contrast our work flips the problem formulation around in the sense that we ask how could a system of interacting components realize a given computation defined as an input-output transformation. This is similar in spirit in addressing how to design a transistor or a chip to implement in some medium a certain signal transformation. What then are the advantages and limitations with our approach? We acknowledge our ignorance of specific details of molecular operations and architectures in different biological systems including neural architectures. Instead, using a global constraint, such as a computation of an input-output transformation we open up the possibility to assess what design principles, if any, are necessary or sufficient to realize such computations. In the last decade, there has been an increasing amount of data and interest in understanding biological networks of genes, proteins, and metabolites. In this vein, there has been a surge of studies asking how computations such as adaptation, detection of fold changes, and French-flag (i.e. bandpass) can occur in biological systems(Ma et al. 2009)(Adler et al. 2017). However, as rule, this has been investigated using strong constraints on the specific form of dynamics and exhaustive search in small 3-node circuits. Here we introduce the notion to use evolutionary search algorithms in order access large (thousands of nodes) networks. Our findings demonstrate the existence and fast convergence to these large systems instantiating the steady-state computations we investigate in this paper. Interestingly, in contrast to gradient descent techniques where we find a limit in terms of one-order of magnitude networks ($N < 20$)(Hiscock 2017), our vanilla GA readily exploits the search space. Thus, we can therefore address a computational analogue to the large networks we observe inside cells or between large number of cells.

We like to remark that our approach may appear akin to reservoir computing (Lukoševičius, Jaeger, and Schrauwen 2012) but there are important differences. First, here we do not employ gradient descent training of the local (reservoir) network, since the evolutionary search benefits from explorative search beyond local minima. Secondly, we train the system to learn steady-state functions, whereas liquid computing or reservoir computing exploits transients, i.e. the temporal dynamics in the system, without necessarily requiring stable states. Our motivation is that steady-state functions provides a benchmark of what can be computed and may constitute a fundamental computational operation implemented in living systems. Yet, the analogy between steady-state and temporal computing will be further investigated in future studies where we will extend our analysis to transient temporal signals.

Furthermore, in our work we demonstrate for the first time, to the best of our knowledge, multiplexing in such large network architectures. Hence, a network can be evolved to learn several different input-output transformations. Moreover, these can be configured in different manners, such as allowing for example a binary input coding projecting to either one or several different output channels.

One possible disadvantage with such large systems, compared to smaller computational network modules would be that the issue of control in an engineering and computational sense could become insurmountable. However, when computing controllability for our different circuits we observe that are indeed controllable in a technical sense. We hypothesize that this is a consequence of their fairly dense wiring, but this is a topic we are currently investigating using our data.

Finally, one advantageous feature for systems capable of learning would be that when the system is trained for one task, the system is thereby positioned to learn other tasks faster. This was in part observed for the reinforcement learning networks that were trained on Atari games. Hence, for any learning system we would like that as a consequence of a given learning paradigm that nearby tasks, or nearby (Atari) games, that transfer learning comes out as natural feature. This was readily observed and quantified in our experiments in that convergence was on average much faster provided that the networked learned one nearby input-output transformation. We are currently investigating to what extent this enforce the network to forget or partially remember the previous input-output functions.

Conclusions and future outlook

By introducing evolutionary search to identify circuits with computational capabilities we open up for a systematic study of the architectural and dynamical constraints of the system which collectively computes a function. This framework can readily be used to examine multifunctional circuits where depending on the encoding of the inputs will compute different functions using the same underlying architecture. The dependencies between degree of network modularity, controllability, and multiplexing when computing different functions either in the steady-state or transient domains appears promising to investigate at this juncture.

References

Ackley, D. H.; Hinton, G. E.; and Sejnowski, T. J. 1985. A learning algorithm for boltzmann machines. *Cognitive science* 9(1):147–169.

Adler, M.; Szekely, P.; Mayo, A.; and Alon, U. 2017. Optimal regulatory circuit topologies for fold-change detection. *Cell systems* 4(2):171–181.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34(4):18–42.

Castellana, M., and Bialek, W. 2014. Inverse spin glass and related maximum entropy problems. *Physical review letters* 113(11):117204.

Cully, A., and Demiris, Y. 2017. Quality and diversity optimization: A unifying modular framework. *CoRR* abs/1708.09251.

Geard, N., and Wiles, J. 2005. A gene network model for developing cell lineages. *Artificial Life* 11(3):249–267.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

Hiscock, T. W. 2017. Adapting machine-learning algorithms to design gene circuits. *bioRxiv* 213587.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Holland, J. H. 1992. Genetic algorithms. *Scientific American* 267(1):66–72.

Hopfield, J. J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences* 81(10):3088–3092.

Ida, A.; Iwashita, T.; Mifune, T.; and Takahashi, Y. 2014. Parallel hierarchical matrices with adaptive cross approximation on symmetric multiprocessing clusters. *Journal of Information Processing* 22:642–650.

Iwashita, T.; Ida, A.; Mifune, T.; and Takahashi, Y. 2017. Software framework for parallel bem analyses with h-matrices using mpi and openmp. *Procedia Computer Science* 108:2200–2209.

Jaeger, J.; Surkova, S.; Blagov, M.; Janssens, H.; Kosman, D.; Kozlov, K. N.; Myasnikova, E.; Vanario-Alonso, C. E.; Samsonova, M.; Sharp, D. H.; et al. 2004. Dynamic control of positional information in the early drosophila embryo. *Nature* 430(6997):368.

Kauffman, S. A. 1969. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology* 22(3):437–467.

Kleene, S. C. 1971, 1952 (10th impression 1991). *Introduction to Metamathematics*. North-Holland Publishing Company (Wolters-Noordhoff Publishing).

Lehman, J., and Stanley, K. O. 2011. Evolving a diversity of virtual creatures through novelty search and local competition. In *GECCO*.

Liu, Y.-Y.; Slotine, J.-J.; and Barabási, A.-L. 2011. Controllability of complex networks. *Nature* 473(7346):167.

Lukoševičius, M.; Jaeger, H.; and Schrauwen, B. 2012. Reservoir computing trends. *KI-Künstliche Intelligenz* 26(4):365–371.

Ma, W.; Trusina, A.; El-Samad, H.; Lim, W. A.; and Tang, C. 2009. Defining network topologies that can achieve biochemical adaptation. *Cell* 138(4):760–773.

McCulloch, W. S., and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5(4):115–133.

Mjolsness, E.; Sharp, D. H.; and Reinitz, J. 1991. A connectionist model of development. *Journal of theoretical Biology* 152(4):429–453.

Mouret, J., and Clune, J. 2015. Illuminating search spaces by mapping elites. *CoRR* abs/1504.04909.

Nepusz, T., and Vicsek, T. 2012. Controlling edge dynamics in complex networks. *Nature Physics* 8:568–573.

Reinitz, J., and Sharp, D. H. 1995. Mechanism of eve stripe formation. *Mechanisms of development* 49(1-2):133–158.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61:85–117.

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10(2):99–127.

Stanley, K. O.; D'Ambrosio, D. B.; and Gauci, J. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life* 15(2):185–212.

Such, F. P.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K. O.; and Clune, J. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR* abs/1712.06567.

Vohradsky, J. 2001. Neural network model of gene expression. *the FASEB journal* 15(3):846–854.

Yang, C.; Ooi, R.; Hiscock, T.; Eguiluz, V.; and Tegner, J. 2018. Learning functions in large networks requires modularity and produces multi-agent dynamics. *arXiv preprint arXiv:1807.03001*.