Random Neural Networks and extensions; applications in computer networks

Gerardo Rubino

Random Neural Networks and extensions; applications in computer networks

G. Rubino

INRIA Rennes, France

May 2018

Outline

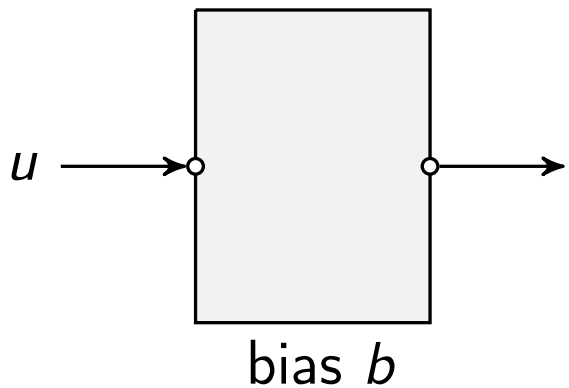
Abstract

- 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).
- Outputs: 4 PhDs, many MS thesis and engineer thesis, several man-years of engineering work, several collaborative project (French and European), one real implementation (a P2P streaming network controlled by PSQA), papers, ...
- Our main learning problems belong to the Supervised Learning area.
- In the presentation, I will focus only on the used tool, the Random Neural Network.
- There is another set of combinatorial optimization activities in networking that we developed using RNNs (combined with GRASP), but we don't discuss them here.

Outline

Classic artificial neurons

- A "classical artificial neuron" can be seen, for instance, as a parametric real function of a real variable.

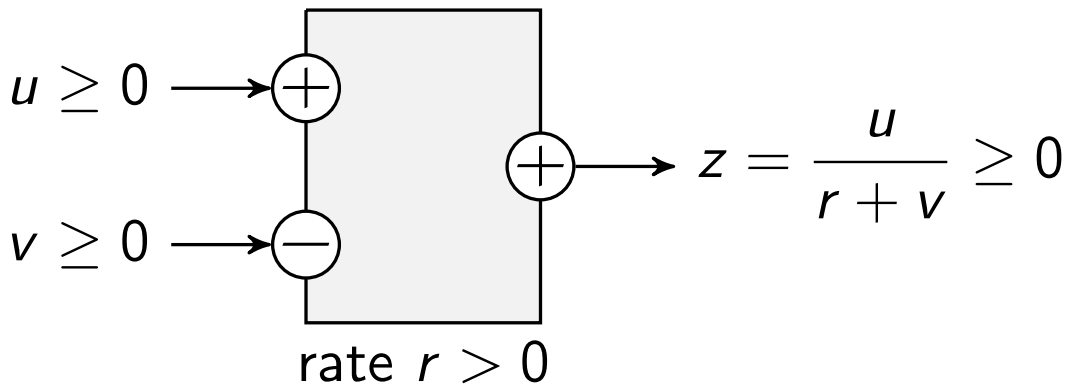


z, for instance $z = \dfrac{1}{1 + \mathrm{e}^{-u+b}}$

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

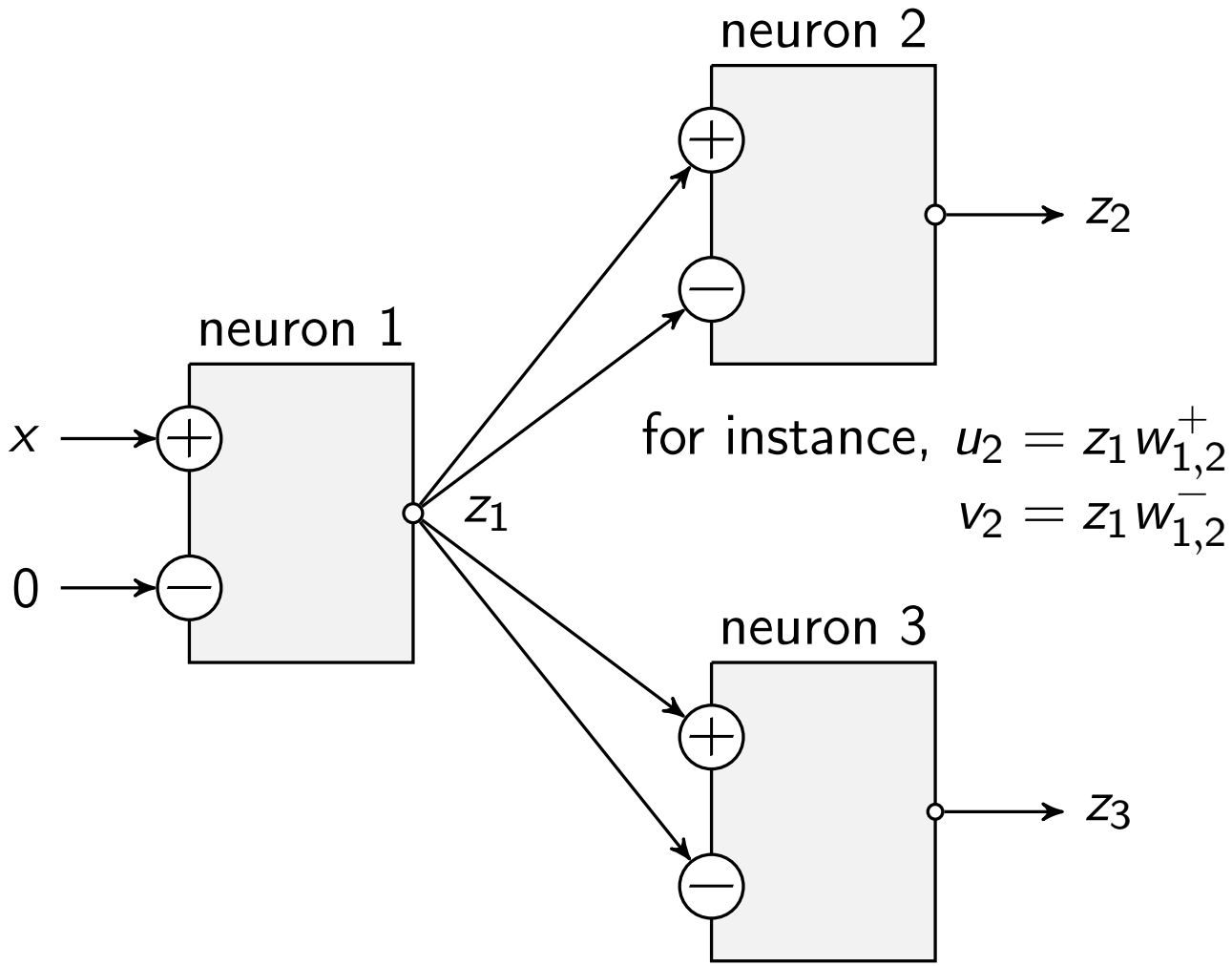
Random Neurons

- A Random Neuron is a parametric positive real function of two real positive variables (we also say "ports"), one called the "positive port", the other one being the "negative port".

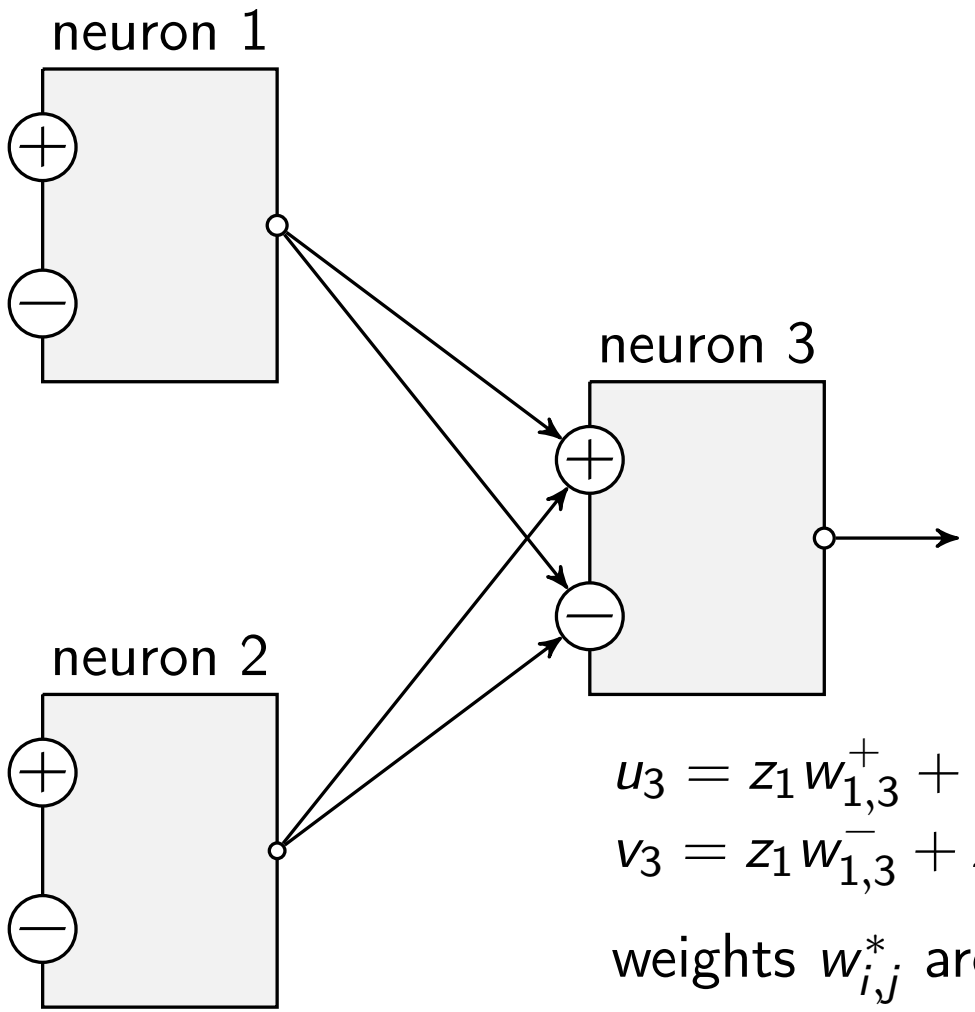


- We see $r > 0$, the *rate* of the neuron, as a parameter.
- There are several variants of this model. In the main and original one, we must use the function $z = \min(u/(r+v),\ 1)$.
- Inventor: E. Gelenbe, Imperial College, in the late 80s.
- Observe that there is nothing random here.

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



Jonctions at input ports are additive:



$$u_3 = z_1 w^+_{1,3} + z_2 w^+_{2,3}$$
$$v_3 = z_1 w^-_{1,3} + z_2 w^-_{2,3}$$

weights $w^*_{i,j}$ are nonnegative

Possible restriction

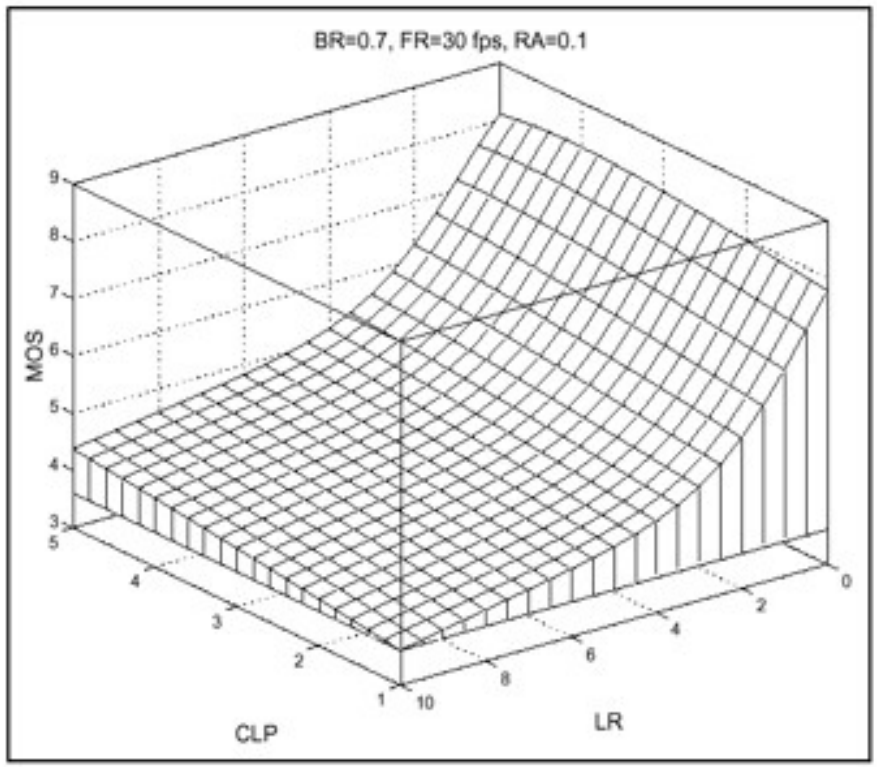
- Because of the queuing origin of the RNN tool, to respect the strict meaning of the model, we must have

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

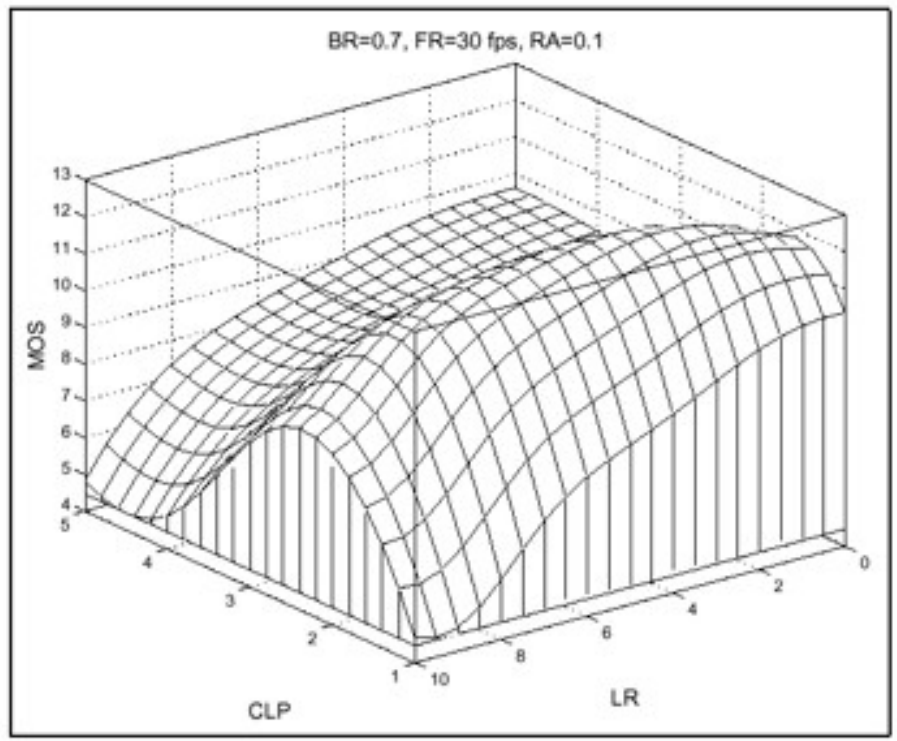
- This is a constraint to be respected in order to keep the queueing interpretation (and thus, to access all the theory and algorithms associated with).
- The same reason explains the use of $z = \min(u/(r+v),\, 1)$ instead of $z = u/(r+v)$.

Why using RNNs in our PSQA project?

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



(a) Correctly trainined



(b) Example of an over-trained ANN

Outline

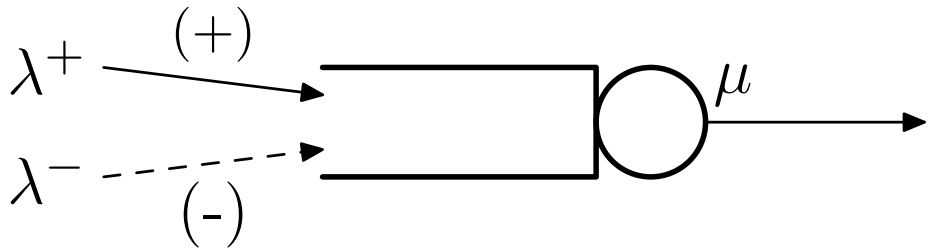
Origin of the model

Consider the M/M/1 model, with arrival rate λ and service rate μ. The queue is stable iff $\lambda < \mu$; in that case, the steady state distribution (π_n) 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.

- $\rho = \Pr($ 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, ρ, satisfies

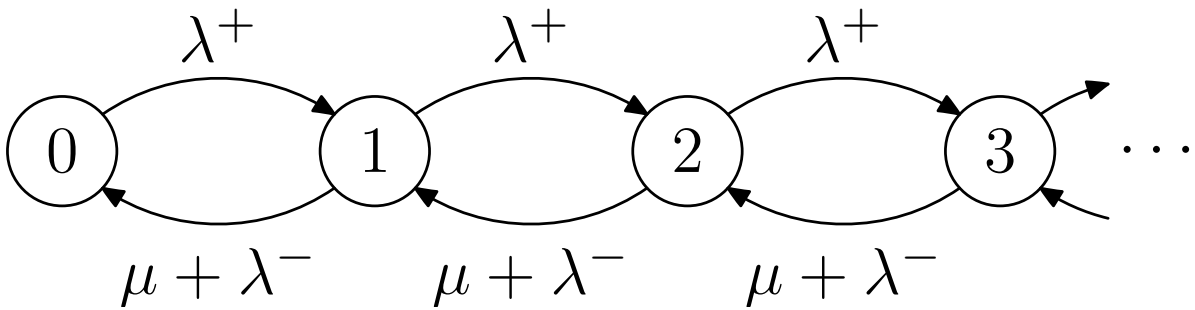
$$\rho = \min\Big\{\frac{\lambda}{\mu},\ 1\Big\}.$$

A G-queue



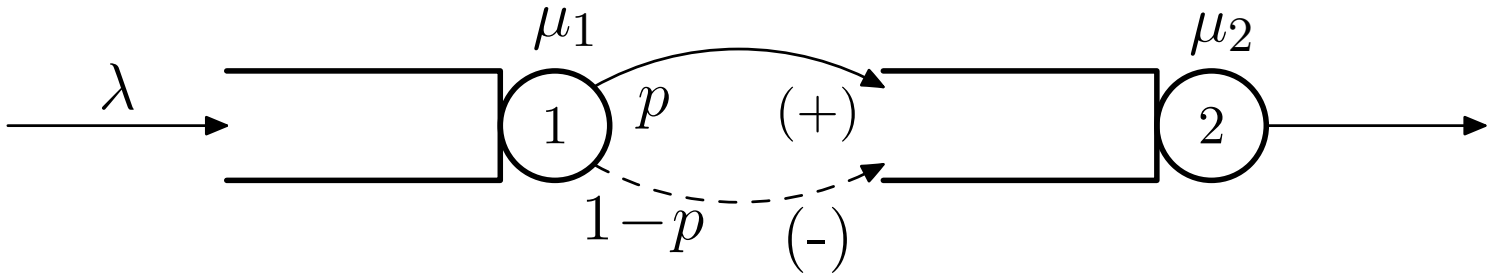
A basic G-queue: positive (standard) customers arrive with rate λ^+; negative ones arrive with rate λ^-; both arrival processes are Poisson; the service rate is μ. To understand the semantics, see the graph below.

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



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

From G-queues to G-networks

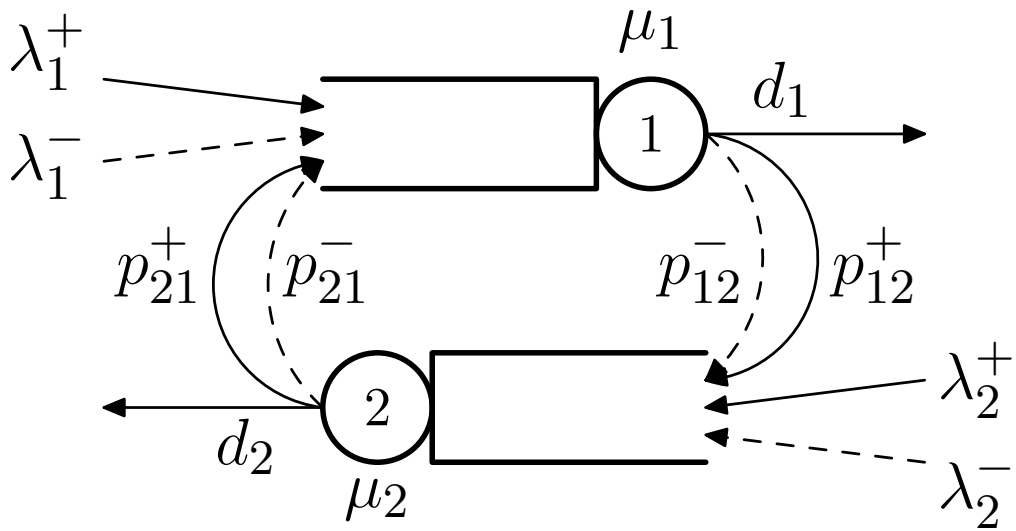


A tandem: first queue is an M/M/1; when leaving the first node, with probability p customers go to a second queue as positive ones, and with probability $1-p$ as negative signals; service rate at queue i is μ_i, $i = 1, 2$.

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

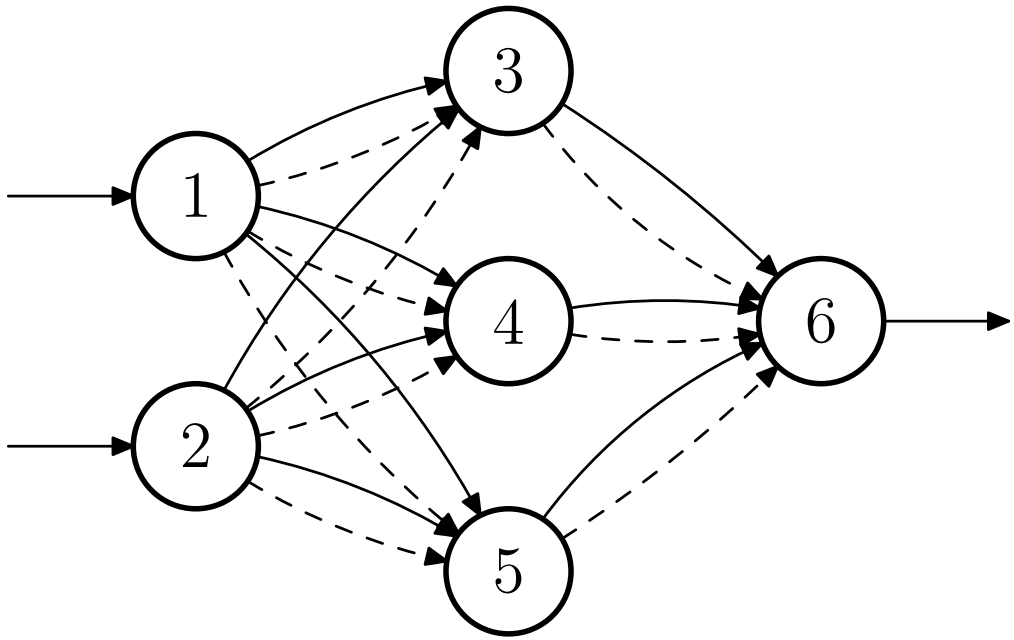
$$\Pr(\text{ at infinity, } X_1 = j, X_2 = k\ ) = \rho_1^j \rho_2^k.$$

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.

A 3-layer structure

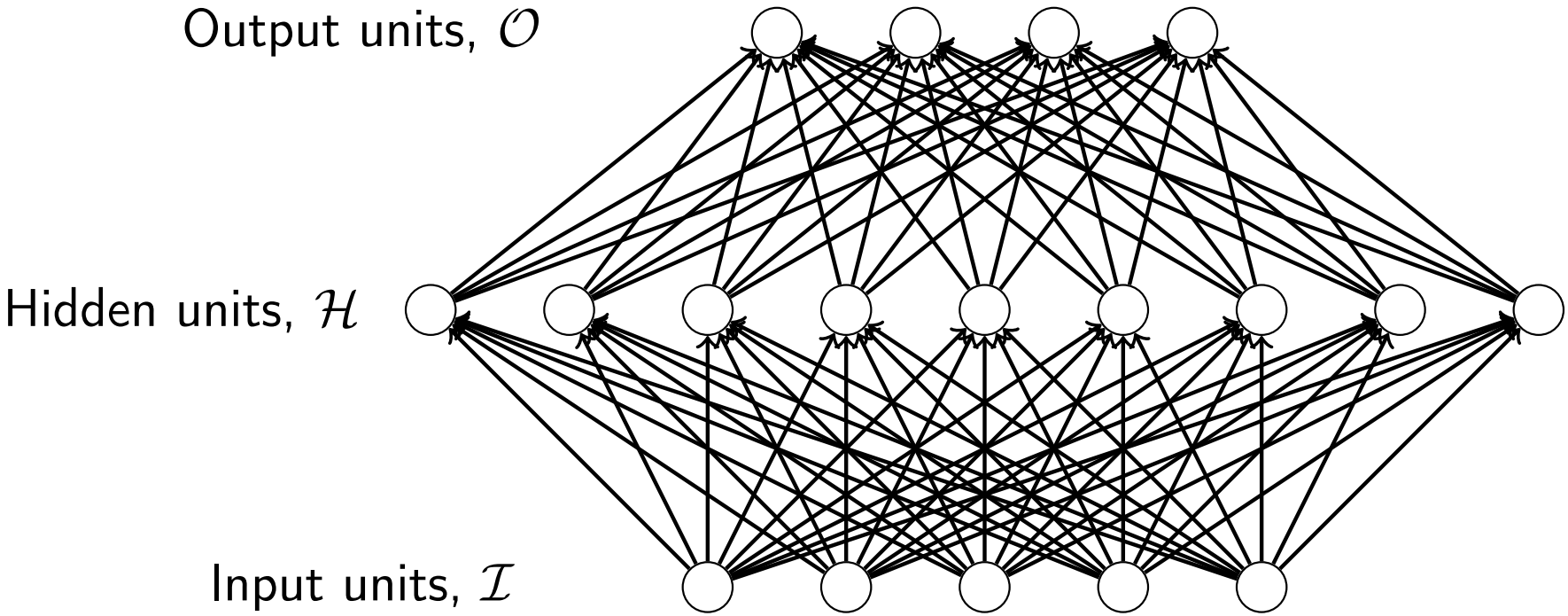


The graph of a 3-layer RNN of the (2,3,1) type (as before, dashed arrows correspond to flows of negative customers/signals).

On the weights of a RNN

- Think queueing; then, when a customers leaves queue i (necessarily a positive customer), it goes to queue j as a positive customer with some fixed probability $p^+_{i,j}$, and 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 $\sum_j \left(p^+_{i,j} + p^-_{i,j}\right) = 1$ (add a queue/neuron 0 representing the network's environment), we deduce that $\sum_j \left(w^+_{i,j} + w^-_{i,j}\right) = r_i$

The general 3-layer Random Neural Network



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.

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

- This shows that the output is a rational function of the input. This allows many treatments. Also, for learning, costs (errors) are rational functions of weights, leading to many advantages over other forms.

Outline

Reservoir Computing

- The idea was to develop models that use the potential for *memorization* of recurrent neural networks without the difficulties in the training process of these networks.
- 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 analysing 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).

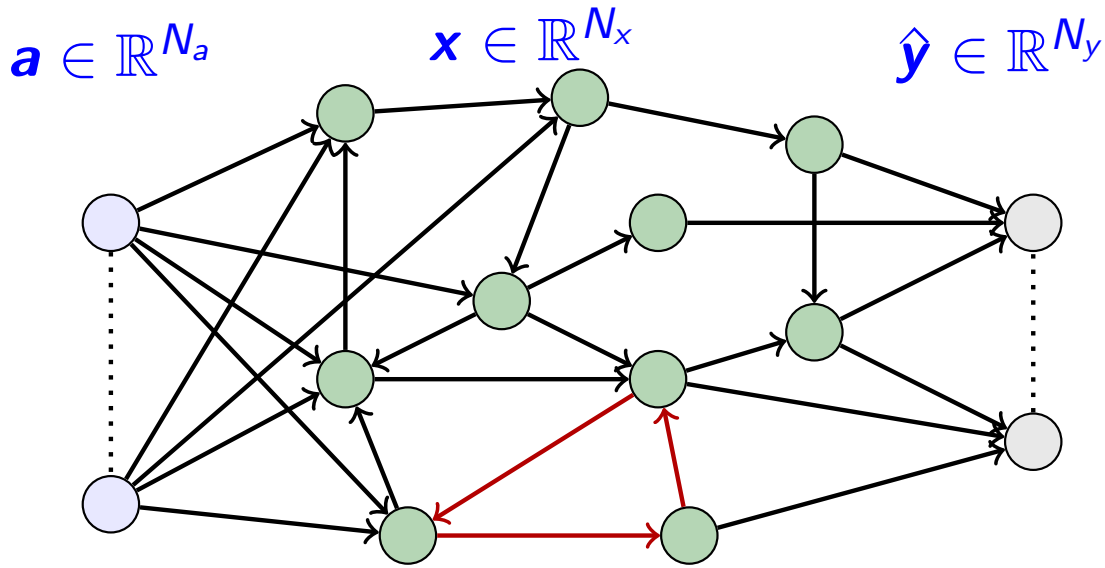
Echo State Networks (ESNs)

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, $\boldsymbol{a}(t) = (a_1(t), \ldots, a_{N_a}(t))$:
  $\rho_u(t) = a_u(t)/r_u$, (in general, we take $a_u(t) < r_u$), for $u \in 1..N_a$.
- For all reservoir units $u = N_a + 1, \ldots, N_a + N_x$,

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

- The input space in then projected into a new "larger" space.
- We compute a linear regression from the projected space to the output space.
- Thus, the network output $\hat{\boldsymbol{y}}(t) = (\hat{y}_1(t), \ldots, \hat{y}_{N_b}(t))$ is computed for any $m \in 1..N_b$:

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

- Learning process: the output weights w^{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

I refer only to the ML side

- Analysis of many theoretical questions around Reservoir Computing with our ESQN model (mainly around convergence, stability).
- In PSQA, the labels come from subjective tests (that is, from panel of human subjects). We have an idea 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 Convolutional Neural Networks, etc.

Outline

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 high-level
- 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.