



HAL
open science

Using Machine Learning in communication network research

Gerardo Rubino

► **To cite this version:**

Gerardo Rubino. Using Machine Learning in communication network research. LA-CCI 2022 - 8th IEEE Latin American Conference on Computational Intelligence, Nov 2022, Montevideo, Uruguay. pp.1-62. hal-03902768

HAL Id: hal-03902768

<https://inria.hal.science/hal-03902768v1>

Submitted on 16 Dec 2022

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.

Using Machine Learning in communication network research

G. Rubino

INRIA Rennes, France

Nov 2022

Context

- Using Machine Learning in communication network research, in a specific INRIA Team (until this year Dionysos, then Ermine).
- Our main research topics concern networking problems, where we use several techniques from the Machine Learning area.
- In Machine Learning, we use methods coming from the main Machine Learning families, from Supervised and Non Supervised Learning, and from Reinforcement Learning.
- Some keywords for our team: wireless networks, optical networks, 5G, 6G and 6G+, SDNs, NFV, IoT, access control in networking, networks economics (in particular, neutrality issues), performance evaluation, dependability evaluation, Quality of Experience, plus different mathematical and Machine Learning tools.

Main topics

- Automatic measuring of the Quality of Experience
using mainly Supervised Learning, today also Unsupervised Learning
- Time Series forecasting
using Supervised Learning with Reservoir Computing architectures
- Network tomography
using Supervised Learning
- (Different service placement problems in 5G, 6G and beyond
using Reinforcement Learning)

Outline

1 — Measuring Quality of Experience (QoE)

1.1 — Random Neural Networks

1.2 — Queuing origins of Random Neural Networks

1.3 — Our resulting PSQA tool for QoE measuring

2 — Time Series forecasting

3 — Network tomography

Outline

1 — Measuring Quality of Experience (QoE)

1.1 — Random Neural Networks

1.2 — Queuing origins of Random Neural Networks

1.3 — Our resulting PSQA tool for QoE measuring

2 — Time Series forecasting

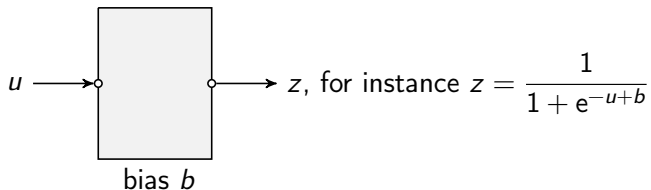
3 — Network tomography

Content

- The Machine Learning tool used: Random Neural Networks
- The queueing origins of Random Neural Networks
- Our resulting PSQA tool for QoE measuring

Classic artificial neurons

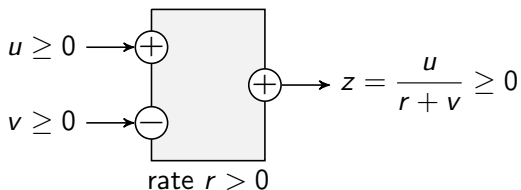
- A “classic artificial neuron” can be identified with its activation function, that is, can be seen as a real function of a real variable.



- We consider b as a parameter of the function/neuron.
- There are many different activation functions used in practice: $z = \tanh(u - b)$, $z = 1(u \geq b)$, $z = (x - b)^+$ (ReLU), ...
- Then, we build Neural Network by interconnecting these functions using real weights...

Random Neurons

- A **Random Neuron** is a parametric positive real function of two real positive variables or “ports”, one called the “positive port”, the other one being the “negative port”.

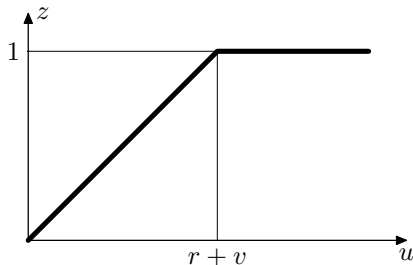


- We see $r > 0$, the *rate* of the neuron, as a parameter.
- There are variants of this model. The main one is $z = \min(u/(r + v), 1)$ (the original one).
- Inventor: Erol Gelenbe, Imperial College, in the late 80s.
- Observe that there is nothing random here.

Output z as a function of u

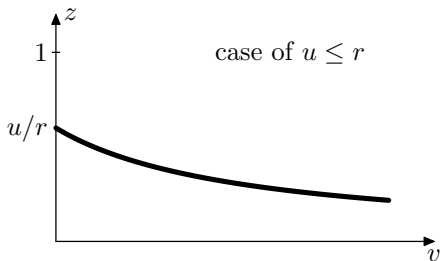
We illustrate here the obtained behavior in the case of

$$z = \min(u/(r + v), 1).$$



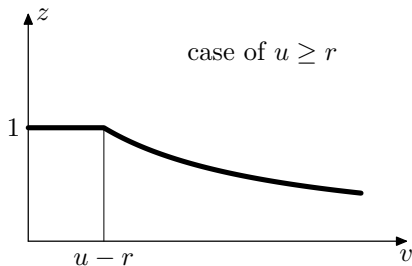
Output z as a function of u .

Output z as a function of v when $u \leq r$



Output z as a function of v when $u \leq r$
(in this case the neuron is not saturated).

Output z as a function of v when $u \geq r$



Output z as a function of v when $u \geq r$; see that as far as $v \leq u - r$, the neuron is saturated.

Derivatives

- $\frac{\partial z}{\partial u} = \begin{cases} 1 & u \leq r + v, \\ 0 & u > r + v; \end{cases}$
- if $u \leq r$, $\frac{\partial z}{\partial v} = -\frac{u}{(r + v)^2}$;
- if $u > r$, $\frac{\partial z}{\partial v} = \begin{cases} 0 & v \leq u - r, \\ -\frac{u}{(r + v)^2} & v > u - r; \end{cases}$
- BOSSER CECI $\frac{\partial z}{\partial r} = \begin{cases} -\frac{u}{(r + v)^2} & u \leq r + v, \\ 0 & u > r + v; \end{cases}$.

Derivatives

- For learning and for other purposes, we need the derivatives of z w.r.t. u , v and also r .

- W.r.t. u : if $u < r + v$, $\frac{\partial z}{\partial u} = 1$;

$$\text{if } u > r + v, \quad \frac{\partial z}{\partial u} = 0.$$

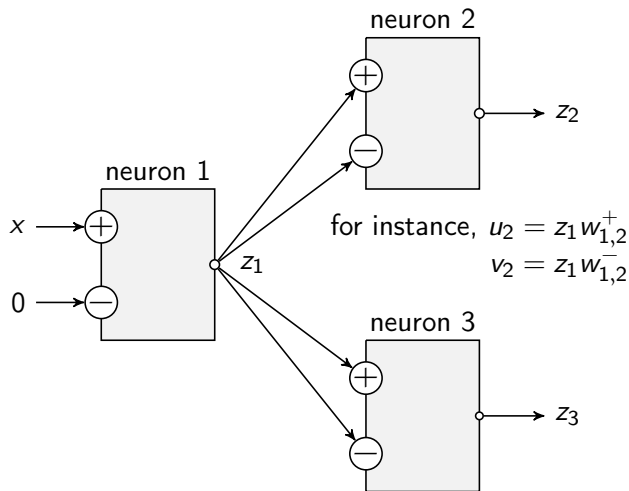
- W.r.t. v : when $u < r$, $\frac{\partial z}{\partial v} = -\frac{u}{(r + v)^2}$;

$$\text{if } u > r, \text{ then for } v < u - r, \quad \frac{\partial z}{\partial v} = 0,$$

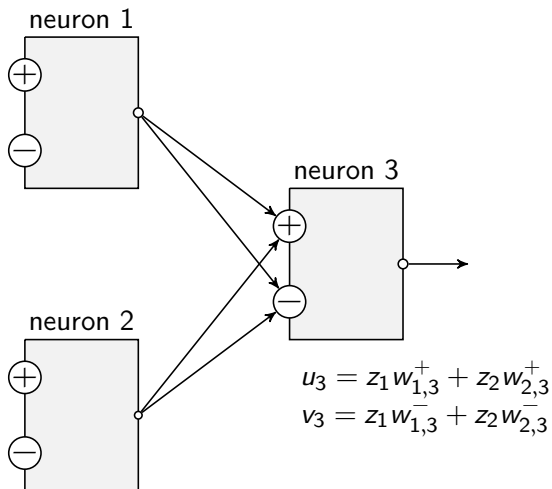
$$\text{and when } u > r, \quad \frac{\partial z}{\partial v} = -\frac{u}{(r + v)^2}.$$

- W.r.t. r : if $u < v$, $\frac{\partial z}{\partial r} = -\frac{u}{(r+v)^2}$;
if $u > v$, then for $r < u - v$, $\frac{\partial z}{\partial r} = 0$,
and if $r > u - v$, $\frac{\partial z}{\partial r} = -\frac{u}{(r+v)^2}$.

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



Junctions at input ports are additive:



Equations

- Consider a Random Neural Network (RNN).
- Let us take the most complex form of the activation function of neuron i

$$z_i = \frac{u_i}{r_i + v_i} \wedge 1.$$

for every neuron i .

- Assume that the network receives the input X_i on the positive port of neuron i , and y_i on its negative port. We then have

$$z_i = \frac{x_i + \sum_j z_j w_{j,i}^+}{r_i + y_i + \sum_k z_k w_{k,i}^-} \wedge 1.$$

- This constitute a non linear system of equations in the z_i s.

Possible restriction

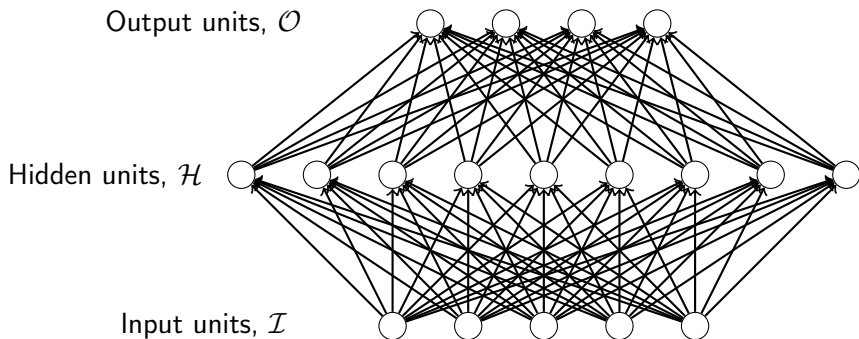
- This model comes from queuing theory.
- Because of the queuing origin of the RNN tool, in order to respect the standard interpretation of the model, we must have

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

- 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 $z = \min(u/(r + v), 1)r$ instead of $z = u/(r + v)$ (see below).

The general 3-layer Random Neural Network

A very used particular architecture:



- Assume the negative ports of input neurons aren't used. Assume a single output neuron, so, a scalar network output. Assume stability.
- 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.

Origin of the model

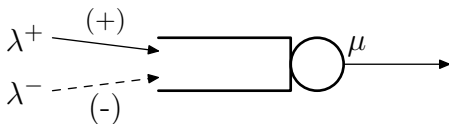


Consider the M/M/1 model, with **arrival rate** λ and **service rate** μ . The queue is stable iff $\lambda < \mu$; in that case, the steady state distribution (π_n) is given by $\pi_n = (1 - \rho)\rho^n$, $n \in \mathbb{N}$, where the number $\rho = \lambda/\mu = 1 - \pi_0$ is the utilization factor,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

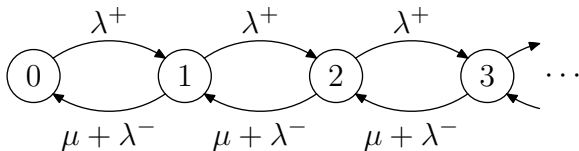
- $\rho = \Pr(\text{ in equilibrium, the system is busy })$;
if $\lambda \geq \mu$, then we have that
 $\Pr(\text{ there exists } t < \infty \text{ such that after } t, \text{ system is always busy }) = 1$.
- See then that the load of the queue at infinity, ρ , satisfies

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

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, associated with this model.**



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

Important

- A Random Neuron and a G-queue are the same mathematical object with two different names.

Proof of the relationship between G-queues and Random Neurons

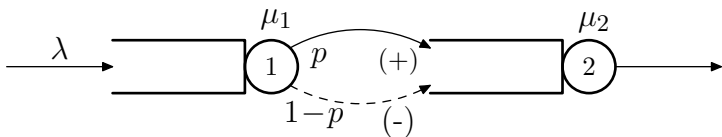
- Assume stability and equilibrium.
- Let T_i^+ be the mean input flow of positive customers in equilibrium, T_i^- that of negative ones, and T_o^+ and T_o^- the corresponding mean output flows.
- We have $T_i^+ = \lambda^+$ and $T_i^- = \lambda^-$. Then, using the mean flow conservation law for positive customers $T_i^+ = T_o^+$, we have

$$\lambda^+ = \mu\rho + \lambda^-\rho,$$

with ρ the server occupation probability (always in equilibrium), also called the system's load, leading to

$$\rho = \frac{\lambda^+}{\mu + \lambda^-}.$$

From G-queues to G-networks

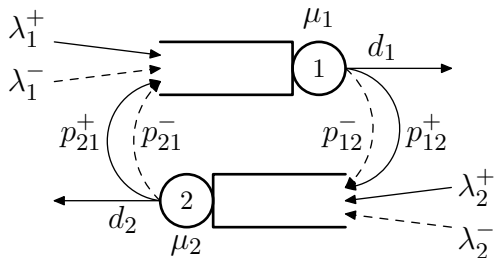


A tandem: first queue is an M/M/1; when leaving the first node, with probability p customers go to a second queue as positive ones, and with probability $1 - p$ as negative signals; service rate at queue i is μ_i , $i = 1, 2$.

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 $j, k = 0, 1$,

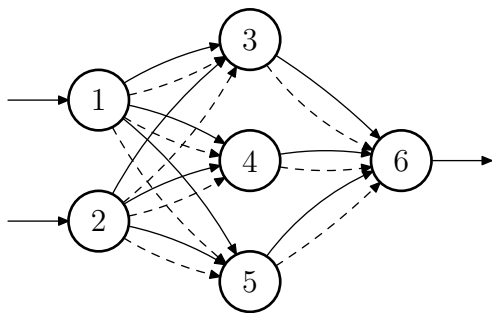
$$\Pr(\text{ at infinity, } X_1 = j, X_2 = k) = \rho_1^j \rho_2^k.$$

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 customer 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 $r_i p_{i,j}^- =: w_{i,j}^-$.
- So, the weights here (the things that *learn*) are the mean speed at which signals between neurons move. This actually also hold for real (“humid”) neurons.
- 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 deduce that $\sum_j (w_{i,j}^+ + w_{i,j}^-) = r_i$

Important

- The simplicity of the RNN model makes that gradient-based procedures are relatively easy to deploy, or to manipulate.
- For instance, learning can be strongly accelerated by adapting Quasi-Newton methods to the rational functions that appear here.
- In the past we explored this aspect of the RNNs.

Some current projects at INRIA about Random Neural Networks

Currently, we are working on the development of a tool implementing these RNN models, and adding some new possibilities, exploiting their nice mathematical properties:

- **sensitivity analysis**: basically, $\partial y_j / \partial x_i$, where x_i is the i th input and y_j is the j th output; observe that the outputs, the sensitivities, the gradient of the output, and the standard costs functions, are all rational functions (w.r.t. the inputs, or to the weights)
- **inverting the RNN mapping when the output is a scalar**: basically, if we denote $y = v(\vec{x})$ where \vec{x} is the vector input to the network and y is the real output, then we look for the subset $v^{-1}(y \geq y_0)$, that is, the set of input leading to the fact that the output is at least y_0 .

Our main application: measuring QoE

- Considered area: any kind of **application or service** centered on transporting **audio and/or video signals over the Internet**.
- This produces most of Internet traffic.
- 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

- **Perceptual quality** 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.
- Perceptual Quality is the main component of the QoE of the considered applications/services.
- Two main characteristics:
 - It is, by definition, **a subjective concept**.
 - It **lacks a formal definition**.
- Critical problem (solved): how to **measure** it? How to **measure it automatically**? how to reach an accuracy level similar to that of panels of human users?

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...) the observers say how they perceive the quality of the sequences.
- In case of interactive applications, observers can work by pairs, or in some cases, interact with a robot.

Subjective testing

- The result is a table M where component $M_{h,\sigma}$ is the value given by human h to sequence σ .
- A **statistical procedure** is then followed to filter these results in order to **remove outliers**.
- At the end, if the set of surviving observers is \mathcal{S} , **the** quality of sequence σ , its *MOS* (Mean Opinion Score) value is

$$MOS(\sigma) = \frac{1}{|\mathcal{S}|} \sum_{h \in \mathcal{S}} M_{h,\sigma}.$$

- The procedure is surprisingly **robust**: with the same set of sequences but a different panel, the same subjective test gives results extremely close to the first one: if μ_σ and μ'_σ are the MOS values of sequence σ in both panels, the number $\sum_\sigma (\mu_\sigma - \mu'_\sigma)^2$ is pretty “small”.

Subjective testing (cont.)

- For each type of media, there are norms specifying how to perform the corresponding subjective test. A reference here is the production of the ITU (International Telecommunication Union).
- For instance, some norms ask users to watch video sequences potentially degraded by some noisy effect, and rate them from 1 to 5 according to the following table:

MOS score	description
5	degradation imperceptible
4	degradation perceptible but not annoying
3	degradation slightly annoying
2	degradation annoying
1	degradation very annoying

Objective testing

- Objective testing means evaluating the perceived quality automatically, without using any panel of users.
- To give an example, consider video sequences. Some objective testing methods (coming from the coding area) are based on the PSNR (Peak Signal To Noise Ratio) of two images A and B :
 - for instance, assume the two images are composed of M rows and N columns of 8-bits pixels (monochrome case).
 - The MSE (Mean Squared Error) between the two images is

$$\text{MSE}(A, B) = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N (A_{i,j} - B_{i,j})^2 \in [0..255^2].$$

- The PSNR between the two images is defined only if they are different ($\text{MSE} \neq 0$) and its value (in dB) is

$$\text{PSNR}_{\text{dB}}(A, B) = 10 \log_{10} \left(\frac{255^2}{\text{MSE}(A, B)} \right).$$

Problems with PSNR approach

Noise in the sky



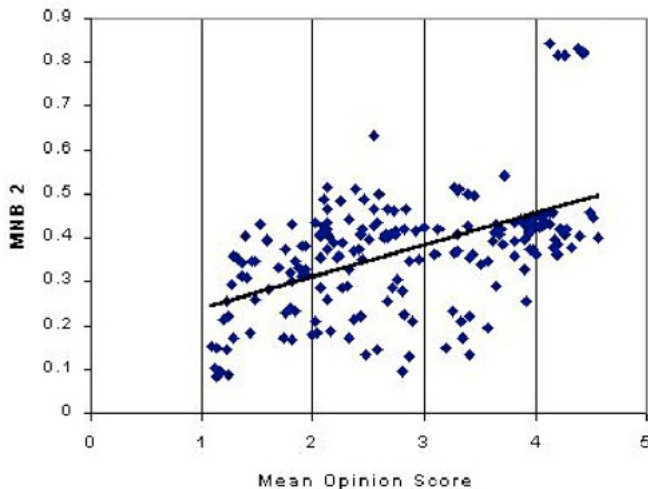
Problems with PSNR approach...

Same amount of noise in the grass



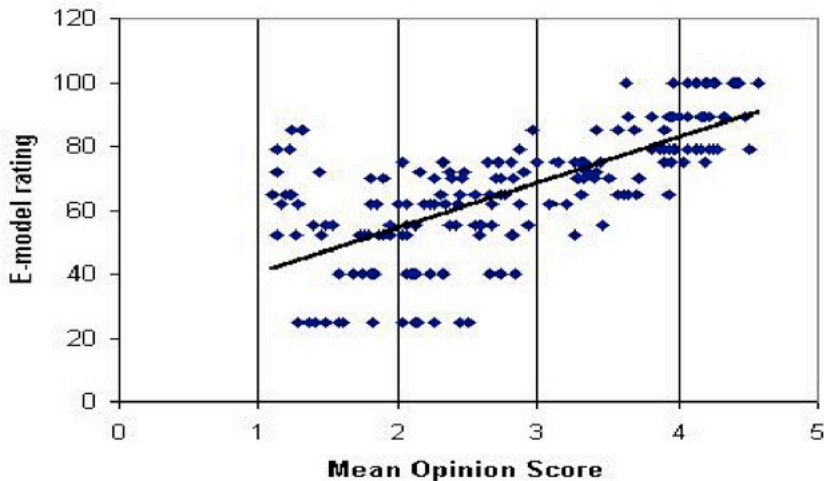
Example in audio

Bad correlation between an objective metric and subjective tests in audio.



Example in VoIP

Bad correlation between an objective metric and subjective tests in Voice over IP.



PSQA: Pseudo Subjective Quality Assessment

- In the Dionysos team we promote our solution called **PSQA** to all the initial problems and many more, for all types of media considered, and for both one-way and two-ways communications.
- PSQA is a metric with **no reference**, **automatic**, (so far, “optimally”) **accurate**, and it **works in real time** if necessary or useful.
- PSQA is **network-dependent** and **application-dependent**.
- It is a **parametric approach** (a “black-box” approach), mapping QoS parameters and source-based parameters into perceptual quality (into a MOS value).

PSQA (cont.)

- The mapping is based on supervised learning.
- We explored different tools for this supervised learning task, including the ToolBox of Matlab.
- The winner was the RNN model.

How PSQA works

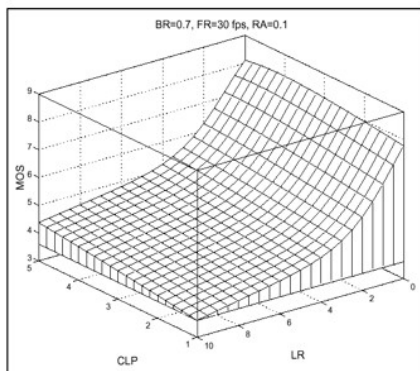
- First of all, we select
 - (i) **measurable QoS metrics** characterizing the state of the network (logically speaking, instantaneous measures), and **assumed, a priori, to have an impact on the Perceptual Quality**,
 - (ii) and **metrics related to the channel or to the source**, again, **expected to have impact on the PQ**.
- Example in video: (i) the instantaneous packet loss rate LR , (ii) the Bit Rate BR and the Frame Rate FR .
- Example in VoIP: (i) the instantaneous packet loss rate LR , the average size of a burst of loss packets $MLBS$, (ii) the Bit Rate BR and the Offset of the FEC (Forward Error Correction) $OFEC$.

- Let $\vec{x} = (x_1, \dots, x_n)$ denote the vector whose components are the n chosen metrics. We call it configuration.
- Configurations live in some product space $S_1 \times \dots \times S_n$.
- Our postulate: PQ depends only on \vec{x} (when “things go well”) and not on signal content.
- We select a few short signals (following standards) representative of the application or service target, $\sigma_1, \dots, \sigma_K$.
- We build a pretty small set of K configurations ($K = 100, 200, \dots$) by a mix of random sampling and quasi-Monte Carlo (or weak discrepancy sequences). Call them $\vec{\gamma}_1, \dots, \vec{\gamma}_K$.
- Last, we build a platform allowing to send signals σ_i through a simulated or deployed network where we can simultaneously control all components of the configurations.

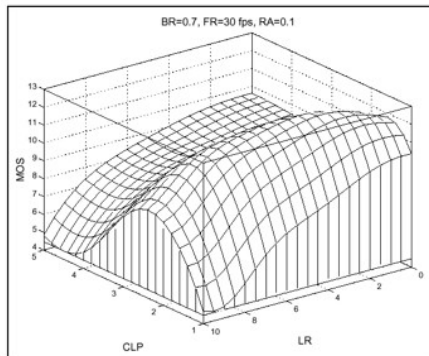
- Then we send different original σ_i using the platform, when the configuration is $\vec{\gamma}_j$ using one configuration at a time, and obtain a possibly degraded sequence σ'_j . We show it to a panel of humans and we obtain its PQs Q_j .
- We thus obtain a set of K sequences, with variable but known PQ (MOS values) coming from subjective testing sessions, and for each, we know which configuration was used to obtain it.
- Then, we use a RNN (a G-network), of the classical feedforward type with 3 layers, to learn the mapping from configurations to PQ (a mapping from $\text{QoS} \times \text{"channel state"}$ to MOS values). Data: the M pairs $(\vec{\gamma}_j, Q_j)$.
- In the example of video, we obtain a function $\nu(LR, BR, FR)$. In the VoIP example, we obtain a function $\nu(LR, MLBS, BR, OFEC)$.

Why using RNNs in PSQA project?

Because of the favorable comparison with standard software (at the time of the beginning, several years ago). An example: with the same data and the same neural network size, on the left, PSQA, and on the right, Matlab (an old version of the ToolBox on learning techniques).

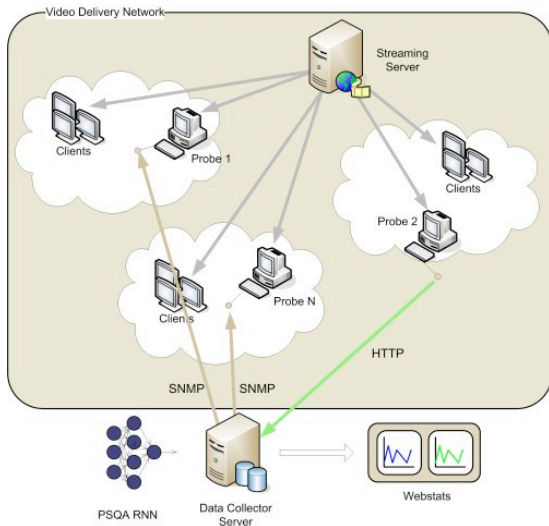


(a) Correctly trained



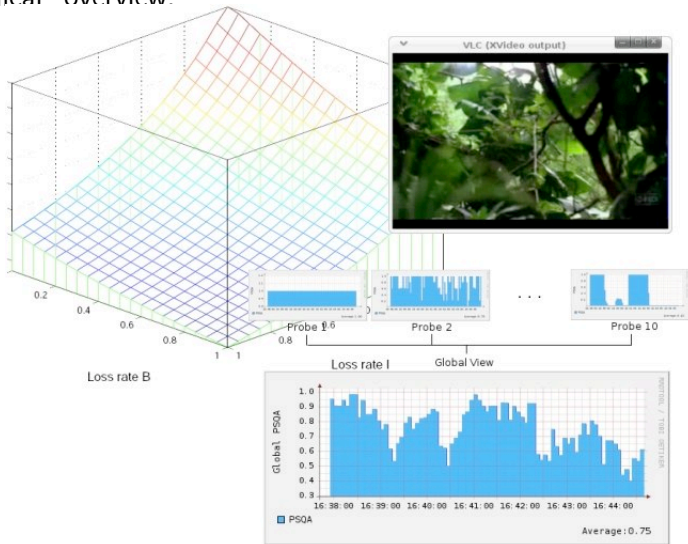
(b) Example of an over-trained ANN

Auditing the network with PSQA



Auditing the network with PSQA (cont.)

A “graphical” overview:



Current/future work

- Improving the PSQA technology to integrate a preliminary Phase 0 in the tool calibration (using Unsupervised Learning) by adding methodology to choose the sequences for training, taking into account the current transmission technologies.
- Integration of
 - sensitivity analysis: which input variables are the most important ones?
 - inverting the PSQA mapping: are we close to the border of the subspace of configurations where quality is good enough?

Some references

- About the RNN model and the PSQA tool:
“*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. See the references in the paper for the descriptions of the model by its creator.
- 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 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.

Outline

1 — Measuring Quality of Experience (QoE)

1.1 — Random Neural Networks

1.2 — Queuing origins of Random Neural Networks

1.3 — Our resulting PSQA tool for QoE measuring

2 — Time Series forecasting

3 — Network tomography

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).

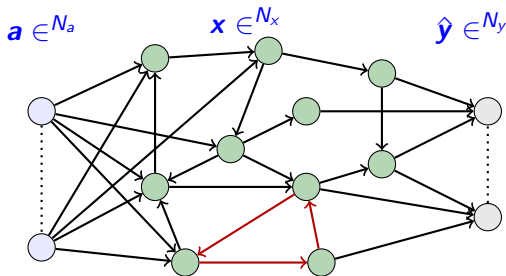
Echo State Networks (ESNs)

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*.
- the reservoir size is much larger than the input size.

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”.
Discrete time.
- Input at time t , $\mathbf{a}(t) = (a_1(t), \dots, a_{N_a}(t))$:
 $\rho_u(t) = a_u(t)/r_u$, (in general, we take $a_u(t) < r_u$), for $u \in 1..N_a$.
- For all reservoir units $u = N_a + 1, \dots, N_a + N_x$,

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

- The input space is then projected into a new “larger” space, where “separations” are easier.
- We compute (for instance) a linear regression from the projected space to the output space.
- Thus, the network output $\hat{\mathbf{y}}(t) = (\hat{y}_1(t), \dots, \hat{y}_m(t), \dots, \hat{y}_{N_b}(t))$ is computed using

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

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

- Main paper on ESQNs:
“*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.

Some illustrations

NARMA is a simulated dataset that is often analyzed in the RC literature.

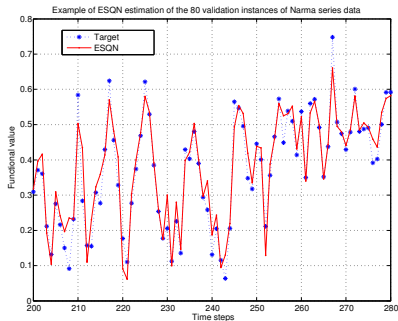


Figure: NARMA validation data set. The reservoir was randomly initialized and it had 80 units.

Some illustrations

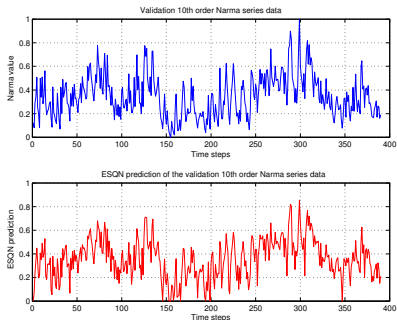


Figure: Fixed 10th order NARMA times series. Comparison between the original dataset (blue) and the values forecasted by ESNQ (red).

Some illustrations

Internet traffic prediction

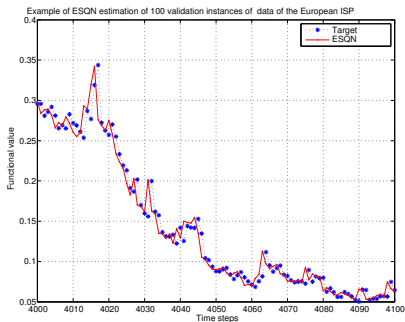


Figure: Normalized Internet traffic data from an European Internet Service Provider. The reservoir has 40 neurones.

Outline

1 — Measuring Quality of Experience (QoE)

1.1 — Random Neural Networks

1.2 — Queuing origins of Random Neural Networks

1.3 — Our resulting PSQA tool for QoE measuring

2 — Time Series forecasting

3 — Network tomography