Optimizing Structural Properties Of Neural Networks With Genetic Algorithms

T. Staudt, E. Schultheis*
University of Göttingen
(Dated: today)

In this article we examine how structural properties of neural networks, like the size or local synapse densities, affect their learning success for various tasks. To do so, we look at a rate based model for neural networks and apply the FORCE rule for the learning process. A sophisticated matching algorithm allows us to quantify the learning success of a network so that the fitness of structural characteristics, expressed by integer or floating-point parameters, can be evaluated. We then use evolutionary optimization methods on these parameters to show that (1) ring topologies are generally more successful than strictly random topologies, (2) *FAT LIST OF INCREDIBLE RESULTS THAT CHANGE THE UNIVERSE*

Keywords: genetic algorithms, evolutionary algorithms, FORCE learning, rate networks

INTRODUCTION

Looking at the field of artificial neuronal networks, one finds a great variety of different models describing both the dynamic of networks as well as their plasticities, i.e. their learning behavior. But what exactly makes artificial neuronal networks learn successfully? And why are certain types of networks and certain learning rules effective for some problems, but fail at solving others? Confronting and eventually answering these questions is critical when trying to gain a deep understanding of neural networks and artificial intelligence.

Using the above questions as a guideline, we decided to analyze what network characteristics make the FORCE algorithm, introduced in [1], especially suitable for learning periodic patterns. More precisely, we wanted to identify which properties of a network are important for its learning success, and how to choose their values for an optimal result. To this end, one option would be to take an initial network and an adequate periodic pattern to be learned, and optimize the network's internals for this very task. Then patterns may emerge that hint at how an optimal network design looks.

However, we took a different approach: Instead of fine tuning single internal weights, we were concerned with the general rules of the network's structure. So we wanted our networks to be random to a certain degree, which seems biologically plausible, but we also wanted them to follow certain patterns and allow for specialization: How dense are the synapses distributed? Is there a strong compartmentalization? Are recurrent structures frequent? How strong should the feedback signal be, and how strong should the synapses fire on average?

To address these issues we looked at parameterized random networks, or in the language of genetics, at different *network genotypes*. These genotypes contain parametric information about the networks (*phenotypes*) to be created. Different phenotypes with the same genotype may thus have strong quantitative differences, but they share structural qualities whose fitness for a preassigned range of tasks we wanted to analyze and improve.

Since calculating the fitness of a genotype required the evaluation of many different phenotypes and because of the resulting stochastic nature of the problems we faced, we decided to use genetic algorithms for the optimization process.

We first lay out the main aspects of our model in a rather high-level overview. More details on the actual implementation are then provided in the third section (or ultimately in the published source code / supplementary material). Finally, some of the obtained results are presented and discussed.

CONCEPTS AND MODEL

Network Model We chose the same rate based network architecture that is used in the original FORCE publication [1] (denoted as architecture A) and also in [2], where our notation mainly stems from. So we look at a network $\mathcal{N}$ with N internal neurons and states x_i for $1 \leq i \leq N$ that loosely represent the neurons' membrane potential. The firing rate r_i of the i-th internal neuron is given by $r_i = \tanh x_i$. Furthermore, the network contains R readout neurons with states z_i for $1 \leq i \leq R$, where

$$z_i = \sum_{j=1}^{N} \omega_{ij}^{\text{read}} \, r_j \tag{1}$$

with the readout weights ω^{read}. Taking into account both internal dynamics as well as an external feedback pathway, the dynamics $t \mapsto x_i(t)$ of the single neurons are governed by [3]

$$\dot{x}_i(t) = -x_i(t) + \sum_{i=1}^{N} \omega_{ij}^{\text{rec}} \, r_j(t) + \sum_{i=1}^{R} \omega_{ij}^{\text{fb}} \, z_j(t) \ , \tag{2}$$

where we introduce the internal recurrent synapse weights ω_{ij}^{rec} and the feedback synapse weights ω_{ij}^{fb}. One can see that the neurons exponentially decay if they are not presented stimulus by other neurons. We solved this system of differential equations by a simple Newton method with integration time steps $dt = 0.01$, which was also used in [1].

Learning Learning took place by stepwisely adapting the weights ω^{read} so that the dynamics $t \mapsto r_i(t)$ of the readout neurons should eventually match a given periodic target pattern $t \mapsto r_i'(t)$, called a *task* from now on, as accurately as possible. To update ω^{read} we chose a variant of the FORCE algorithm [1] that rapidly modifies the feedback loop to keep the errors $|r_i(t) - r_i'(t)|$ small from the beginning on.

While the authors of [1] update the weights λ^{read} in fixed intervals (every second time step), we used an adaptive mechanism to determine how often learning should occur. This was initially introduced for performance reasons, as the single learning steps proved to be the main computational bottleneck, but we also found that the long time network dynamics tended to be slightly more accurate for adaptive learning steps.

Network Genotypes Since we are not interested in single network properties but rather want to find out which structural elements of networks are important for learning success, we introduced *network genotypes.* Such a genotype G consists of a set Θ of parameters that determine which concrete networks are created when G is expressed in a single phenotype. Technically G may be understood as a stochastic network generator that returns a network $\mathcal{N} = G(\lambda)$ for every seed value λ. The genotypes' parameters $\theta \in \Theta$ were either integer or floating point values and can be classified as affecting either (1) the topology of ω^{rec} or (2) the weight values of ω^{rec} and ω^{fb}. Important examples for (1) are the networks size N, the occupation probability p (for random Erdös-Renyi networks), the neighborhood range k (for ring topologies), or the number l of layers (for feed-forward networks). We furthermore introduced ratio parameters that allowed us to interpolate between different topologies. Optimized parameters of type (2) were the feedback strength `feedback` (with $\omega^{\text{fb}} \propto$ `feedback`) and the gain value `gain` (with $\omega^{\text{rec}} \propto$ `gain`).

Fitness The next step was to define the *fitness* of a given genotype when confronted with a diverse range of tasks, like waves of different frequencies. To express this formally, we introduced *challenges*: For a given seed value λ, the challenge C yields a task $r' = C(\lambda)$ according to some (parametrized) distribution. If we now make use of a success function $S(\mathcal{N}, r') \in \{0, 1\}$ that decides, if the network $\mathcal{N}$ has successfully reproduced the desired output r', the fitness $F_C(G)$ of a genotype G for a challenge C may reasonably be defined as the expectation value of $\lambda \mapsto S(G(\lambda), C(\lambda))$. For fixed challenges, the main task of optimizing the genotype is now reduced to finding parameters Θ that maximize F_C.

METHODS AND IMPLEMENTATION

Tasks For the optimization to work properly, the tasks presented to the networks must be reasonably chosen; they must be learnable by FORCE (at least in principle) and must be compatible with technical aspects like the integration time step. While it was shown in [1] that large recurrent neural networks are capable of learning very complex tasks, the optimization requires an extensive number of learning processes, since every single evaluation of the fitness F_C needs dozens of tested phenotypes (see below). Therefore, the networks had to be restricted to being quite small (≈ 100 neurons) and are consequently less powerful.

To get a first estimate whether it might be possible to learn a given task using FORCE with 100 neurons in our network model, we looked at a single sinusoidal of frequency ω and determined the frequency range in which an (unoptimized) network (i.e. `gain`, `feedback`, p) is able to learn the function (see fig. **??**).

Since we expect/hope that the optimized network extends this range of possible frequencies, we choose the frequencies for the optimization tasks from the slightly larger interval $[0.6e-2, 4]$ such that their logarithms are uniformly spaced. This allows for good resolution for both high and low frequencies.

Success As mentioned in *Concepts and Methods*, we need a function S that decides weather a network succeeded or failed to solve a task. To this end a measure $f(r', r)$ for the concordance of the target function $r'(t)$ and the network readout $r(t)$ is sought. Though one could use a simple difference based metric, like the norm $f_2(r, r') = \|r - r'\|_2$, this has the drawback that small phase mismatches could produce high differences, even though the shape of r matches the one of r' well.

Instead, a measure that is time-shift independent and maximized for functions of identical shape is the maximum of the cross correlation

$$f_{\text{cor}}(r, r') = \max_t \frac{\langle r, r'(\cdot - t)\rangle}{\sqrt{\langle r, r\rangle \langle r', r'\rangle}} \ . \tag{3}$$

Algorithmically, the signals r and r' are split into overlapping chunks which are correlated separately. This saves computational power and prevents minute differences in frequency to accumulate into a huge phase shift, so the phase only has to remain relatively constant over the timescale of a single chunk (which is about 10 seconds of simulated time). This measure becomes problematic for functions that are very close to zero for a longer time (order of chunk size), but since we are going to work with superpositions of sinusoidals of different frequencies, this drawback is unproblematic.

Using this measure of quality, the success function for a network $\mathcal{N}$ with readout r for the task r' was taken to be

$$S(\mathcal{N}, r') = \begin{cases} 1 & \text{if } f_{\text{cor}}(r, r') \geq 0.95, \\ 0 & \text{if } f_{\text{cor}}(r, r') < 0.95 \ . \end{cases} \tag{4}$$

The threshold value of 0.95 was determined empirically, as signals r with $f(r, r') \geq 0.95$ met our assessment of what we considered a good reproduction of the target pattern.

Genetic Optimization Due to the stochastic nature of a genotype G's fitness $F_C(G)$, a very high number of samples would be required to obtain sufficiently smooth approximations of $F_C(G)$ for typical optimization methods like gradient descend or simulated annealing. But as the computational cost for even one learning process was considerable high, these methods were not feasible.

An alternative approach would be to use techniques from Monte Carlo integration, e.g. stratified or importance sampling, to aid in the evaluation of $F_C(G)$, but since $G(\lambda)$ and $C(\lambda)$ depend non-continuously on the seed value λ, this too would not solve the problem.

Therefore, we optimized $F_C(G)$ using a genetic algorithm, which is inherently capable of coping with the stochastic fitness function. In each iteration of the algorithm, a generation $\mathcal{G}_t = \{G_i \mid 1 \leq i \leq N\}$ of N individuals (Generators) is converted into a new one, $\mathcal{G}_{t+1}$, according to the following scheme:

Evaluation: The fitness $F_i = F_C(G_i)$ of each individual is measured using at least 20 samples (and adaptively more if the $(F_i)_{1\leq i\leq N}$ are spaced closely, to at most 100).

Selection: All individuals are grouped into disjunct pairs for which we check which one has the higher fitness. This is repeated $k = 10$ times, counting the number of wins for each individual. Based thereon we use the $p = 50\%$ best individuals $\mathcal{G}_t^{\text{survive}}$ to build the next generation. By using pair comparisons instead of simple ordering according to the fitness, we allow weaker individuals to reach the next generation (albeit with low probability), keeping the gene pool more diverse.

Reproduction: We obtain the next generation $\mathcal{G}_{t+1}$ by first generating $2N$ candidates via *mutating* or *recombining* elements of $\mathcal{G}_t^{\text{survive}}$. This means we either change a single parameter randomly or take two individuals and randomly choose for each parameter from which parent to take the value. We then perform a single sample fitness test and remove all networks that fail, until only N networks are left; these form $\mathcal{G}_{t+1}$. By this approach the most unpromising elements of the new generation are removed before the expensive fitness evaluation takes place.

Choice of Parameters - $N = 100$ fixed, since greater N always increased quality and computational cost increased quadraticaly to N - Mutation rate

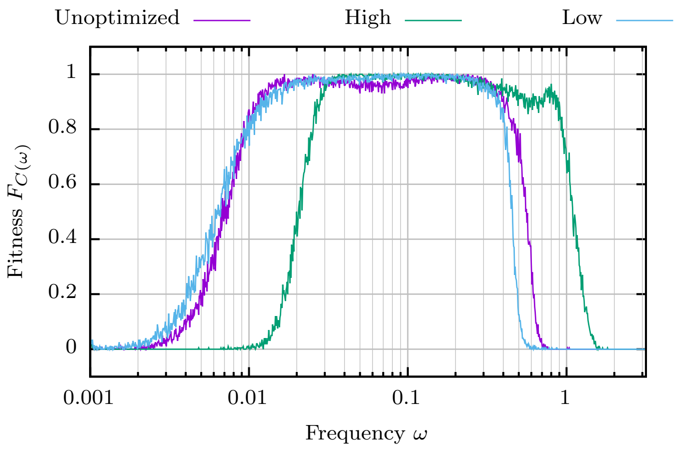

FIG. 1. CAPTION TEXT

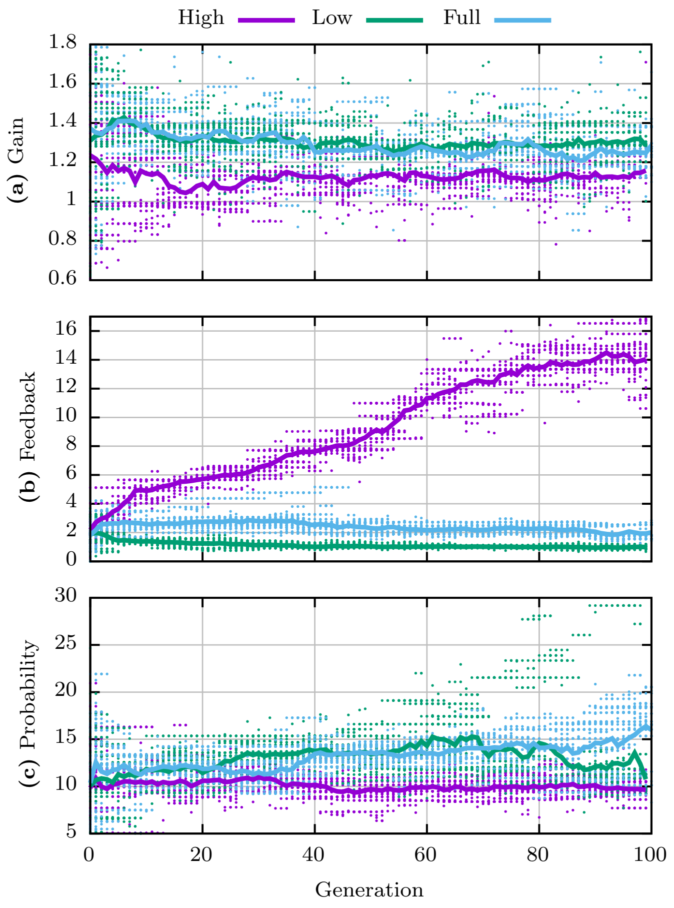

FIG. 2. CAPTION TEXT

RESULTS

Frequency Dependent Optimizations

- short description: get feeling for frequency range (by using unoptimized parameters from the force paper), make optimizations with 3 different challenges: 1) challenges that cover the frequency area completely, challenges that cover low frequencies, and challanges that cover high frequencies, then pick genotypes with high fitness.

Topology Optimizations

Baseline gain and feedback tests

(feedback shifts possible frequencies to higher values)

success of omega: erdös-renyi, ring optimizations: ER topology, Ring Topology, Mixing

DISCUSSION

- Remarks * Influence of initial conditions -¿ Degenerated initial population poses major problems (does not find same / takes very long to find same parameters as when started with diversity). -¿ With broad diversity: often fast convergence -¿ Gen. Optimizer good as optimizing filter, but has hard times to explore new parameter regions if no right genotypes are present (especially isolated maxima of F_C). Solution: Could introduce migrants, that supply the gene-pool with underrepresented genotypes. * According to experience: maybe not ideal for finding single set of parameters but rather for successful parameter-ranges (branches?)

- Problems * Many magic / “empiric” parameters scattered in the model that were not examined exhaustively * Only few and uncomprehensive number of test / experiments conducted due to the computational cost. * Test with several other topologies did not yield concrete results, only rough tendencies. * Influence of statistics: Learning success depends stronger on choice of challenges than parameters (partially) -¿ must be very careful how to interpret the results

* thomas.staudt@stud.uni-goettingen.de, erik.schultheis@uni-goettingen.de

[1] D. Sussillo and L. F. Abbott, Neuron **63**, 544 (2009).
[2] G. M. Hoerzer, R. Legenstein, and W. Maass, Cerebral Cortex **24**, 677 (2014).
[3] Note that we did not include external input signals in this formula, as is done in [1] and [2]. We in fact enabled inputs in our code base, but did not use them for the main results.