Sapienza
Università di Roma

Recurrent neural networks as replicas of physical and biological stochastic systems

Facoltà di Scienze Matematiche, Fisiche e Naturali
Laurea Magistrale in Fisica

Lorenzo Bertini
ID number 1755287

Advisor
Prof. Maurizio Mattia

Academic Year September 2024

Thesis not yet defended

Recurrent neural networks as replicas of physical and biological stochastic systems
Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: bertini@1755287@studenti.uniroma1.it

Contents

Chapter 1

Reservoir computing framework

1.1 Recurrent neural networks

1.1.1 General concepts

In machine learning, a neural network (NN) is a model designed in analogy to the neuronal organization of biological neural networks, or brains. An artificial NN is made of units called *neurons*, which approximately model neurons in a brain. The neurons in the graph are connected to each other by edges that model synaptic connections.

Each neuron carries a signal $r_i(t)$, usually a real and limited number, that is the result of a applying some non-linear function of the sum of its inputs. This sum is called *activation*, and the function is called *activation function*. Each input is weighted by a synaptic strength: these can be mapped into a connectivity matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ for the network, where N is the number of neurons. Each element W_{ij} represents the strength of the input received by neuron j from neuron i. The internal state of the network is represented by the vector $\mathbf{r}(t) = \{r_1(t), \cdots, r_N(t)\} \in \mathbb{R}^N$.

1.1.2 Recurrent neural networks (RNNs)

In contrast to the uni-directional *feedforward neural networks*, where the flow of the signal is uni-directional, in *recurrent neural networks* (RNNs) the flow is bi-directional, meaning that the output of some node can affect subsequent input to the same node. This means that their topology can have cycles[2], and this allows them to:

1. Possibly develop a self-sustained temporal activation dynamics, even without a driving input signal. This makes RNNs *dynamical systems*, while FNNs are functions.

2. While driven by an input signal, preserve a nonlinear transformation of the input history in it's internal state $\mathbf{r}(t)$. This means that RNNs have *dynamical memory*.

1.1.3 Neuron types

There are two kinds of neurons used in RNNs[3]:

Artificial neurons The spiking activity of *artificial neurons* is obtained averaging the number of spikes per time interval, resulting in a rate-based activity. Here, a neuron state

$r(t)$ represents its average firing rate, and its evolution is described by discrete-time differential equations, in the form $r_{n+1} = f(r_n)$. At each time step, the input of a neuron is the sum of every other neuron activation at the previous time step, weighted by the strength matrix. The new activation for the neuron is given by a non-linear activation function f of this input

$$r_i(t) = f\left(\sum_j W_{ij} r_j(t-1)\right) \quad \forall i \qquad\qquad \mathbf{r}(t+1) = f(\mathbf{W}\mathbf{r}(t)) \tag{1.1}$$

Spiking neurons The activity of *spiking neurons*, usually seen in computational neuroscience, is modelled with individual spikes rather than averages, and they are described by continuous-time differential equations, in the form $\dot{r} = f(r)$. Here, the neuron state $r(t)$ represents the membrane potential. Using the aforementioned form for the evolution equation, one can see that corresponds to the equation for *integrate-and-fire* (IR) neuron models, where the current (the activation function) could be a function of the weighted sum of spikes from neighboring neurons

$$\frac{dr_i(t)}{dt} = I\left(\sum_j W_{ij} r_j(t)\right) \quad \forall i \qquad\qquad \frac{d\mathbf{r}(t)}{dt} = I(\mathbf{W}\mathbf{r}(t)) \tag{1.2}$$

1.1.4 RNN dynamics

RNNs carry out tasks by processing an input signal $\mathbf{u}(t) = \{u_1(t), \cdots, u_{N_u}(t)\} \in \mathbb{R}^{N_u}$ (where t is not necessarily time), which gets mapped into the network state by an input matrix $\mathbf{W}_i \in \mathbb{R}^{N_u \times N}$. In the evolution equation, an additional term $h(\mathbf{W}_i\mathbf{u}(t))$ gets added to $\mathbf{W}\mathbf{r}(t)$, where $h(\cdot)$ is a linear function.

The output of a network is a signal $\mathbf{y}(t) = \{y_1(t), \cdots, y_{N_y}(t)\} \in \mathbb{R}^{N_y}$, which gets extracted from the network with an output matrix $\mathbf{W}_o \in \mathbb{R}^{N \times N_y}$ and a linear function $g(\cdot)$: $\mathbf{y}(t) = g(\mathbf{W}_o\mathbf{r}(t))$. The linear functions h, g are usually the identity, and will often be omitted. The equation for the internal dynamics of a RNN in the discrete case is then

$$\mathbf{r}(t) = f(\mathbf{W}_i\mathbf{u}(t) + \mathbf{W}\mathbf{r}(t-1)) \tag{1.3}$$

$$\mathbf{y}(t) = g(\mathbf{W}_o\mathbf{r}(t)) \tag{1.4}$$

The input and the output signals are often called *input* and *output* (or *readout*) *layers*. The network is then called *hidden layer*, because it acts as a black box: the internal representation $\mathbf{r}(t)$ is not directly needed. Typically, $N \gg N_u, N_y$: the hidden layer internal space is dimensionally bigger than both the output and the input layers.

1.1.5 RNN training

Given an input signal $\mathbf{u}(t)$, the output signal $\mathbf{y}(t)$ produced by the network with such input, and a target output signal $\mathbf{y}_{\text{targ}}(t)$, training the network means tuning its weights such that a chosen error function $E(\mathbf{Y}, \mathbf{Y}_{\text{targ}})$ between the output and the target is minimized. This error can be for example the normalized root mean square (RMS):

$$E(\mathbf{Y}, \mathbf{Y}_{\text{targ}}) = \sqrt{\frac{\left\langle \left|\mathbf{y}(t) - \mathbf{y}_{\text{targ}}(t)\right|^2 \right\rangle_t}{\left\langle \left|\mathbf{y}_{\text{targ}}(t) - \left\langle \mathbf{y}_{\text{targ}}(t)\right\rangle_t\right|^2 \right\rangle_t}}$$

1.1.6 Leaky dynamics

The neuron models in (1.1) and (1.2) have no memory [1]: their state $r(t)$ depends fractionally and indirectly from the state at the previous step. These networks are good for modeling discrete-time systems with jumps. For slow and continuous systems, it is better to use networks with a continous dynamics. For spiking neurons one model is the *leaky integrate-and-fire* (LIF). This introduces a global time constant $\gamma = 1/\tau$ and a uniform leaking rate α[1]:

$$\frac{1}{\gamma}\frac{d\mathbf{r}}{dt} = -\alpha\mathbf{y}(t) + I\left(\mathbf{W}\mathbf{y}(t) + \cdots\right) \tag{1.5}$$

The dicrete dynamics for their rate-based counterparts can be obtained integrating equation (1.5) with a method of choice. The Euler method is the most commonly used, and yields the following:

$$\mathbf{r}(t+1) = (1-\alpha\gamma)\mathbf{r}(t) + \gamma f\left(\mathbf{W}\mathbf{r}(t) + \cdots\right) \tag{1.6}$$

Another popular design is to set $\alpha = 1$ and redefine $\gamma = \alpha$ as the leaking rate, yielding

$$\mathbf{r}(t+1) = (1-\alpha)\mathbf{r}(t) + \alpha f\left(\mathbf{W}\mathbf{r}(t) + \cdots\right) \tag{1.7}$$

1.1.7 Feedback layer and input bias

Some models extend the dynamics including a feedback from the output of the reservoir at the previous time step, through a feedback layer $\mathbf{W}_{\mathrm{f}} \in \mathbb{R}^{N_y \times N}$. Moreover, a constant input bias b_i can be added to each neuron input. The equation then becomes

$$\mathbf{r}(t) = f\left(\mathbf{W}_i\mathbf{u}(t) + \mathbf{W}\mathbf{r}(t-1) + \mathbf{W}_f\mathbf{y}(t-1) + \mathbf{b}\right)$$

1.2 Reservoir computing

Training RNNs traditionally involves tuning all the connections $\mathbf{W}_i, \mathbf{W}, \mathbf{W}_o$ by gradient-descent methods. This is inherently difficult to get right and computationally expensive. The *reservoir computing* framework was born trying to avoid the shortcomings of gradient-descent methods.

Correct and add stuff

It operates a conceptual and computational separation between the recurrent network, and the often linear readout that produces the output. In this technique, the RNN is a passive nonlinear temporal expansion function, called *reservoir* (hence the name) and does not get trained: it gets passively driven by an input signal, and maintains in its internal state a nonlinear transformation of the input history. Only the readout layer, that maps the internal state on the output vector, is obtained from training.

Tell about general reservoir advantages

[1] The equation for the membrane potential for leaky integrate-and-fire neurons is

$$C_m\frac{dV_m}{dt} = I(t) - \frac{V_m(t)}{R_m}$$

where a current $-V_m/R_m$ is added to model the membrane not being a perfect insulator. The perfect insulator limit is recovered for $R_m \to \infty$. The parameters are $\gamma = 1/C_m$, the reciprocal capacitance, and $\alpha = 1/R_m$, the conductance.

1.2.1 Echo state networks

Given the reservoir computing framework, the reservoir evolution dynamics, the problem is what property should the RNN posses to make a good reservoir.

Consider an input $\mathbf{u}(t) \in U^J, t \in J$, an input sequence $\mathbf{u}^h = \cdots, \mathbf{u}(h-1), \mathbf{u}(h)$ of h steps, and a RNN with an evolution operator T such that $\mathbf{r}(t+h) = T(\mathbf{r}(t), \mathbf{u}^h)$. Assume that U and A are compact. The network is said to have *echo states* if the network state $\mathbf{r}(t)$ is uniquely determined by any left-infinite input sequence $\mathbf{u}^{-\infty}$. In other words, for every input sequence, if the network has echo states, there is only one possible final state $\mathbf{r}(t)$. The property of having echo states is called *echo state property*, and a network with this property is called *echo state network* (ESN).

An equivalent formulation of this property would be to say that there exists an input echo function $E(\cdot)$ such that for all left infinite input histories $\cdots, \mathbf{u}(t-1), \mathbf{u}(t)$, the current network state is

$$\mathbf{r}(t) = E\left(\cdots, \mathbf{u}(t-1), \mathbf{u}(t)\right)$$

This means that the internal state can be understood as an "echo" of the input history (hence the name).

It can be proved[1] that in a network with this property the internal state asymptotically depends only on the driving input signal: the dependency on the initial condition is progressively lost. Having a fading memory gives ESNs some features:

Some sources are needed for this

1. They becomes robust to variation or inaccuracies in the initial state.
2. They can focus on capturing the temporal dependencies within the input data without being "distracted" by the initial state.
3. ??

One of the proposed implementation of a reservoir computing model with a discrete dynamics makes use of an echo state network as a reservoir, and it's usually also called echo state network (ESN).

1.2.2 Training

As already noted, the key point of reservoir computing is to tune only the readout layer. This is a common supervised non-temporal task of mapping an input to a desired output. Given an input signal $\mathbf{u}(t)$ and a corresponding desired output $\mathbf{y}(t)$, the input is used to "drive" the reservoir according to equation (1.3), producing $\mathbf{r}(t)$. The trained output layer $\mathbf{W}_o$ should then satisfy this system of linear equations

$$\mathbf{W}_o \mathbf{r}(t) = \mathbf{y}(t) \quad \forall t$$

The time series $\mathbf{u}(t)$, $\mathbf{r}(t)$ and $\mathbf{y}(t)$ can be arranged into matrices (one dimension being time) with $\mathbf{U} \equiv [\mathbf{u}(1), \dots, \mathbf{u}(T)] \in \mathbb{R}^{N_u \times T}$, $\mathbf{R} \in \mathbb{R}^{N \times T}$ and $\mathbf{Y} \in \mathbb{R}^{N_y \times T}$ respectively. With this notation, the problem becomes finding $\mathbf{W}_o$ as a solution of the linear system $\mathbf{W}_o \mathbf{R} = \mathbf{Y}$.

Finding solutions to an overdetermined system of linear equations is a common problem called *linear regression*. The normal equation formulation of the problem would be $\mathbf{W}_o \mathbf{X} \mathbf{X}^T = \mathbf{Y} \mathbf{X}^T$. The standard approach is ordinary least squares regression: this procedure minimizes the euclidean norm $\|\mathbf{W}_o \mathbf{R} - \mathbf{Y}\|^2$, which is indeed the loss function. Then, Tikhonov regularization

$$\mathbf{W}_o = \mathbf{Y}^T \mathbf{X}^T (\mathbf{X}\mathbf{X}^T + \beta \mathbf{I})^{-1}$$

where $\mathbf{I}$ is the identity matrix and β is a regularization coefficient.

Bibliography

[1] Herbert Jaeger, The “echo state” approach to analysing and training recurrent neural networks – with an Erratum note

[2] Mantas Lukoševičius, Herbert Jaeger, Reservoir computing approaches to recurrent neural network training, Computer Science Review 3 (2009) 127–149

[3] Matteo Cucchi et al, Hands-on reservoir computing: a tutorial for practical implementation

[4] Kohei Nakajima Ingo Fischer, Reservoir Computing: Theory, Physical Implementations, and Applications

[5] Mantas Lukoševičius, A Practical Guide to Applying Echo State Networks