Recurrent Networks with FORCE Learning

Amrut Nadgir

October 2020

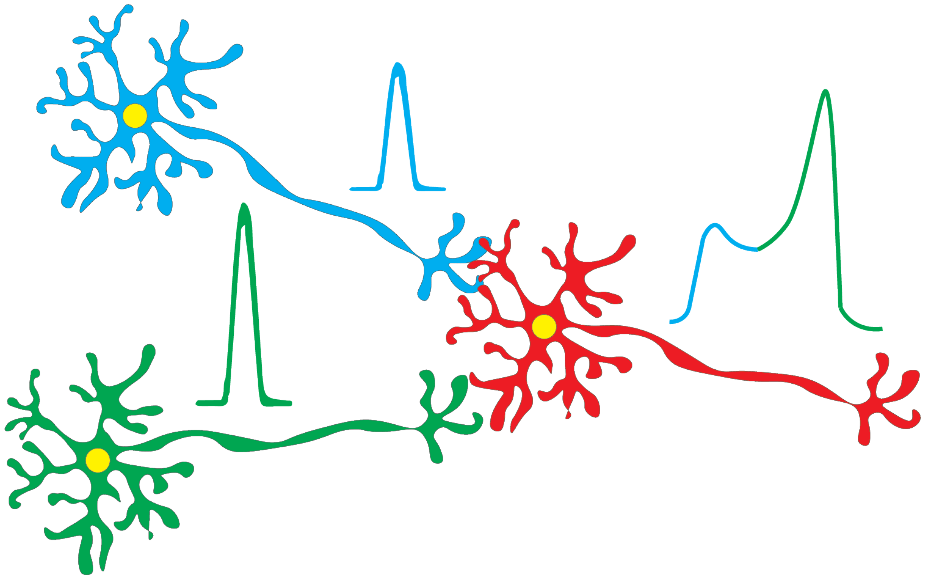

Figure 1: Neuron Art

1 Background

1.1 Network Structure

This recurrent network model is based on the one used by Sussillo and Abbott in their FORCE learning paper.

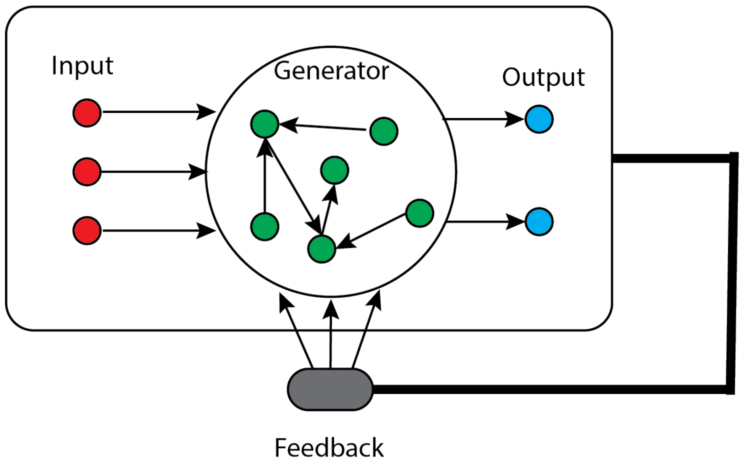

Figure 2: Diagram of the different RNN components. Input units send signal to all generator nodes which send signal to output. Feedback is a black box function of the previous network activity.

As seen above, each network can have a set of inputs, outputs, internal generator nodes and feedback.

1.2 Descriptive Equations

We will simulating a basic integrate and fire model of neural activity governed by the following differential equation.

$$\tau \frac{dx_i^G}{dt} = -x_i^G + \sum_{j \in E^G} w_{ji}^G A_j^G + \sum_{k \in E^I} w_{ki}^I A_k^I + \sum_{m \in E^F} w_{mi}^F A_m^F$$

The superscripts G, F, and I indicate the nodes/weights within the generator, feedback and input sets respectively. $A_i^G = \tanh(x_i)$ is the activation of node i in the generator network and can represent the firing rate of a neuron.A_k^I and A_m^F are values of the inputs and the feedback respectively. In this model, τ is a proportionality constant, E is the set of nodes presynaptic to node i and w_{ji} is the weight of the edge from j to i.

Each output of this network is evaluated as,

$$A_n^O = \sum_{j \in G} w_{jn}^O A_j^G$$

Where $A_n^O(t)$ is the output at t, G is the set of all nodes in the generator network and the superscript O indicates output nodes/weights. Essentially, the output is a linear combination of the activity of all the input nodes.

1.3 Discretization

When implementing this in code, the equation was discritized and evaluated in timesteps by using the following model-

$$x_i^G(t) = r_i x_i^G(t-1) + \sum_{j\in E^G} w_{ji}^G A_j^G(t-1) + \sum_{k\in E^I} w_{ki}^I A_k^I(t-1) + \sum_{m\in E^F} w_{mi}^F A_m^F(t-1)$$

t parametrizes the timestep of the equation and r_i is the preset recurrent weight for each internal neuron.

1.4 FORCE Learning

These types of recurrent neural networks can generate spontaneous chaotic activity which can be trained to match various patterns using FORCE learning. FORCE learning is a training paradigm that begins by rapidly changing weights so the network output closely matches the desired function. It converges when large rapid weight changes are no longer necessary for minimal error in output. To implement FORCE learning, we used the Recursive Least Squared algorithm as detailed in Sussillo and Abbot's paper.

Often, only certain subsets of the weights are trained so the algorithm has a better chance of converging.

2 Results

For both results, we used a 1000 node and about 12000 edge generator network with random connections between the nodes. We had no inputs and one output which we trained to match our desired function. In addition, we used the function, $1.3\tanh(\sin(\pi z(t-10)))$ where $z(t-10)$ is the network output delayed by 10 timesteps.

FORCE training was successfully able to train both networks to produce the desired output. However, the convergence and success of the training was heavily dependent on the initial chaotic activity, weights and learning rate.

2.1 Sawtooth

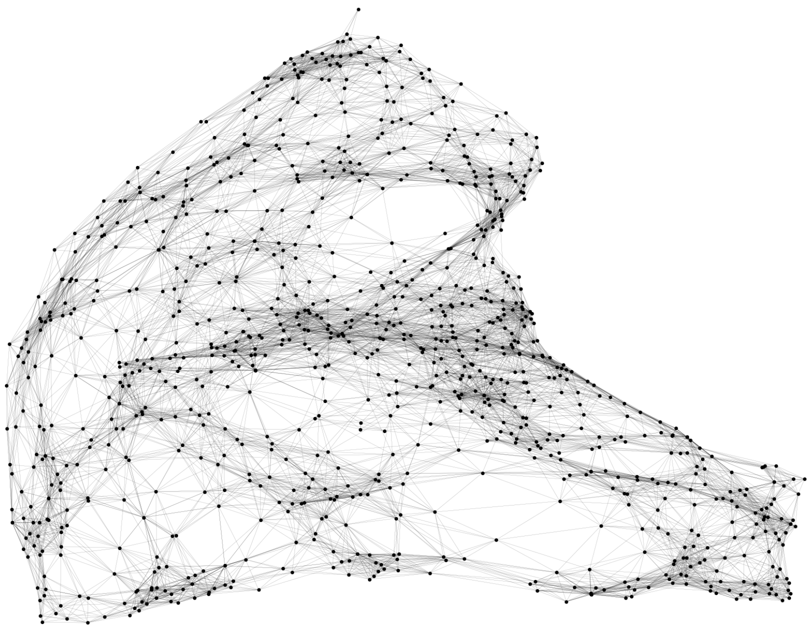

Figure 3: A visualization of the network structure we trained on the sawtooth wave. It is aesthetically pleasing.

We observed chaotic activity from the network and wanted to train it to fit a periodic waveform. Our initial test was to train the network for 1000 timesteps on a sawtooth time series which oscillated between -1 and 1. For this test, we only trained the output weights and held all other weights constant. This ensured quick convergence.

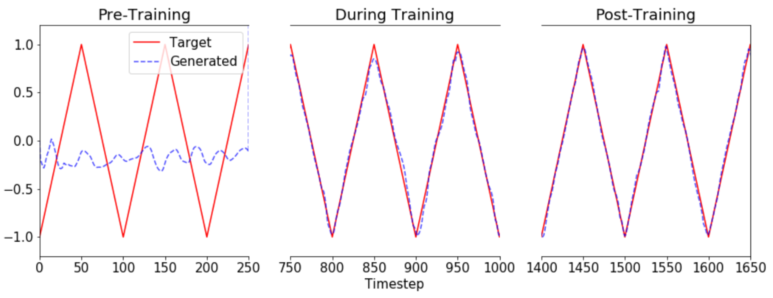

Figure 4: Network output before, during and after training. The network converges to the desired output within 1000 timesteps.

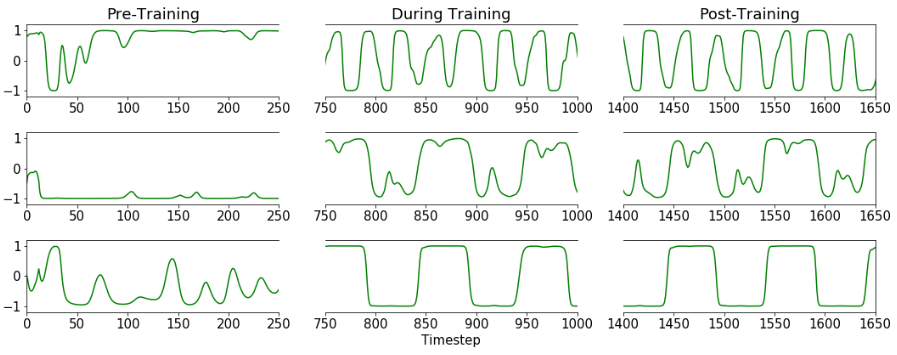

Figure 5: Activations of three random nodes before, during and after training. We see periodicity develop in the node activity during training

2.2 Sine Wave + Sawtooth

We also trained a slightly different network on a more complicated periodic waveform of a combined sawtooth and sine wave. In this case, we had to train for 3000 timesteps and we trained the output weights as well as the feedback weights.

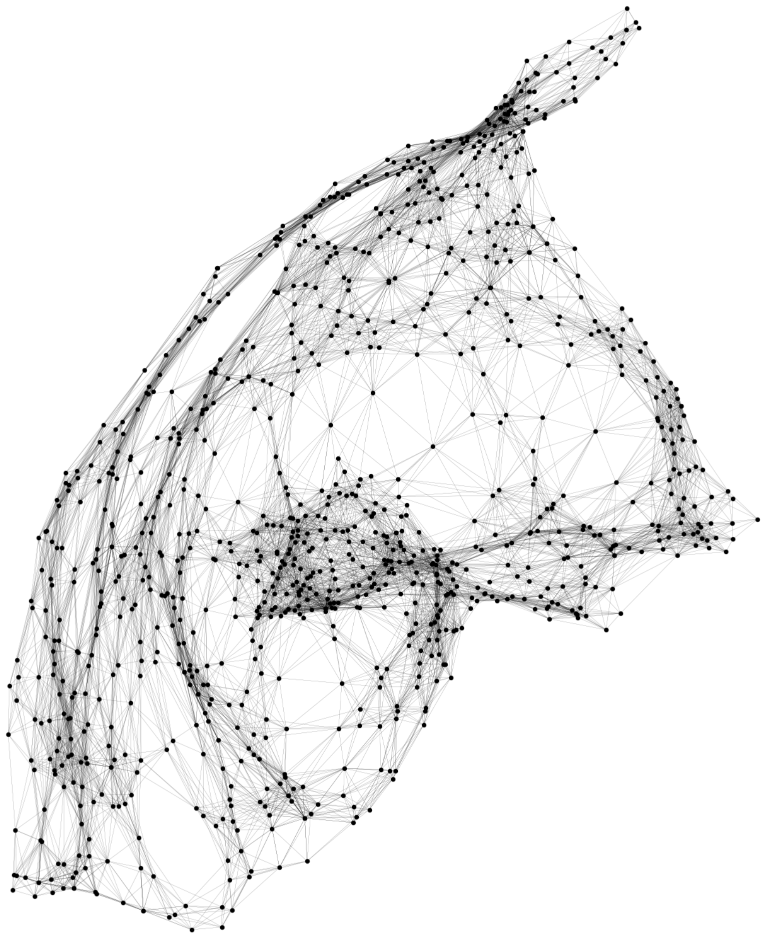

Figure 6: A visualization of the network structure which we trained on the combined periodic waveform.

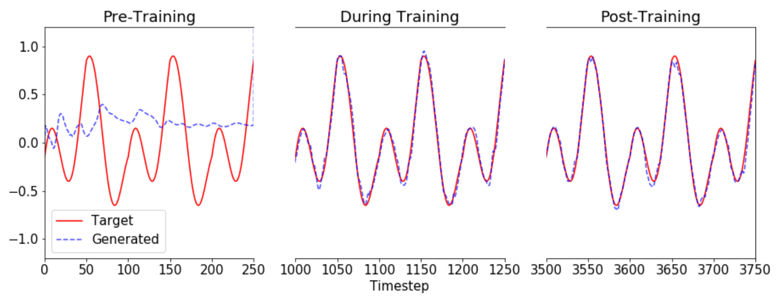

Figure 7: Network output before, during and after training. The network converges to the desired output within 3000 timesteps.

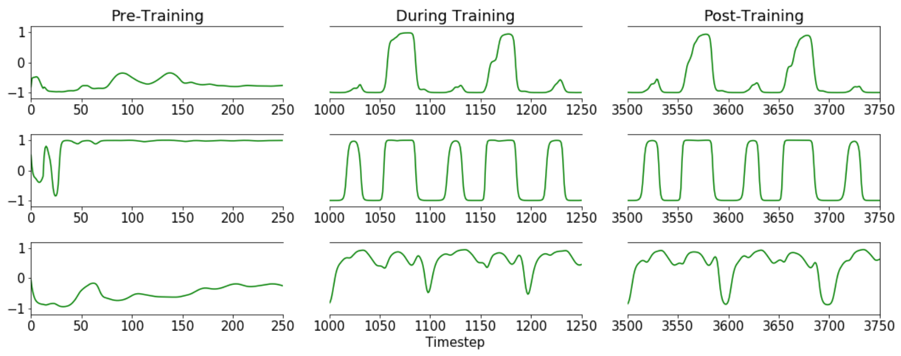

Figure 8: Activations of three random nodes before, during and after training. We see periodicity develop in the node activity during training. Some nodes show two separate periodicities.