



# Machine learning with Random Neural Networks

Gerardo Rubino

## ► To cite this version:

Gerardo Rubino. Machine learning with Random Neural Networks. 2018 - Conference on Deep learning: From theory to applications, Centre Henri Lebesgue and Technicolor, Sep 2018, Rennes, France. pp.1-34. hal-01962948

**HAL Id: hal-01962948**

**<https://inria.hal.science/hal-01962948>**

Submitted on 21 Dec 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Machine learning with Random Neural Networks

G. Rubino

INRIA Rennes, France

September 2018

# Outline

- 1 — RNNs
- 2 — Queuing origins
- 3 — Extension in the Reservoir Computing class
- 4 — About deep architectures
- 5 — Conclusions and current projects
- 6 — Some references

# Abstract

- At the DIONYSOS team we have been using ML tools for more than 10 years, for specific networking problems:
  - the PSQA (Pseudo-Subjective Quality Assessment) project, mapping QoS + channel metrics into quantified QoE assessments,
  - time series predictions (for a PSQA 2.0 generation).
- Our main learning problems belong to the Supervised Learning area. we are starting to explore deep architectures.
- In the presentation, I will focus only on the used tool, the Random Neural Network.

# Outline

1 — RNNs

2 — Queuing origins

3 — Extension in the Reservoir Computing class

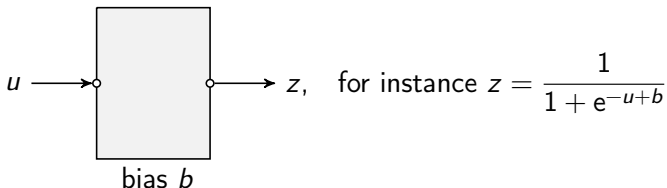
4 — About deep architectures

5 — Conclusions and current projects

6 — Some references

## Classic artificial neurons

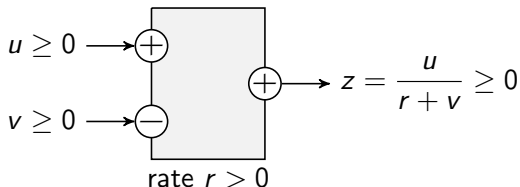
- A “classic artificial neuron” can be seen as a parametric real function of a real variable:



- We consider  $b$  as a parameter of the function implemented by the neuron (called *activation* function).
- There are many different activation functions used in practice:  
 $z = \tanh(u - b)$ ,  $z = 1(u \geq b)$ , ReLU...

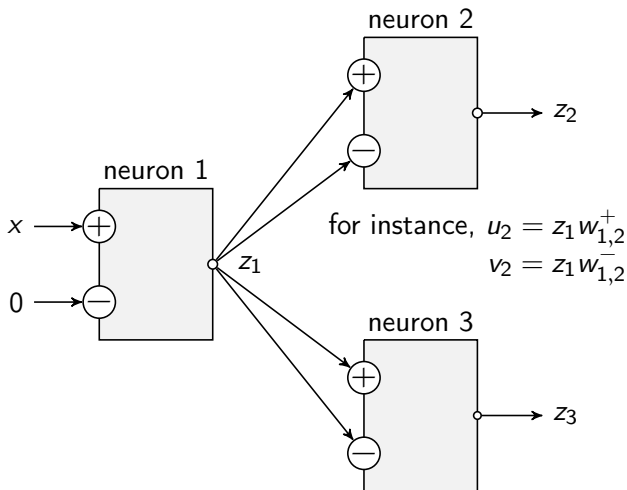
## Random Neurons

- A **Random Neuron** is a parametric positive real function of two real positive variables (we also say “ports”) called “positive” and “negative” ports.



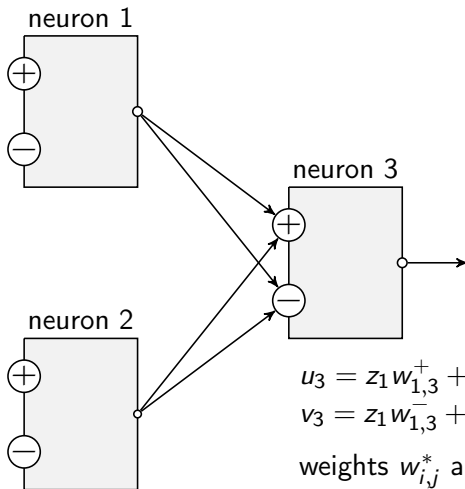
- We see the number  $r > 0$  as a parameter (the *rate* of the neuron).
- There are several variants of this model. In the main and original one, we must use the function  $z = \min(u/(r + v), 1)$ .
- Inventor: E. Gelenbe, Imperial College, in the late 80s.
- Observe that there is nothing random here.

A Random Neural Network (RNN) is a set of interconnected RNs. The connections are weighted by **nonnegative** reals.





Junctions at input ports are additive:



## Possible restriction

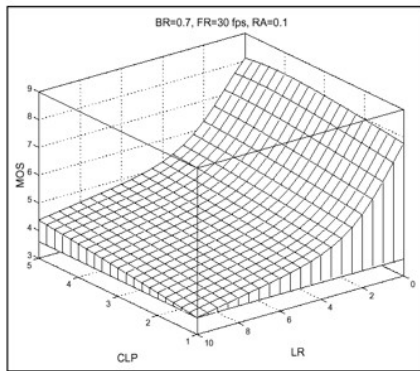
- Because of the queuing origin of the RNN tool, in order to respect the strict meaning of the model we must have

$$\text{for each neuron } i, \quad \sum_j (w_{ij}^+ + w_{ij}^-) = r_i.$$

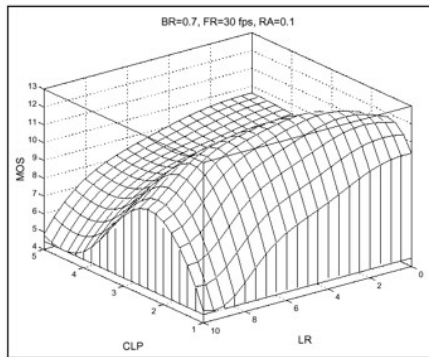
- This is a constraint to be respected in order to keep the queueing interpretation (and thus, to access all the theory and algorithms associated with).
- The same reason explains the use of  $z = \min(u/(r + v), 1)$  instead of  $z = u/(r + v)$  (more on this below).

## Why using RNNs in our PSQA project?

Because of the favorable comparison with standard software (at the time, several years ago). An example: with the same data and with the same cost (more precisely, the same number of weights), RNN on the left, Matlab on the right (an old version of its ToolBox on ML).



(a) Correctly trained



(b) Example of an over-trained ANN

# Outline

1 — RNNs

2 — Queuing origins

3 — Extension in the Reservoir Computing class

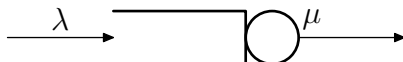
4 — About deep architectures

5 — Conclusions and current projects

6 — Some references

# Origin of the model

## Refresher:

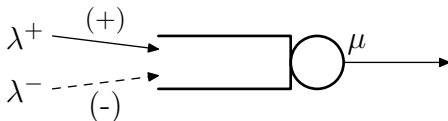


Consider the M/M/1 model, with arrival rate  $\lambda$  and service rate  $\mu$ . The queue is stable iff  $\lambda < \mu$ ; in that case, the steady state distribution  $(\pi_n)_{n \geq 0}$  is given by  $\pi_n = (1 - \rho)\rho^n$ ,  $n \in \mathbb{N}$ , where the number  $\rho = \lambda/\mu = 1 - \pi_0$  is the utilization factor, or load, of the system.

- $\rho = \Pr(\text{ system busy at infinity })$ ; if  $\lambda \geq \mu$ , then we have that  $\Pr(\text{ there exists } t < \infty \text{ such that after } t, \text{ system is always busy }) = 1$ .
- See then that the load of the queue at infinity,  $\rho$ , satisfies

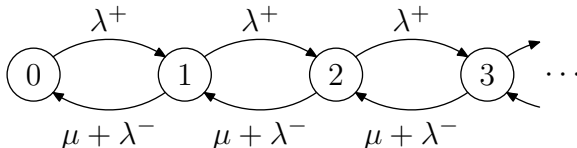
$$\rho = \min\left\{\frac{\lambda}{\mu}, 1\right\}.$$

## A G-queue



A basic G-queue: positive (standard) customers arrive with rate  $\lambda^+$ ; negative ones arrive with rate  $\lambda^-$ ; both arrival processes are Poisson; the service rate is  $\mu$ . Add usual independence conditions. To understand the semantics, see the graph below.

We have stability  $\iff \lambda^+ < \mu + \lambda^-$ , and there,  $\rho = \frac{\lambda^+}{\mu + \lambda^-}$ .



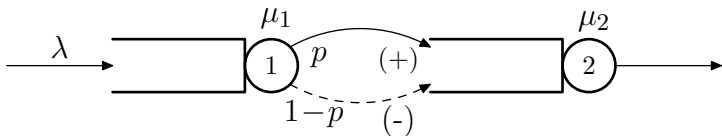
The Markov graph associated with the basic G-queue depicted above.

# The dynamics

In words,

- positive customers behave as ordinary ones;
- negative customers are actually signals: when a negative unit arrives at the queue, if there is some positive customer there, it removes it from the system (both “disappear”), and if the queue is empty, nothing happens;
- only positive customers can be observed; for negative ones, only their action can;
- “G-queue” and “Random Neuron” are two names for the same mathematical object;
- there are two different vocabularies, depending on the context: “G-queue”  $\equiv$  “Random Neuron”; “backlog of the queue”  $\equiv$  “potential of the neuron”; “busy queue”  $\equiv$  “active or excited neuron”; etc.

# From G-queues to G-networks



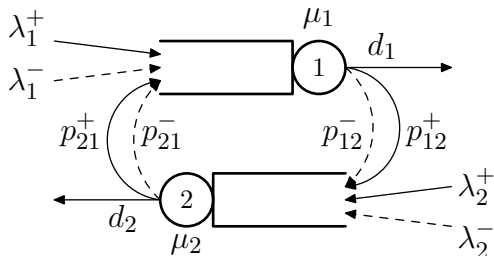
A tandem: first queue is an M/M/1; when leaving the first node, with probability  $p$  customers go to a second queue as positive ones, and with probability  $1 - p$  as negative signals; service rate at queue  $i$  is  $\mu_i$ ,  $i = 1, 2$ .

Here, theory says that stability happens  $\iff \lambda < \mu_1, \mu_2$ , and in that case, if  $X_i = 1$  ( queue  $i$  is busy at  $\infty$  ),  $i = 1, 2$ , we have, for  $j, k = 0, 1$ :

$$\Pr(\text{ at } \infty, X_1 = j, X_2 = k ) = [(2\rho_1 - 1)j + 1 - \rho_1] [(2\rho_2 - 1)k + 1 - \rho_2].$$



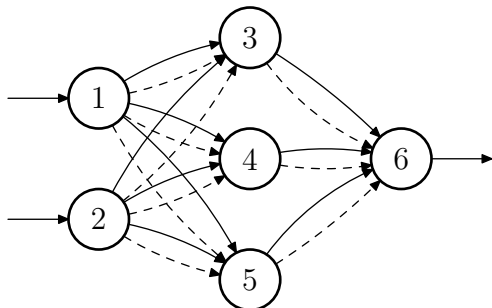
## A recurrent G-network



A general recurrent G-network with two nodes. We denote by  $d_i$  the probability of leaving the network after a service at queue  $i$ .

The asymptotic independence of the indicators of busy neurons is valid in general, so also here (and potentials are also asymptotically independent).

## A 3-layer structure

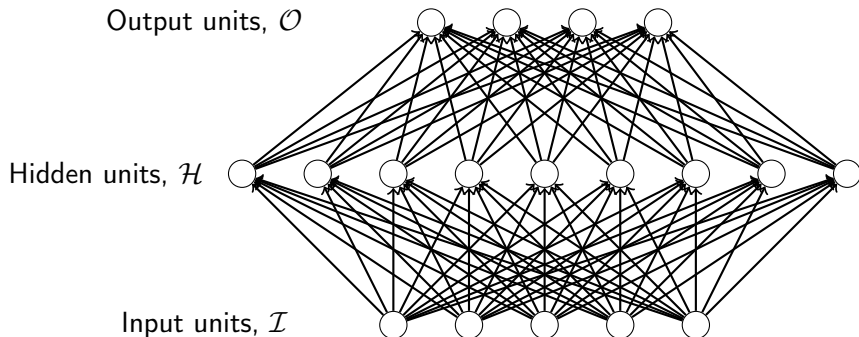


The graph of a 3-layer RNN of the (2,3,1) type (as before, dashed arrows correspond to flows of negative customers/signals).

## On the weights of a RNN

- Think queueing: when a customer leaves queue  $i$  (necessarily a positive customer), it goes to queue  $j$  as a positive customer with some fixed probability  $p_{i,j}^+$ , or as a negative customer with some fixed probability  $p_{i,j}^-$ .
- Then, the mean throughput (the frequency) of the flow of positive customers (of spikes) from  $i$  to  $j$  is  $r_i p_{i,j}^+ =: w_{i,j}^+$ , and of negative ones is  $r_i p_{i,j}^- =: w_{i,j}^-$ .
- Since for any neuron  $i$  in the network, we must have  $\sum_j (p_{i,j}^+ + p_{i,j}^-) = 1$  (add a queue/neuron 0 representing the network's environment), we have  $\sum_j (w_{i,j}^+ + w_{i,j}^-) = r_i$ .

# The general 3-layer Random Neural Network



Network inputs: those flows arriving at the input neurons; network outputs: the flows leaving the output neurons.

# Random Neural Networks implement rational functions

- Assume the negative ports *of input neurons* aren't used. Assume a single output neuron, so, a scalar network output.
- Call  $x_i$  the signal arriving at the positive port of input neuron  $i$ . Then, we can explicitly write the network output as a function of the inputs.

$$z_o = \frac{\sum_{h \in \mathcal{H}} \frac{\sum_{i \in \mathcal{I}} \frac{x_i}{r_i} w_{i,h}^+}{r_h + \sum_{i \in \mathcal{I}} \frac{x_i}{r_i} w_{i,h}^-} w_{h,o}^+}{r_o + \sum_{h \in \mathcal{H}} \frac{\sum_{i \in \mathcal{I}} \frac{x_i}{r_i} w_{i,h}^+}{r_h + \sum_{i \in \mathcal{I}} \frac{x_i}{r_i} w_{i,h}^-} w_{h,o}^-}.$$

- The output is thus a rational function of the input. This allows many treatments. Also, for learning, costs (errors) are **rational functions** of weights, also leading to simple and powerful treatments.

# Outline

1 — RNNs

2 — Queuing origins

3 — Extension in the Reservoir Computing class

4 — About deep architectures

5 — Conclusions and current projects

6 — Some references

# Reservoir Computing

- The idea of these models is to develop systems that use the potential for *memorization* of recurrent neural networks without their difficulties in the training process.
- They appeared at the beginning of the 2000s, and are known today under the name of *Reservoir Computing* (RC) paradigm.
- The two most popular RC models are
  - the *Echo State Network (ESN)*  
(H. Jaeger, “The echo state approach to analyzing and training recurrent neural networks”, German National Research Centre for Information Technology, Tech. Rep. 148, 2001);
  - and the *Liquid State Machine (LSM)*  
(W. Maass, “Liquid state machines: Motivation, theory, and applications,” in *Computability in Context: Computation and Logic in the Real World*, Imperial College Press, 2010, pp. 275-296).

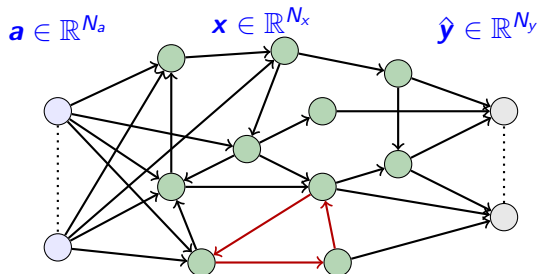
# Echo State Networks (ESNs)

The main representative of the family. It has

- a first recurrent part called *reservoir*, with **fixed weights**,
- and a second supervised learning part called *readout*.

Three-layered neural networks:

- Input layer
- Hidden layer
- Output layer



The learning process is restricted to the output weights (readout).



## Our Echo State Queueing Networks (ESQNs)

- Echo State Queueing Networks: applying the RC idea to a RNN. Joint work with S. Basterrech (Univ. of Prague).
- Three sets of neurones, a recurrent topology “in the middle”.
- Input at time  $t$ :  $\mathbf{a}(t) = (a_1(t), \dots, a_{N_a}(t))$ .
- Load of input neuron  $u \in 1..N_a$ :  
 $\rho_u(t) = a_u(t)/r_u$  (in general, we take  $a_u(t) < r_u$ ).
- For all reservoir units  $u = N_a + 1, \dots, N_a + N_x$ ,

$$\rho_u(t) = \frac{\sum_{v=1}^{N_a} \frac{a_v(t)}{r_v} w_{v,u}^+ + \sum_{v=N_a+1}^{N_a+N_x} \rho_v(t-1) w_{v,u}^+}{r_u + \sum_{v=1}^{N_a} \frac{a_v(t)}{r_v} w_{v,u}^- + \sum_{v=N_a+1}^{N_a+N_x} \rho_v(t-1) w_{v,u}^-}.$$

- The input space is thus “projected” into a new “larger” space.
- The models then compute a linear regression from the projected space into the output space.
- Thus, the network output  $\hat{\mathbf{y}}(t) = (\hat{y}_1(t), \dots, \hat{y}_{N_b}(t))$  is computed for any  $m \in 1..N_b$ :

$$y_m(t) = w_{m,0}^{\text{out}} + \sum_{i=1+N_a}^{N_a+N_x} w_{m,i}^{\text{out}} \rho_i(t).$$

- Learning process: the output weights  $w_*^{\text{out}}$  can be computed using any (fast) procedure, for instance, Least Mean Square algorithms.
- Remark: we can replace this simple structure by, for instance, a classical feedforward 3-level RNN.

# Outline

1 — RNNs

2 — Queuing origins

3 — Extension in the Reservoir Computing class

4 — About deep architectures

5 — Conclusions and current projects

6 — Some references

## Some ongoing work on Deep RNNs

- Imperial College is developing some extensions allowing exploring deep architectures based on RNNs.
- Specific tests have been done using well-known databases that show good performances of the model against classic ones.
- The linear side of convolutional structures are obtained by appropriate parameters' choices in the RNN, and then using power expansions related to the denominator of the rational function associated with a Random Neuron.

## Some preliminary experiments at Inria

- In our team at Inria, in collaboration with Technicolor, we can report some preliminary results concerning mixed architectures concerning classic and random neurons.
- The idea is to see the effect of replacing whole layers of neurons in a deep network by RNN-based ones.
- Again, using standard databases, we obtained promising results when RNN-based layers appear at the end rather than in the first layers of the network.

# Outline

1 — RNNs

2 — Queuing origins

3 — Extension in the Reservoir Computing class

4 — About deep architectures

5 — Conclusions and current projects

6 — Some references

## I refer only to the ML side here

- Random Neural Networks are, at least, interesting competitors in the ML field.
- Several theoretical questions around Reservoir Computing with our ESQN model (mainly around convergence, stability) are open.
- In PSQA, the labels come from subjective tests (that is, from panel of human subjects). We have a project to eliminate the use of those panels. This leads to
  - Big Data problems, coming from the extensive use of automatic but less performant tools to provide quality assessments,
  - then, to the exploration of the use of deep learning architectures.

# Outline

1 — RNNs

2 — Queuing origins

3 — Extension in the Reservoir Computing class

4 — About deep architectures

5 — Conclusions and current projects

6 — Some references



## Some local references

- Maths behind our main application:  
*“Quantifying the Quality of Audio and Video Transmissions over the Internet: The PSQA Approach”*, G. Rubino, in Design and Operations of Communication Networks: A Review of Wired and Wireless Modelling and Management Challenges, edited by J. Barria, Imperial College Press, 2005.
- Practical aspects of our uses of RNN in learning:  
*“Evaluating Users’ Satisfaction in Packet Networks Using Random Neural Networks”*, G. Rubino, P. Tirilly and M. Varela, Springer-Verlag Lecture Notes in Computer Science, no. 4132, 2006.
- An example of using RNNs in combinatorial optimization:  
*“A GRASP algorithm with RNN-based local search for designing a WAN access network”*, H. Cancela, F. Robledo and G. Rubino, Electronic Notes in Discrete Mathematics 18 (1), 59–65, December 2004.

- An example of application of PSQA:  
*“Controlling Multimedia QoS in the Future Home Network Using the PSQA Metric”*, J.-M. Bonnin, G. Rubino and M. Varela, in The Computer Journal, 49(2):137–155, 2006.
- On the design of a P2P streaming network based on PSQA:  
*“A robust P2P streaming architecture and its application to a high quality live-video service”*, H. Cancela, F. Robledo Amoza, P. Rodríguez-Bocca, G. Rubino and A. Sabiguero, in Electronic Notes in Discrete Mathematics 30: 219–224, 2008,  
 plus another paper with a demo,  
*“Automatic Quality of Experience Measuring on Video Delivering Networks”*, D. De Vera, P. Rodríguez-Bocca and G. Rubino, in SIGMETRICS Performance Evaluation Review, Vol. 36, Issue 2, associated with a demonstration at Sigmetrics’08 awarded with the Best Demonstration Prize.

- An example of improvement on the initial RNN tool:  
“*Levenberg-Marquardt Training Algorithms for Random Neural Networks*”, S. Basterrech, S. Mohammed, G. Rubino and M. Soliman, in The Computer Journal, Vol. 54, N. 1, 125–135, 2011.
- An example of extension of the initial RNN tool:  
“*Echo State Queueing Networks: a combination of Reservoir Computing and Random Neural Networks*”, S. Basterrech and G. Rubino, in Probability in the Engineering and Informational Sciences, Vol. 31, No. 4, pp. 1–16, 2017.