Random Neural Networks and Extensions. Applications to networking

Gerardo Rubino

TMA Conference 2018
TMA Experts Summit

Random Neural Networks and Extensions. Applications to networking.

G. Rubino

INRIA Rennes, France

Vienna, June 2018

Outline

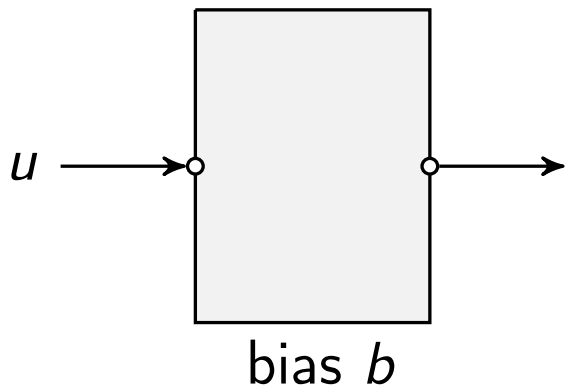
Abstract

- At the DIONYSOS INRIA 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;
  - PSQA is a technology measuring QoE accurately and in real time
  - time series predictions (for a PSQA 2.0 generation of the tool).
- Our main learning problems belong to the Supervised Learning area.
- In the presentation, I will only focus 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

- Let us see a "classic artificial neuron" as a parametric real function of a real variable.

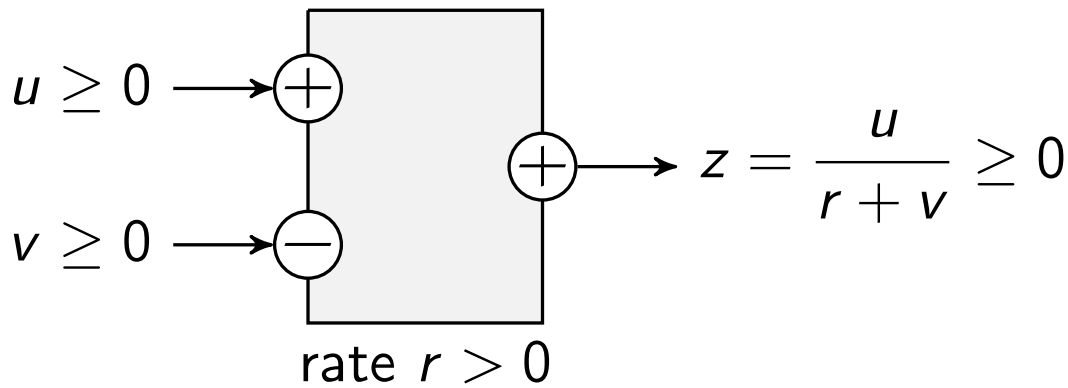


$$\text{for instance, } z = \frac{1}{1 + \mathrm{e}^{-u+b}}$$

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

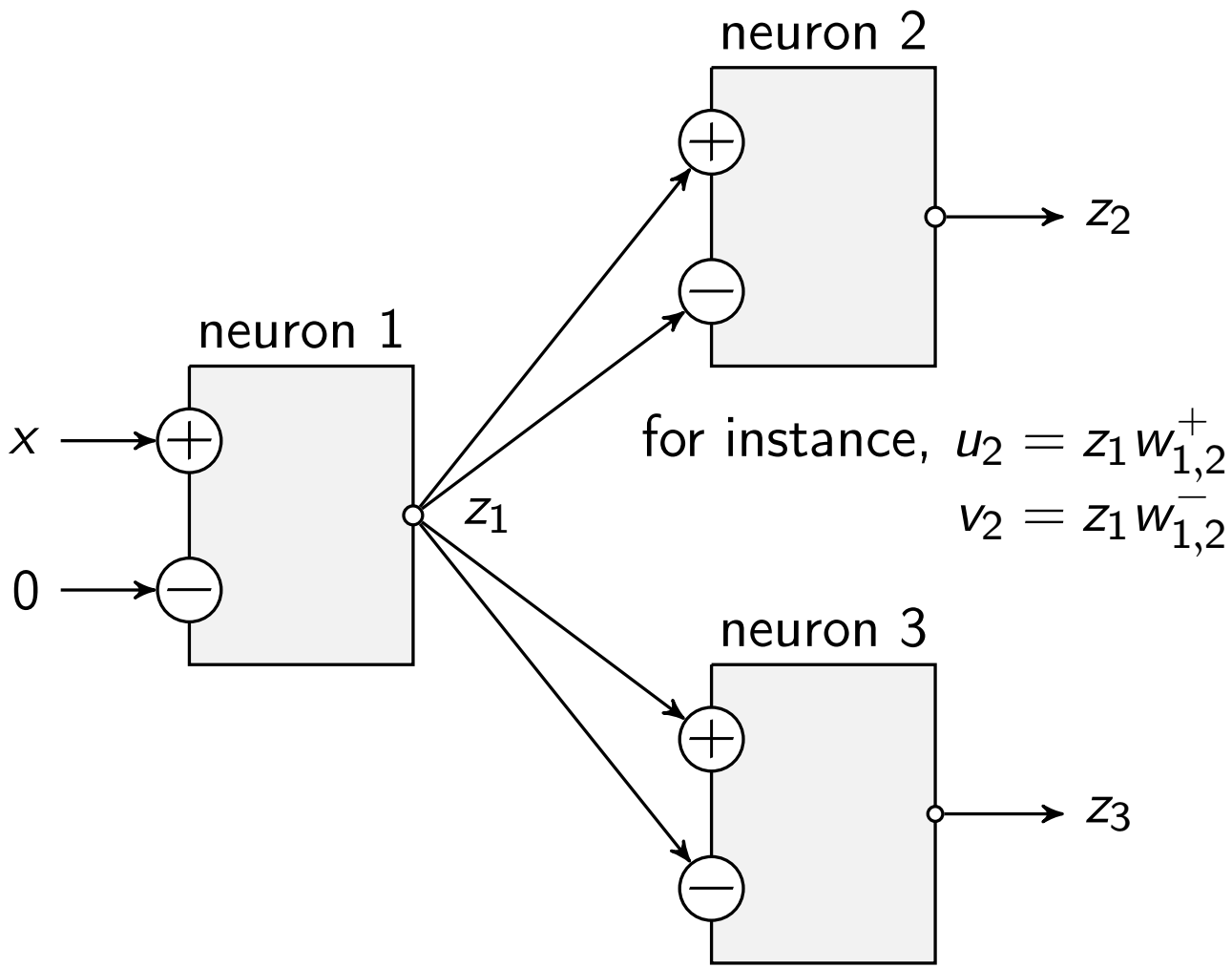
Random Neurons

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

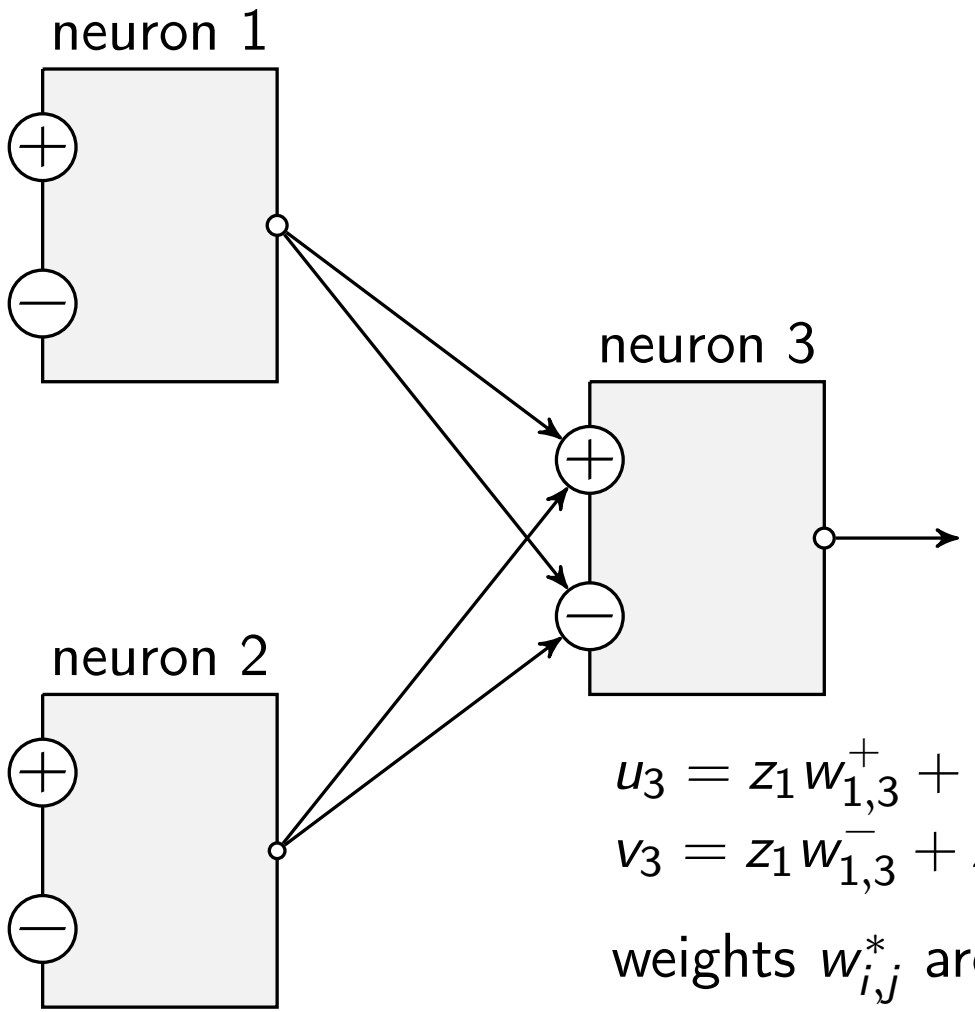


- We see $r > 0$, the *rate* of the neuron, as a parameter.
- There are several variants of this model. In the main and original one, we must use the function $z = \min(u/(r+v),\ 1)$.
- Inventor: E. Gelenbe, Imperial College, in the late 80s.
- Observe that there is nothing random here. The name comes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

A Random Neural Network (RNN) is a set of interconnected RNs. The connections are weighted by nonnegative reals denoted $w^+_{i,j}$ and $w^-_{i,j}$ for the weights of the connection *from i to j* arriving at the positive and negative ports respectively.



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

Restriction

- Because of the queuing origin (see below) 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 the theory of these queueing systems and the algorithms associated with.
- The same reason explains the use of $z = \min(u/(r+v),\, 1)$ instead of simply $z = u/(r+v)$.

Outline

Perceptual Quality

- Consider any kind of application or service centered on transporting audio and/or video signals over the Internet.
- The media can be sent one-way (e.g., a video streaming service) or two-ways (e.g., an IP-telephony application).
- For many reasons (compression techniques used, congestion in the network – leading to delays, or to losses, external perturbation agents such as interferences in some networking technologies, etc.) transmission can suffer from content degradation.
- Perceptual quality (PQ) refers to the (subjective) perception (the view, the feeling) the user has on the "value" of this media transmission system, on the quality of what she receives, on the impact of the degradations due to the transmission over the network.
- No formal definition.

Subjective testing

- Subjective testing is the standard way of measuring perceived quality.
- A panel of appropriately chosen human observers is built, and a set of media sequences is shown to the panel members.
- Following specific rules (absolute rating, comparisons...), sometimes discussing with a robot, the observers say how they perceive the quality of the sequences.
- Then, the results are statistically quantified and filtered.

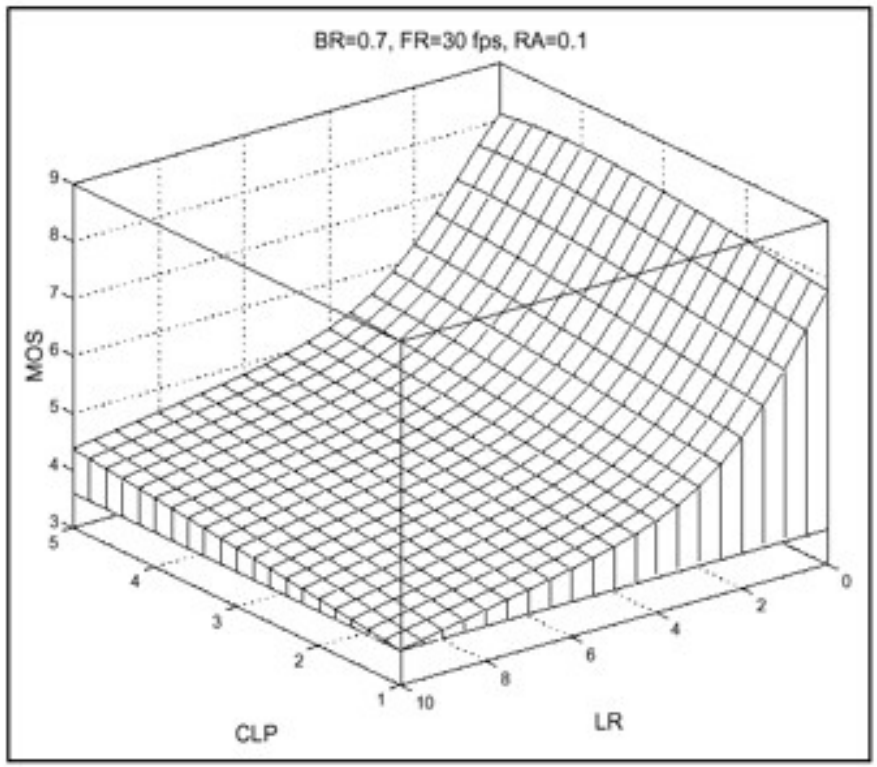
Our approach

- Our approach is called PSQA: Pseudo-Subjective Quality Assessment[1]. In a nutshell:
  - select a set of measurable (in real time) parameters assumed to have an impact on the PQ and characterizing (i) the network and (ii) the channel states;
    examples: for (i) packet loss rate, frame loss rate, delay, ...
    for (ii), bandwidth, shift of a protecting FEC, frames per sec, rate of playout interruptions, ...
  - define their joint living space, and sample a set of points in that space mixing Monte Carlo and Quasi-Monte Carlo techniques;
  - for each sample s, build a sequence (say a video one) where the chosen parameters have the values in s, and tag it in a subjective testing session, computing MOS, or the whole distribution (quantiles...);
  - learn the mapping from the parameters to the PQ.
- For the learning phase, we used those RNNs.

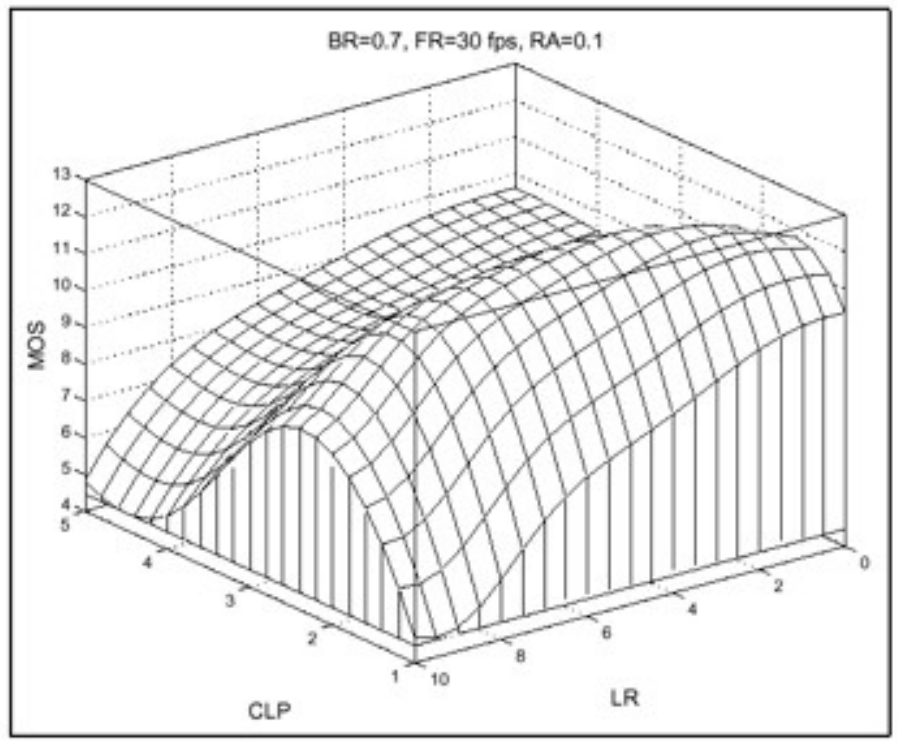
[1] For an external view, see "Measurement of Quality of Experience of Video-on-Demand Services: A Survey", Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi, IEEE Comm. Surveys & Tuturials, 18, 1, 1st Q 2016.

Why using RNNs in our PSQA project?

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

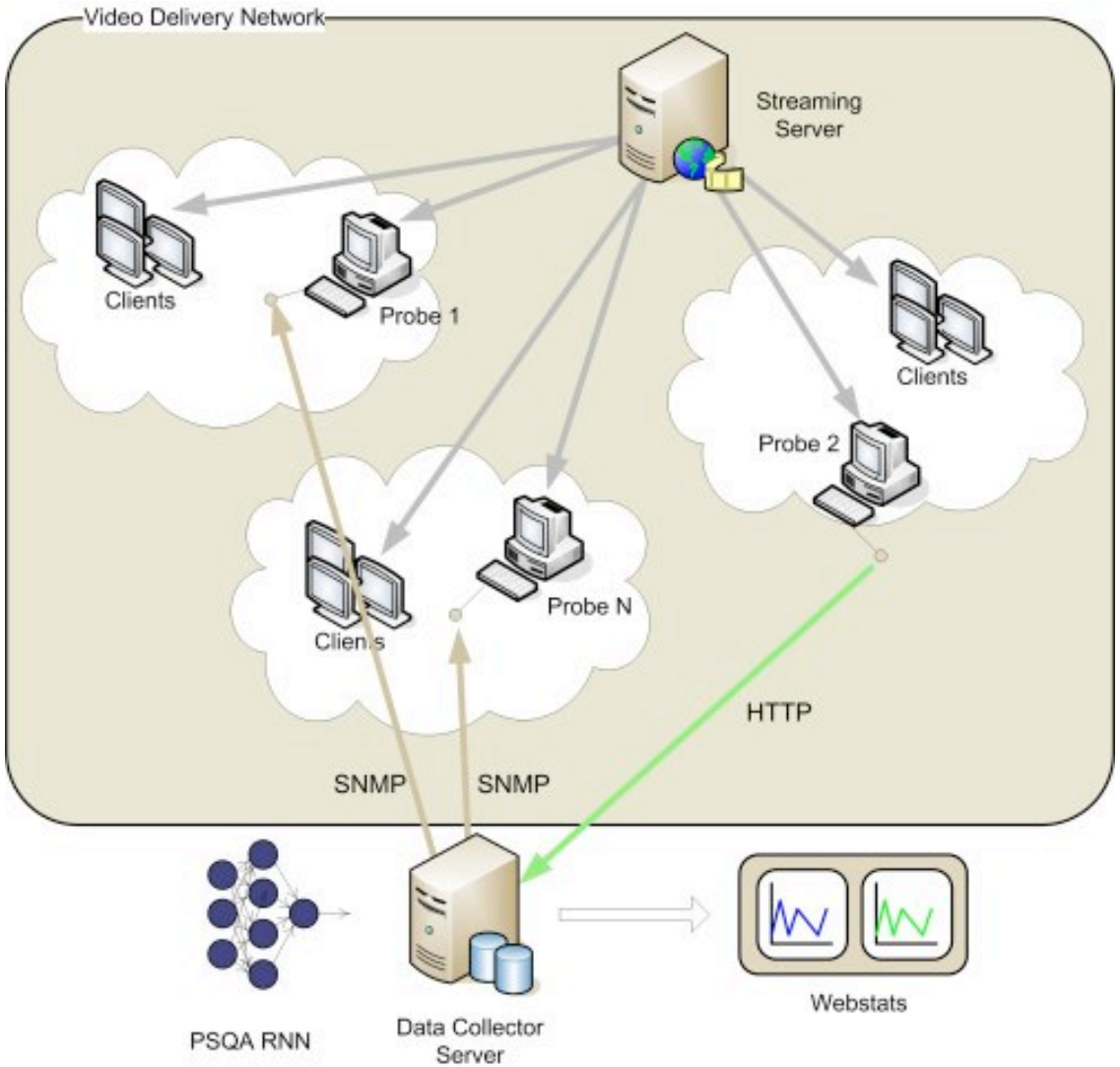


(a) Correctly trainined



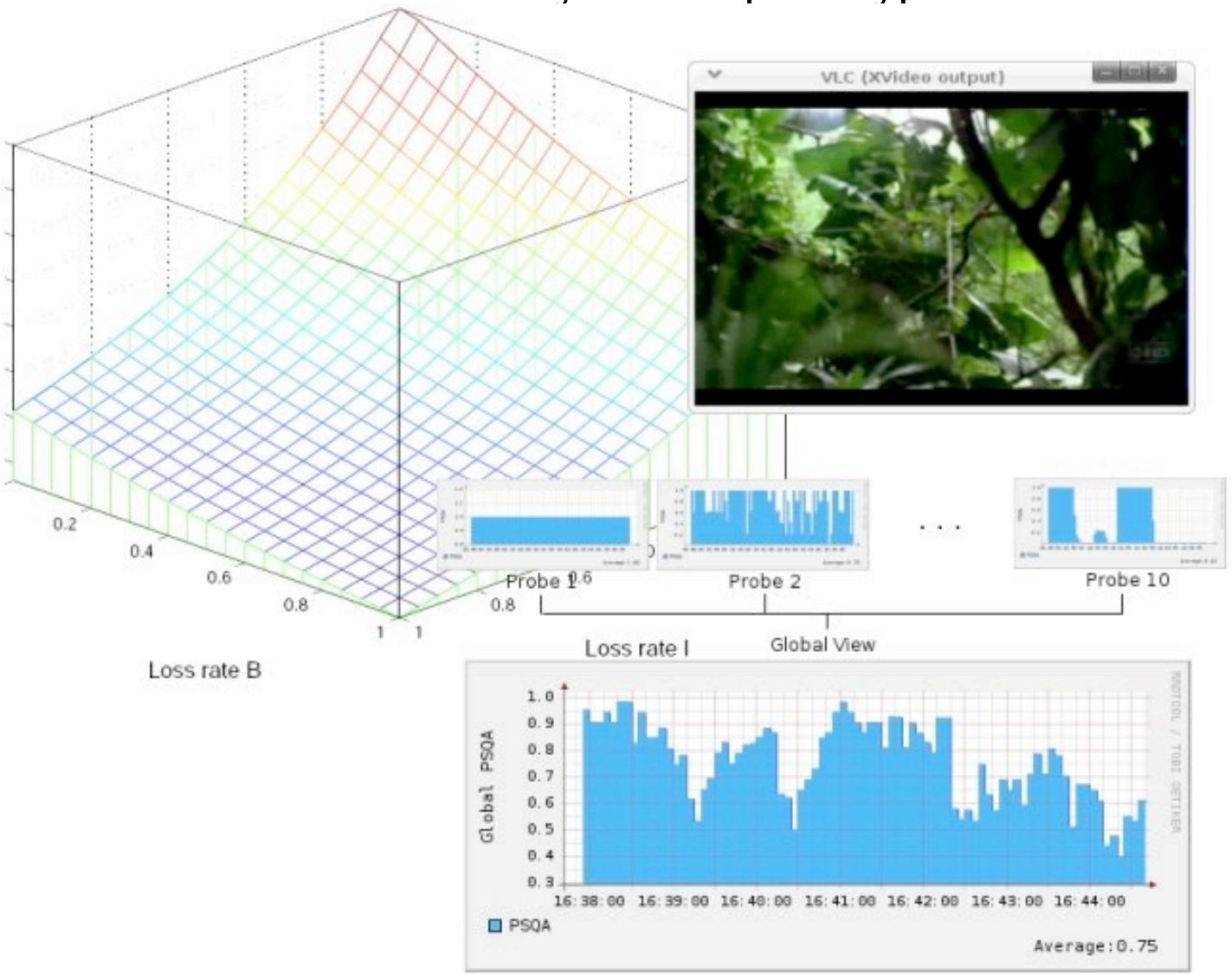
(b) Example of an over-trained ANN

Auditing the network with PSQA



Auditing the network with PSQA (cont.)

A "graphical" view of an industrially tested prototype.



Outline

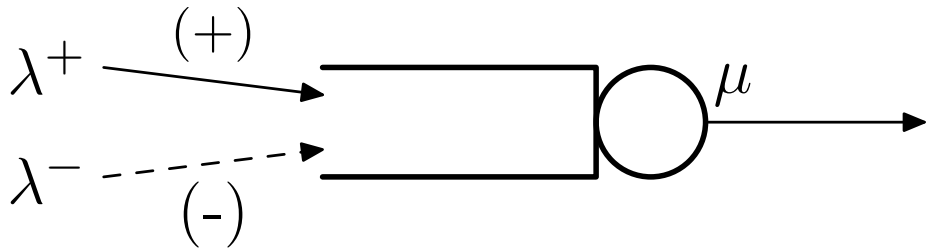
Origin of the model

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 $(\pi_n)_{n\geq 0}$ 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, ρ, satisfies

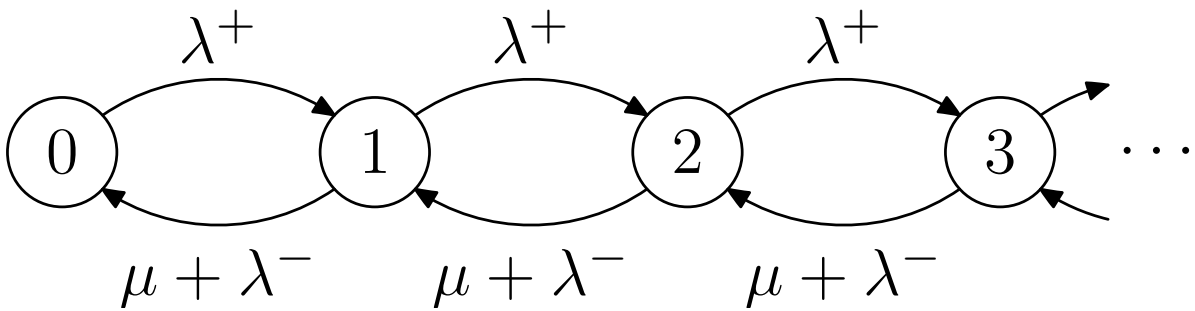
$$\rho = \min\left(\frac{\lambda}{\mu},\, 1\right).$$

A G-queue



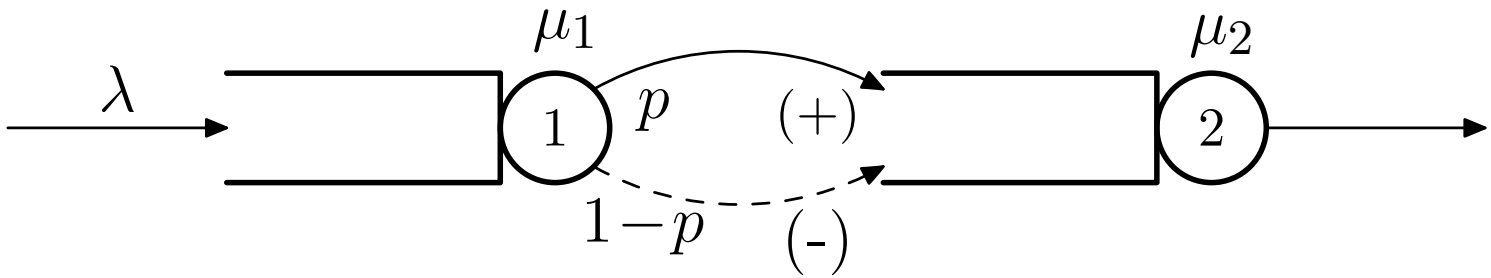
A basic G-queue: positive (standard) customers arrive with rate λ^+; negative ones arrive with rate λ^-; both arrival processes are Poisson; the service rate is μ. To understand the semantics, see the graph below.

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



Markov graph associated with the basic G-queue.

From G-queues to G-networks



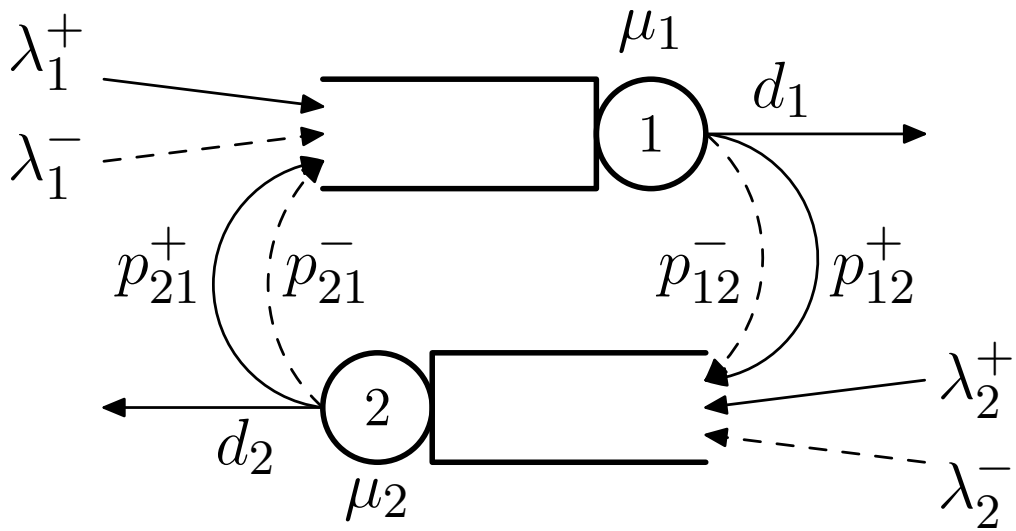
A tandem: first queue is an M/M/1; when leaving the first node, with probability p customers go to a second queue as positive ones, and with probability $1-p$ as negative signals; service rate at queue i is μ_i, $i = 1, 2$.

Here, theory says, for instance, that stability happens $\Longleftrightarrow$ $\lambda < \mu_1, \mu_2$, and in that case, if X_i is the number of units in i, $i = 1, 2$, we have

$$\Pr(\text{ at infinity, } X_1 = j, X_2 = k\ ) = (1-\rho_1)\,\rho_1^j (1-\rho_2)\rho_2^k.$$

For instance, $\Pr(\text{ in equilibrium, both queues aren't empty }) = \rho_1\rho_2$.

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.

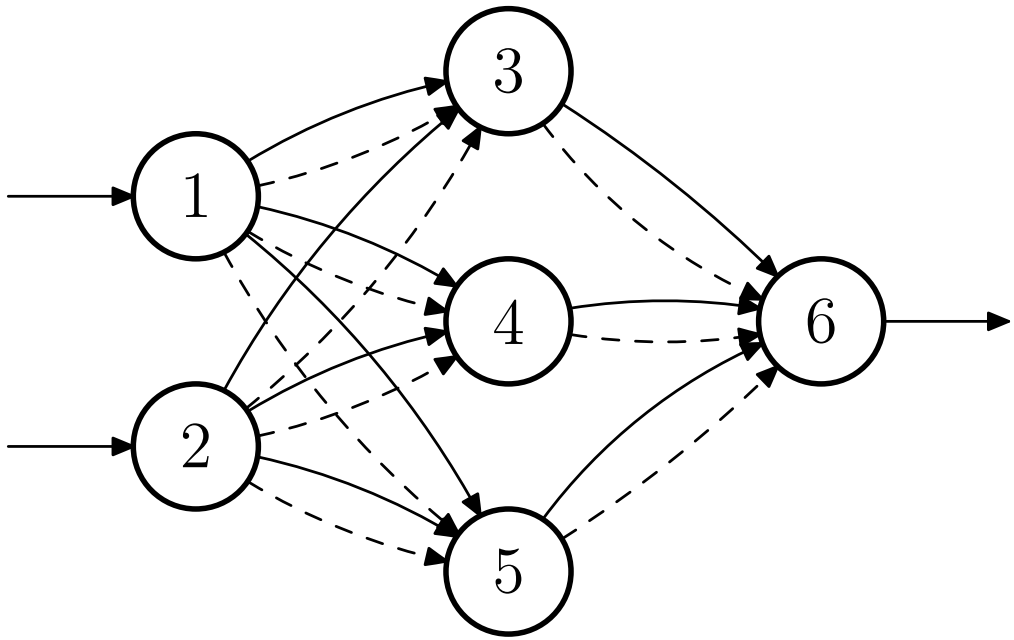
G-queues = random neurons

- Both objects are mathematically the same. Only the vocabulary and the usage change:
  - queue = neuron,
  - customers = signals,
  - backlog of the queue = potential of the neuron,
  - service rate of the queue = parameter of the neuron,
  - queue isn't empty = neuron fires, etc.
- In a G-network, when a customer leaves queue i, it goes to queue j as a positive customer with some probability $p^+_{i,j}$ and as a negative one with probability $p^-_{i,j}$. If μ_i is the service rate at queue i, then we denote $w^+_{i,j} = \mu_i p^+_{i,j}$ and $w^-_{i,j} = \mu_i p^-_{i,j}$.

G-queues = random neurons

- When used as a neural network, the inputs of the network are the rates of the flows of external customers; the neurons' rates are fixed; the outputs are the loads of the neurons sending customers to outside (if T_i is the throughput from queue i to outside, then $T_i = \rho_i \mu_i p^+_{i,0}$ where 0 denotes the environment).
- The $w^*_{i,j}$ are the weights of the neural tool.
- Observe that 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 $r_i p^-_{i,j} =: w^-_{i,j}$. So, in this model, information is stored/coded in the frequencies of the spikes travelling through the network.

A 3-layer structure

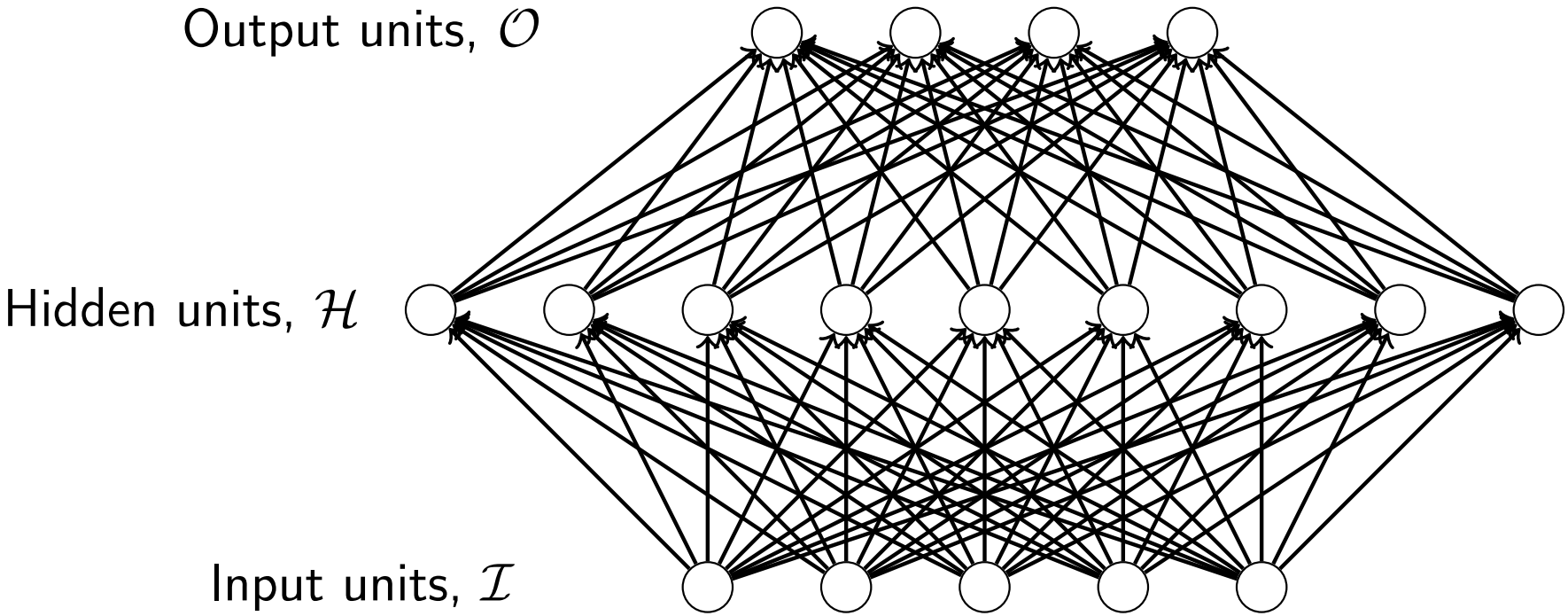


The graph of a 3-layer RNN of the (2,3,1) type (as before, dashed arrows correspond to flows of negative customers/signals). Observe that there is no negative flow coming from outside (the usual way of configuring this tool).

On the weights of a RNN

- Since for any neuron i in the network, we must have $\sum_j (p^+_{i,j} + p^-_{i,j}) = 1$ (adding a queue/neuron 0 representing the network's environment), we have $\sum_j (w^+_{i,j} + w^-_{i,j}) = r_i$
- Also, in stability conditions, the load ρ_i at queue i is < 1, and in the unstable case, $\rho_i = 1$. This explains the expression $z = \min(x/(r + y), 1)$.

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, again, allowing some specific mathematical treatments.

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).
- They have been shown efficient tools for predicting the future of time series.

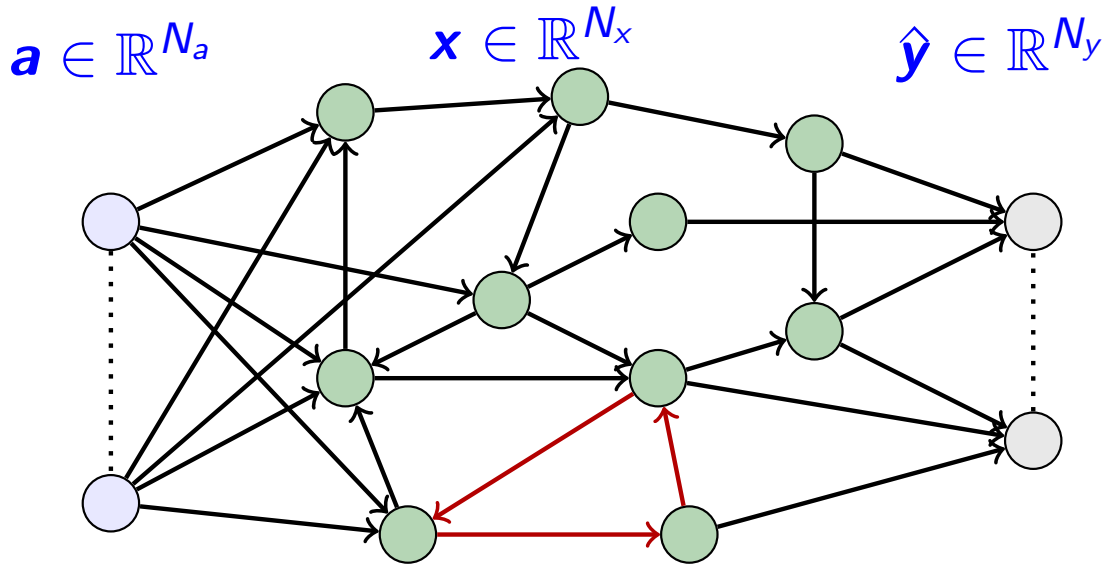
Echo State Networks (ESNs)

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 $u = N_a + 1, \ldots, N_a + N_x$,

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

- The input space 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.

Initial motivation

- Time series prediction is an interesting general target.
- We also have a particular interest in good prediction tools for specific network metrics, related to the PSQA project.
- PSQA is a tool for providing a quantitative assessment of PQ in real time. For a new generation of tools, we want to predict the PQ in a close future (a few minutes), for network controlling purposes.

Outline

I refer only to the ML side

- We are exploring the use/interest of Random Neurons in deep architectures. Some encouraging preliminary results available (NIPS 2017 DL workshop, December 9th, 2017).
- Analysis of several theoretical questions around Reservoir Computing with our ESQN model (mainly around convergence, stability, etc.).
- In PSQA, the labels come from subjective tests (that is, from panel of human subjects). We have a project whose goal is the elimination of the (main) 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 of huge amounts of sequences,
  - and then, to the exploration of the use of specific deep architectures for the learning phase (not necessarily related to RNNs).

PSQA 2.0

- We are exploring the transfert of PSQA to industry. It has been already tested by many companies in research projects and by a couple of operators, successfully.
- We are now looking for companies working in measuring or related areas, to implement an operational version of our experimental Network Perceptual Quality Analysis System.
- We intend also to add some other services to the initial measuring tools such as
  - the predictor of PQ previously described,
  - the computation of sensitivities (how important is this factor, say the Packet Loss Rate, to Perceptual Quality?),
  - an "inverter": given a quality level Q_0, how far are our selected factors from the area corresponding to a quality level $< Q_0$?

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 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.