Paper Implementation: Reproduce some basic results of the article "Generating Coherent Patterns of Activity from Chaotic Neural Networks"

Hsieh Yu-Guan

June 18, 2017

Summary

In this report we'll be working on the article "Generating Coherent Patterns of Activity from Chaotic Neural Networks" published by David Sussillo and L.F. Abbott in 2009. In the work of D. Sussillo and L.F. Abbott, they were interested in neural networks that are capable of generating complex activity patterns either spontaneously or in response to different stimuli.

The network is meant to be chaotic before training, i.e. its activity should be highly sensitive to initial conditions. This is acheieved by the usage of strong recurrent synaptic connections inside it. The so-called FORCE learning algorithm is then employed to modify synaptic weights either external to or within the network. After the training phase, the network will be able to produce a wide variety of complex output patterns and we can switch between different outputs by controlling the input signal.

Now let's come back to this report itself. I only worked on a relatively small part of the article: the generation of periodic patterns in the absence inputs by using FORCE learning. Moreover, three distinct network structures are considered in the article (this will be detailed later on), but only the first one, which is probably also the simpest, is implemented here.

Introduction

When we're performing some motor action, what is the source of the nerual activity that initiates and carries out this behavior? In the paper of D. Sussillo and L.F. Abbott, they examined the hypothesis that such actions may come from the reorgnaization of spontaneous neural activity, which then leads us to another question: how can a such reorgnaization happen?

The objective is thus to find a possible synaptic weight modification rule that allows the network activity to be reorgnaized into coherent patterns required to produce complex motor actions after training. However, comparing with feedfoward architectures, the training of a recurrent network is shown to be much more difficult due to several reasons. First, feeding erroneous output back into a network during training may prevent its activity from converging. Secondly, in a recurrent network the credit assignement for output errors becomes quite a hard problem because of the presence of recurrent weights. Finally, while using a network that display chaotic activity prior to training can be beneficial, chaos must be avoided during the learning phase.

FORCE learning solved these problems by keeping the errors always small and forcing the synaptic modifications to be strong and rapid during traing. This is quite different from classic learning algorithms which usually reduce the errors little by little. Therefore, the goal of FORCE learning is not significant error reduction, but rather reducing the amount of modification needed to keep the errors small.

According to the article, the contribution of FORCE learning can be viewed from two different angles. From a biological point of view, it can be regarded as a model for learning-induced moficifation of biological models. Or more simply, it can just be seen as a powerful algorithm that is able to construct recurrent networks being able to generate complex and controlable output patterns.

Network Structures

The network that was studied in the paper and will be study here is composed of individual neurons characterized by their firing rates and sparsely interconnected through excitatory and inhibiroty synapses of various strengths (when it comes to the biological reality, we may need to seperate group of excitatory and inhibitory neurons, but when we're modeling neural circuits, such details become less important). Initially, the connectivity and synaptic strengths of the network are chosen randomly.

The next problem is the definition of the output of the networks. The technical choice is to define it as a linear combination of neuronal activities. Formally, if at time t, we assemble the firing rates of all the neurons into a column vector $\mathbf{r}(t)$, and the weights connecting these neurons to the output is described by the column vector $\mathbf{w}(t)$, then the network output is

$$z(t) = \mathbf{w}^\top \mathbf{r}(t).$$

Schematically, this output will be stored in a readout unit and its value may be fed back into the network (it depends on the concrete structure). Though thus far the network output is only a scalar, by using several differents decoding vectors $\mathbf{w}$ we can easily define a multidimensional readout. This is exactly what D. Sussillo and L.F. Abbott did in their paper when they later asked the network to generate multiple outputs; however this point will not covered by the report.

Now that we have defined z as the output of the network, the goal of the algorithm is to set $z = f$ for somme target function $f : t \mapsto f(t)$. Either z is generated in the abscence of any input or it may depend on inputs of the network in a specific way. In the scope of this report, only the first case will be discussed.

Finally, as shown in Figure 1, three different network structures are proposed in the article. In the first structure the feedback to the generator network is provided directly from the readout unit. During training, we modify only the decoding vector $\mathbf{w}$ and the internal recurrent synaptic strengths are left intact. Such a network is simple but far from realistic: every neuron receives the same feedback (though it may be weighted differently) and this feedback itself originates from an abstract output that doesn't exist in real circuit. To address this problem, two other structures more biologically plausible are given. In the second structure we use a separated feedback network to supply feedback and in the third one modifications on internal synaptic strengths are allowed. Without giving more details here, I just mention that after proper adaptions, FORCE learning can be carried out on the three structures and yield promising results indifferent to the underlying network structure.

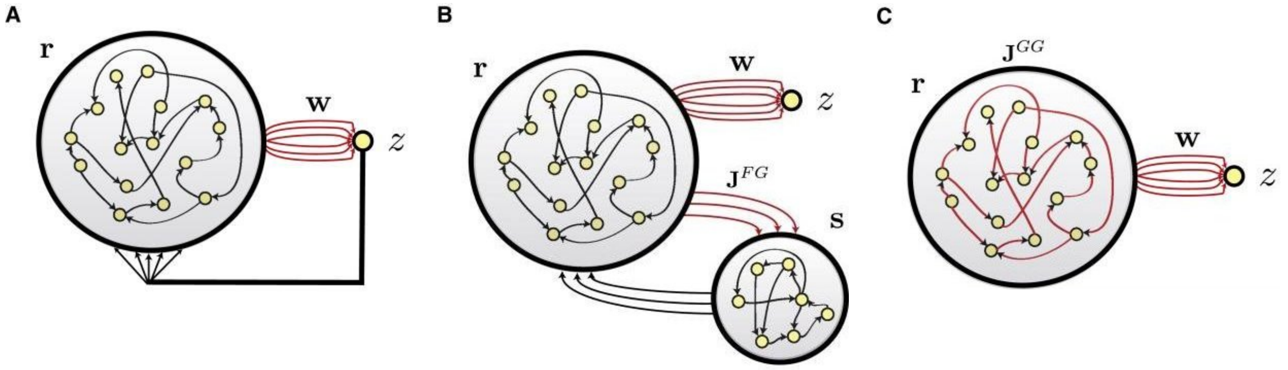

Figure 1: Network Architectures (picture from the paper).
In all three cases, a linear readout unit derives an output z from the firing rates $\mathbf{r}$ of the recurrent generator network. Only connections shown in red are subject to modification.
(A) The output of the readout unit is directly fed back to the generator network.
(B) Feedback to the generator network is provided by a separated feedback network recurrently connected and whose inputs come from the generator network through synapses of strength $\mathbf{J}^{FG}$ (red) that are also modified during training.
(C) A simple recurrent network with no external feedback, but all synaptic strengths $\mathbf{J}^{GG}$ are subject to be modified by the learning algorithm.

Now let's put all the elements together and give a formal description of the network that will be used in the framework of this report. First we have a column vector $\mathbf{x}$ containing some abstract values of each neuron (just for modeling convenience, or one may be able to find some biological meaning for this vector that I ignore here). The firing rates are computed by $\mathbf{r} = \tanh(\mathbf{x})$ (application of the function term by term). The dynamic of the network is governed by the equation

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + g_{GG}\mathbf{J}^{GG}\mathbf{r} + g_{Gz}\mathbf{J}^{Gz}z,$$

where τ is the caracteristic time, $\mathbf{J}^{GG}$ and $\mathbf{J}^{Gz}$ are respectively the matrix of internal synaptic weights and the vector of connection weights from the readout unit to the generator network (the feadback loop), and z is the output defined earlier. g_{GG} is introduced to scale the strengths of the recurrent connections. When $g < 1$ the network is inactive prior to training; on the contrary if $g > 1$ the network exhibits chaotic spontaneous activity. g_{Gz} plays a similar role with regard to the connections from the readout unit to the main network.

We use moreover a sparseness parameter ρ to impose sparsity on the network: each pairwise connection is set and held to 0 with probability $1 - \rho$. In the whole report, we'll choose $\tau = 10$ ms, $g_{GG} = 1.5$, $g_{Gz} = 1$, and the network contains $N = 1000$ neurons unless otherwise stated. For initialization, nonzero elements of $\mathbf{J}^{GG}$ are drawn independently from a Gaussian distribution with zero mean and variances $1/(\rho N)$ while elements of $\mathbf{J}^{Gz}$ are drawn from a uniform distribution between -1 and 1. We notice that feedback synapses are stronger than internal ones in order to have an appreciable effect on the network activity and this is necessary for suppressing the inherent chaos of the network. Similarly, elements of $\mathbf{w}$ and $\mathbf{x}$ are respectively generated by Gaussian distributions with zero means and standard deviations $\sqrt{1/(\rho N)}$ or $\sigma_x = 0.5$.

FORCE learning

The term FORCE learning stands for first-order reduced and controlled error learning. Training of recurrent networks is in general a hard problem as already mentioned in the introduction (even though here we modify only the readout vector $\mathbf{w}$, the presence of the feedback loop makes any effect of the modification difficult to predict). Since erroneous output shouldn't be fed back to the network, the proposed algorithm must rapidly reduce the magnitude of the difference between the actual and desired output to a small value, and then keep it small while searching for the proper fixed readout vector that can generate the target function without further modification of this vector.

The general schema of such an algorithm consists of updating $\mathbf{w}$ at times separated by an interval Δt based on the readout error at this moment which is defined by:

$$e_-(t) = \mathbf{w}^\top(t-\Delta t)\mathbf{r}(t) - f(t).$$

The minus subscript signifies that this is the error prior to weight update at time t. The update of $\mathbf{w}(t-\Delta t)$ to $\mathbf{w}(t)$ should allow us to decrease the error so that

$$e_+(t) = \mathbf{w}^\top(t)\mathbf{r}(t) - f(t)$$

satisfies $|e_+(t)| < |e_-(t)|$. Since the end of the training takes place only when the decoding vector $\mathbf{w}$ stabilizes and no longer changes, this also requires $e_+(t)/e_-(t) \to 1$. Note that Δt is the interval of time between modifications of readout weights and doesn't necessarily equal to the basic integration time step for the network simulation. In fact, for the simulations carried out throughout this report, I take $\Delta t = 1$ ms whereas the basic integration time step is 0.1 ms.

Knowing all of this, it's still far from obvious to deduce the appropriate modification rule that will work. In the paper, D. Sussillo and L.F. Abbott express their favor to use the recursive least-squares (RLS) algorithm here. I'll not dig into the mathematical details of this algorithm, but briefly in our case it gives

$$\mathbf{w}(t) = \mathbf{w}(t-\Delta t) - e_-(t)\mathbf{P}(t)\mathbf{r}(t),$$

where $\mathbf{P}(t)$ is an $N \times N$ matrix that is updated at the same time as the weights according to the rule

$$\mathbf{P}(t) = \mathbf{P}(t-\Delta t) - \frac{\mathbf{P}(t-\Delta t)\mathbf{r}(t)\mathbf{r}^\top(t)\mathbf{P}(t-\Delta t)}{1+\mathbf{r}^\top\mathbf{P}(t-\Delta t)\mathbf{r}(t)}.$$

We set the initial value for $\mathbf{P}$ to be $\mathbf{P}(0) = \mathbf{I}/\alpha$ where $\mathbf{I}$ is the identity matrix and α is a constant parameter that needs to be chosen carefully according to the target function being learnined. In this report I use $\alpha = 1$ as suggested by the article. The modification rule can then be viewed as a starndard delta-type rule, but instead of a scalar quantity, multiple learning rates are specified by $\mathbf{P}$, which is in reality a running estimate of the quantity $(\sum_t \mathbf{r}(t)\mathbf{r}^\top(t) + \alpha\mathbf{I})^{-1}$. The use of $\mathbf{P}$ allows us to have a finer control over learning because it adapts the learning rate to the magnitude of different principal components of the network activity. However, concerning how it works and why it works, it will no be explained here.

One can easily show that if we assume that $\mathbf{w}(0) = \mathbf{0}$ for simplicity, the error after the first weight update at time Δt is

$$e_+(\Delta t) = -\frac{\alpha f(\Delta t)}{\alpha + \mathbf{r}^\top(\Delta t)\mathbf{r}(\Delta t)}$$

which is small as long as $\alpha \ll N$ knowing that $\mathbf{r}^\top \mathbf{r}$ is of order N. Furthermore, every time when the weights are updated, we have

$$e_+(t) = e_-(t)(1 - \mathbf{r}^\top(t)\mathbf{P}(t)\mathbf{r}(t)).$$

The quantity $\mathbf{r}^\top \mathbf{P} \mathbf{r}$ is always positive and decreases to zero over the course of learning; this means the size of error is reduced by the weight update and as required $e_+(t)/e_-(t) \to 1$.

FORCE Learning Examples

In this section we'll apply the FORCE procedure on the network for different target functions. The network is able to learn to generate a variety of patterns after training. The first example presented here is a triangle wave of period 0.6 s (Figure 2). The network activity is chaotic before training (blue lines), and during the training phase, as predicted the error is reduced dramatically after a first weight update and the output of the network (red line) matches target function throughout the rest of training. The progression of learning can be observed through the decrease of size of fluctuations of the readout vector (orange line). $\mathbf{w}$ fluctuates rapidly at the beginning of training to force the network to produce the desired periodic pattern, but gradually it stabilizes and find the static weights that are needed to generate the target function without requiring modifications. After the training, the network continues to generate the desired pattern indefinitely without need of modifcation on decoding weights.

We also notice that the network activity chaotic prior to training becomes periodic in order to produce the periodic target function, and it continues to be periodic after learning. The learning process is indeed quite rapid here, only five cycles of the traingle wave is sufficient here (and in the paper the convergens takes place only after four cycles). This time depends however a lot on the form of the target function. For example, to learn

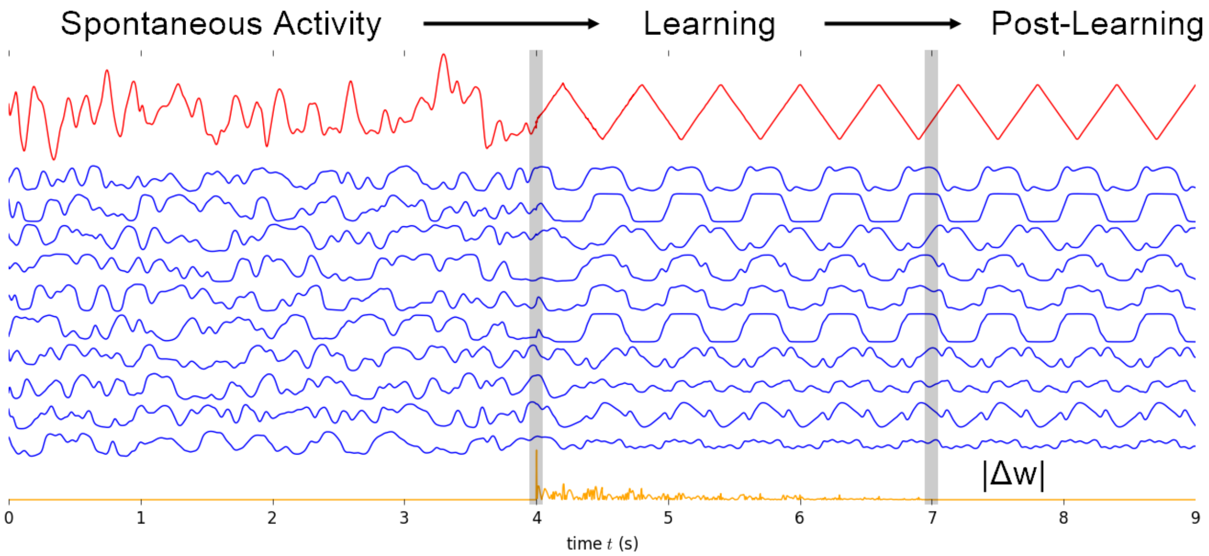

Figure 2: FORCE Learning Applied to a Triangle Wave.
Network output z is in red, the firing rates of 10 sample neurons are in blue and the orange trace represents the magnitude of fluctuations of $\mathbf{w}$ ($\Delta\mathbf{w} = \mathbf{w}(t) = \mathbf{w}(t - \Delta t)$).
The network exhibits chaotic spontaneous activity before training. During training, the readout error is reduced to small magnitude rapidly and then the output matches always the target function. Readout weights fluctuate greatly at first, gradually the fluctuations diminish and finally almost no more modifications of $\mathbf{w}$ are needed. At this moment we can turn off the learning procedure and the network persists in generating the desired triangle wave indefinitely without requiring any weight modifcation.