CHAPTER 1

USING RANDOM NEURAL NETWORKS TO QUANTIFY THE QUALITY OF AUDIO AND VIDEO TRANSMISSIONS OVER THE INTERNET: THE PSQA APPROACH

Gerardo Rubino
INRIA / IRISA,
Campus de Beaulieu
35042 Rennes Cedex, France
E-mail: rubino@irisa.fr

Consider the problem of delivering multimedia streams on the Internet. In order to decide if the network is working correctly, one must be able to evaluate this delivered quality as perceived by the end user. This can be done by performing subjective tests on samples of the received flows, but this is expensive and not automatic. Being able to perform this task in an automatic way and efficiently, such that it can be done in real time, allows multiple applications, for instance in network control. Such a goal is achieved by our PSQA metric: Pseudo-Subjective Quality Assessment. It consists of training a Random Neural Network (RNN) to behave as a human observer and to deliver a numerical evaluation of quality, which must be, by construction, close to the average value a set of real human observers would give to the received streams. This chapter reviews our previous work on PSQA, and provides some new elements about this technology. The use of RNN is justified by its good performance as a statistical learning tool; moreover, the model has nice mathematical properties allowing using it in many applications and also obtaining interesting results when PSQA is coupled with standard modelling techniques.

1. Introduction

Consider an audio stream sent through an IP network, or a video one, or consider an interactive voice communication over the Internet. When quality varies, which is today very often the case (think of a wireless segment, for instance), it is useful to be able to quantify this quality evolution with time, in order to understand how the global communication system works, why the performance is as observed, how it can be improved, how the system

can be controlled to optimize quality, etc. Observe that we are here interested in quantifying the quality *as perceived by humans users*. This task is so useful when analyzing a networking application dealing with these kinds of flows that it has been standardized (some examples are [1] for audio flows and [2] for video ones). The corresponding area is called *subjective testing*. In a nutshell, it consists of taking a panel of human observers (say, around 10 to 20 persons sampled from the global population of observers, or a few experts, say 3 or 4, the choice between random observers or experts depending on the goals), and making them evaluate numerically the quality as they perceive it, by comparing an important number of sequences. The main problem with this approach is that it is expensive and, by definition, it is not an automatic process (and *a fortiori* not a real-time one).

If we try to perform this quality quantyfying process automatically, a first idea is to use *objective tests*. This basically refers to using techniques coming from coding, where the original sequence σ and the received one σ' are compared by computing some distance between them. A typical example of this is the PSNR metric (Peak Signal to Noise Ratio). It is well known that this approach does not correlate well with subjective ones, that is, with values coming from real human observers[a]. In [?] we specifically discuss about the bad behaviour of PSNR for video analysis.

In a series of papers [3,4,5] we developed a new methodology called PSQA (see next section). It allows to reach the objective of being able to measure quality *as perceived by the users*, automatically and accurately. This chapter describes the approach and some of its applications. Next section describes PSQA; Section 3 is a short presentation of Random Neural Networks, the mathematical tool used to implement the PSQA technology, with some new elements concerning sensitivity analysis; in Section 4 we explain how we couple PSQA with standard performance evaluation techniques, close to the work presented in [5]. Section 5 concludes the chapter.

2. The PSQA technology

PSQA stands for Pseudo-Subjective Quality Assessment, and it is a technology proposed first in [3] (under the name QQA: Quantitative Quality Assessent), in order to perform this quantyfing task automatically, in real-

[a]What we say is that objective measures are not good to quantify quality when considering flows that have been perturbated by travelling through a network where they can suffer from losses, delays, etc. They are of course very useful when analyzing *coding* effects.

time if necessary, and accurately. This last condition means that a PSQA metric must give to a stream a value close to the value an average human observer would give to it (or, in other cases, a pessimistic expert, or an optimistic one, depending on the goals of the study), as coming out of a subjective testing experiment. To describe how PSQA works, let us consider the following example, which a simplified version of the one used in [3]. We want to analyze a video streaming application, sending its flows from a source to a receiver. We proceed as follows:

(1) We first identify a set of parameters which, beforehand, we *think* have an (important) impact on the perceived quality. Observe that this is an *a priori* assumption. These parameters are of two classes: those related to the codec, which we assimilate to the source sending the stream, and those associated with the network transporting the data.
In our example, let us consider that the sending process can work at different bit rates BR, given in bps (bits per second), and at different frame rates, FR, in fps (frames per second). Assume that (i) the frames have all the same size in bits, (ii) there is no protection against losses (such as FEC – Forward Error Correction), (iii) the player at the receiver side can play a sequence with missing frames.
Concerning the network, we assume that the main problem is loosing packets since the receiver has a large buffer to absorb the possible variations in delay (jitter). For the loss process, given its importance we decide to characterize it by two parameters, the packet loss rate LR and the mean loss burst size MLBS, the latter giving an indication about how the losses are distributed in the flow.
(2) For each selected parameter we choose a range, corresponding to the system to be analyzed and the goals; we also choose a discrete set of values, which make the rest of the process simpler. For instance, in [3] it is considered that LR $\in \{0\,\%, 1\,\%, 2\,\%, 3\,\%, 5\,\%, 10\,\%\}$. Each combination of values for these parameters is called a *configuration* of the system. The total number of configurations can be high (in our previous applications of PSQA we always had spaces with thousands of points).
(3) We perform a selection of M configurations among the set of all possible ones. Typically, for 4 to 6 parameters we used M around 100 to 150. This selection process is made by a merge of two procedures: (i) different configurations are built by randomly choosing the parameters' values in the given space and (ii) several configurations are choosen by *covering*

in some way the extremities of the parameters' ranges (see [3]).
We randomly separate the set of M selected configurations into two subsets $\mathcal{C} = \{\gamma_1, \cdots, \gamma_K\}$ and $\mathcal{C}' = \{\gamma'_1, \cdots, \gamma'_{K'}\}$. We thus have $K + K' = M$ (for instance, in our applications, if $M \approx 100$ we will take, say, $K \approx 80$, or 85, and $K' \approx 20$, or 15. Set $\mathcal{C}$ will be used to *learn* and set $\mathcal{C}'$ to *validate* (see step (7)).

(4) We build a plat-form allowing (i) to send a video sequence through an IP connection, (ii) to control the set of parameters chosen in the first step; in the example, we must be able to choose *simultaneously* any values of the four parameters BR, FR, LR, MLBS.

(5) We choose a sequence σ[b] representative of the population of sequences to be considered; according to the norms used in video subjective testing, σ must be about 10 sec length. We send σ exactly M times from sender to receiver, using the platform, and each time using the values in each of the selected configurations (sets $\mathcal{C}$ and $\mathcal{C}'$). We obtain two sets of *distorted copies* of σ: the copy corresponding to configuration γ_i, $1 \leq i \leq K$ (resp. γ'_j, $1 \leq j \leq K'$) is denoted by σ_i (resp. by σ'_j).

(6) We perform a subjective testing experiment using the M sequences $\sigma_1, \cdots, \sigma_K, \sigma'_1, \cdots, \sigma'_{K'}$. This consists of selecting a panel of human observers and asking them to numerically evaluate the quality of the M sequences *as they perceive them*. For this purpose an appropriate norm must be followed. In our video experiments, we used [2].
Each sequence will thus receive a value usually called MOS (Mean Opinion Score). We denote the MOS of sequence σ_i (resp. of σ'_j) by Q_i (resp. by Q'_j). In more detail, assume there are R observers (typically, $R \approx 20$ if the observers are random elements of the population, or $R \approx 4$ if they are *experts*) and that observer r gives, at the end of the experiment, value q_{ri} to sequence σ_i (and value q'_{rj} to sequence σ'_j). Then, we must perform a statistical test to detect bad observers in the panel (case of random observers); in words, a bad observer is one that does not (statistically) agree with the majority, see [3]. Assume (after a re-ordering of the observers' indexes) that observers $R' + 1, \cdots, R$ are bad ones. Then, their scores are taken out of the set, and the sequences receive as MOS the average of the values given to it by the good observers; that is, $Q_i = \sum_{r=1}^{R'} q_{ri}/R'$ (and $Q'_i = \sum_{r=1}^{R'} q'_{ri}/R'$).

(7) We now identify configuration γ_i with quality value Q_i (and γ'_j with

[b] Actually, we must do the whole process with several different sequences; we just present the technique for one of them.

Q'_j), and look for a real function $\nu()$ of 4 variables associated with the four selected parameters, such that for any set of values in the set $\mathcal{C}$ the function returns a number close to the associated MOS value. That is, for any $\gamma_i \in \mathcal{C}$, $\nu(\gamma_i) \approx Q_i$. This is the *learning* phase. It can be done with different tools. We tried standard Artificial Neural Networks (ANN), Bayesian Networks and RNN. As illustrated in [3], RNN performed by far the best, and that is the main reason why we used that tool.

Remark: before the learning phase, it is more comfortable to *scale* the input variables by dividing them by their maximal possible value (recall that in the second step we associate a range with each variable). This way, all input variables are in [0..1]; the same is done with the output, which corresponds to the normalized quality value.

(8) After having found the $\nu()$ function, we must go through the *validation* phase consisting of testing its value on the set of configurations in $\mathcal{C}'$. If for $\gamma'_i \in \mathcal{C}'$ it is $\nu(\gamma'_i) \approx Q'_i$, then $\nu()$ is validated and the process is finished.

Of course, if this is not the case, something was wrong in the previous process (not enough data, bad sampling of configurations, etc.).

(9) Using the evaluator consists now of measuring BR, FR, LR and MLBS, for instance at the receiver of a communication, and calling $\nu()$ with the measured values as input. This can be done in real time since the inputs can be collected in real time, and the evaluation of $\nu()$ is not expensive.

The next section briefly explains what is a RNN and how it is used in learning (phase (7)).

3. The Random Neural Networks tool

We first present the mathematical RNN model and the particular case we use in the application presented in this chapter. Then, some elements about the use of the model as a learning tool are given. To have an idea about the multiple applications of this tool that have been already published, see [10].

3.1. *G-networks*

A Random Neural Network is a queueing network (also called a G-network), invented by E. Gelenbe in a series of papers [7,8,9] by merging concepts from neural networks and queueing theory. Its nodes are called queues or neurons,

depending on the context, but they are the same mathematical object. The customers or units of the queueing network are called "signals" at the neural side. In this chapter, we will use both terminologies, the neural one and the concepts from queueing models, depending on the specific topic we are speaking about.

The set of nodes in the network is $\mathcal{N} = \{1, \cdots, N\}$. The queueing network receives two types of customers from outside, called *positive* and *negative*. Neuron i receives positive units from outside with rate $\lambda_i^+ \geq 0$ and negative units from outside with rate $\lambda_i^- \geq 0$. Both arrival processes are Poisson. At least one neuron receives positive units from outside; that is, $\sum_{i\in\mathcal{N}} \lambda_i^+ > 0$.

Nodes are FIFO queues (this is not essential but it simplifies the presentation) with a single exponential server and infinite room. The service rate at neuron i is $\mu_i > 0$. Positive units behave as usual customers in queuing networks: they arrive at the node and wait in the queue until the server is available, then get service and are sent to another queue or to outside. When a negative customer arrives at a queue, it destroys itself instantaneously, and if the queue was not empty, the last customer in it is also destroyed. Moving from a queue to another or to outside is an instantaneous process, as usual.

Observe that the previous description means that negative units can not be observed, only their effects can; the network never has negative customers in it. Denote by $X_i(t)$ the number of (positive) units in neuron i at time t, also called the *potential* of neuron i. Neuron i is said to be *active* at time t iff $X_i(t) > 0$. When neuron i is active, it sends units (positive or negative) to other neurons or to outside (with rate μ_i). When a (positive) customer ends getting service at neuron i, it goes to neuron j as a positive one with (routing) probability $p_{i,j}^+$ and as a negative one with (routing) probability $p_{i,j}^-$. The unit leaves the network with probability $d_i = 1 - \sum_{j\in\mathcal{N}}(p_{i,j}^+ + p_{i,j}^-)$.

The previous description means that when neuron i is active, it sends positive signals to neuron j with rate (also called here *weight*) $w_{i,j}^+ = \mu_i p_{i,j}^+$ and negative ones to neuron j with rate $w_{i,j}^- = \mu_i p_{i,j}^-$; it sends units outside with rate $\delta_i = \mu_i d_i$. Observe that $\delta_i + \sum_{j\in\mathcal{N}}(w_{i,j}^+ + w_{i,j}^-) = \mu_i$.

In the stable case, the *activity rate* of neuron i (the *utilization rate* of queue i) is $\varrho_i = \lim_{t\to\infty} \Pr(X_i(t) > 0) > 0$; also, the mean throughput of positive units or signals that arrive at neuron i is $T_i^+ = \lambda_i^+ + \sum_{j\in\mathcal{N}} \varrho_j w_{j,i}^+$, and the mean arrival throughput of negative units at i is $T_i^- = \lambda_i^- + \sum_{j\in\mathcal{N}} \varrho_j w_{j,i}^-$. Look at neuron i as a queue and assume process $X_i()$ is

stationary. Consider that destroyed customers are departures and apply the mean flow conservation theorem: we obtain $T_i^+ = \varrho_i(\mu_i + T_i^-)$ and thus,

$$\varrho_i = \frac{T_i^+}{\mu_i + T_i^-}.$$

Last, we place ourselves in the standard independence conditions concerning all arrival, service and switching (choosing next node and class, or output, for a signal leaving a neuron) processes (as usual in queueing network models).

A basic result is then the following.

Theorem 1: The vector occupation process of this network is a Markov chain with state space $\mathbb{N}^N$. Assume it is irreducible (this depends on the routing probabilities of the model and on the arrival rates), and consider the relations

$$T_i^+ = \lambda_i^+ + \sum_{j\in\mathcal{N}} \varrho_j w_{j,i}^+, \quad T_i^- = \lambda_i^- + \sum_{j\in\mathcal{N}} \varrho_j w_{j,i}^- \quad \text{and} \quad \varrho_i = \frac{T_i^+}{\mu_i + T_i^-}$$

as a (non-linear) system of equations in the set of unknowns $(T_i^+,\ T_i^-,\ \varrho_i)_{i=1,\cdots,N}$. Then,

(i) The network is stable iff the system of equations has a solution where for $i = 1, \cdots, N$ we have $\varrho_i < 1$; in this case, the solution is unique.
(ii) In the stable case, the network is of the product-form type, and we have

$$\lim_{t\to\infty} \Pr(X_1(t) = n_1, \cdots, X_N(t) = n_N) = \prod_{i=1}^{N} (1 - \varrho_i)\varrho_i^{n_i}.$$

Proof: The proof is based on standard properties of Markov chains. See papers [7,8]. □

For instance, consider a single G-queue with arrival rate of positive (resp. negative) customers equal to $\lambda^+ > 0$ (resp. to $\lambda^- \geq 0$) and service rate $\mu > 0$. Then, applying Theorem 1 we have that the queue is stable iff $\lambda^+ < \mu + \lambda^-$, and in that case, its stationary state distribution is $n \mapsto (1-\varrho)\varrho^n$, where $\varrho = \lambda^+/(\mu+\lambda^-)$. The mean backlog at the queue (the mean potential of the neuron) is $\lambda^+/(\mu+\lambda^- - \lambda^+)$. Observe that its occuppation process is that of a $M/M/1$ queue with arrival rate λ^+ and service rate $\mu + \lambda^-$.

3.2. *Feedforward 3-layer G-networks*

In a feedforward queueing network, a customer never visits the same node twice. This makes the analysis simpler. In the case of a G-network, the non-linear system presented in Theorem 1 has an "explicit" solution.

A 3-layer model is a particular case of feedforward networks. There are three types of nodes: the "input" nodes (their set is $\mathcal{I}$), which are the only ones receiving signals from outside, the "hidden" ones (their set is $\mathcal{H}$) receiving all the signals leaving input nodes, and the "ouput" nodes (their set is $\mathcal{O}$), receiving all the signals leaving hidden ones, and sending all their signals outside. There is no connection between nodes in the same subset. Moreover, for learning applications, it is usually assumed that there is no negative unit coming to the network from outside; we make the same assumption here.

The particular topology of the network makes that we have explicit expressions of the activity rates of all neurons, starting from set $\mathcal{I}$, then going to $\mathcal{H}$ and last to $\mathcal{O}$. Consider an input neuron i. Since no other neuron can send units to it, we have $\varrho_i = \lambda_i^+/\mu_i$. Now, for a hidden neuron h, we have

$$\varrho_h = \frac{\sum_{i\in\mathcal{I}} \varrho_i w_{ih}^+}{\mu_h + \sum_{i\in\mathcal{I}} \varrho_i w_{ih}^-} = \frac{\displaystyle\sum_{i\in\mathcal{I}} \lambda_i^+ w_{ih}^+/\mu_i}{\mu_h + \displaystyle\sum_{i\in\mathcal{I}} \lambda_i^+ w_{ih}^-/\mu_i}.$$

Last, for any output neuron o, the corresponding expression is

$$\varrho_o = \frac{\sum_{h\in\mathcal{H}} \varrho_h w_{ho}^+}{\mu_o + \sum_{h\in\mathcal{H}} \varrho_h w_{ho}^-} = \frac{\displaystyle\sum_{h\in\mathcal{H}} \frac{\displaystyle\sum_{i\in\mathcal{I}} \lambda_i^+ w_{ih}^+/\mu_i}{\mu_h + \displaystyle\sum_{i\in\mathcal{I}} \lambda_i^+ w_{ih}^-/\mu_i} w_{ho}^+}{\mu_o + \displaystyle\sum_{h\in\mathcal{H}} \frac{\displaystyle\sum_{i\in\mathcal{I}} \lambda_i^+ w_{ih}^+/\mu_i}{\mu_h + \displaystyle\sum_{i\in\mathcal{I}} \lambda_i^+ w_{ih}^-/\mu_i} w_{ho}^-}.$$

As we will see next, when this queuing network is used to learn, it is seen as a function mapping the rates of the arrival processes into the activity rates of the nodes. In this 3-layer structure, we see that the function $\vec{\lambda}^+ = (\lambda_1^+, \cdots, \lambda_N^+) \mapsto \varrho_o$ is a rational one, and it can be easily checked that both its numerator and denominator are in general polynomials with degree $2H$ and non-negative coefficients.

3.3. *Learning*

Assume we are interested in some real function $f()$ from $[0..1]^{2N}$ into $[0..1]^N$. Function $f()$ itself is unknown, but we have K input-output values whose set is $\mathcal{D} = (\vec{l}^{\,(1)}, \vec{c}^{\,(1)}), \cdots, (\vec{l}^{\,(K)}, \vec{c}^{\,(K)})$ (that is, $f(\vec{l}^{\,(k)}) = \vec{c}^{\,(k)}$, $k = 1, \cdots, K$). We can use a stable G-network to "learn" $f()$ from the data set $\mathcal{D}$. For this purpose, consider a general RNN (not necessarily fedforward) with N nodes and all arrival rates in [0..1]. Now look at it as a black-box mapping the rates of the arrival process (the $2N$ numbers $\lambda_1^+, \cdots, \lambda_N^+, \lambda_1^-, \cdots, \lambda_N^-$) into the activity rates (the N numbers $\varrho_1, \cdots, \varrho_N$). We associate, for instance, the first N input variables of $f()$ with the arrival rates of positive units to nodes 1 to N, then the next N input variables with the arrival rates of negative units, last the N output variables of $f()$ with the activity rates of the G-network. The service rates and the routing probabilities (or equivalently, the weights) are seen as *parameters* of the mapping. The latter is here denoted as $\nu(\vec{\lambda})$, or $\vec{\nu}(W, \vec{\mu}; \vec{\lambda})$, etc., depending on making explicit the parameters or not, with $W = (W^+, W^-)$, $W^+ = (w_{ij}^+)$, $W^- = (w_{ij}^-)$, $\vec{\lambda} = (\vec{\lambda}^+, \vec{\lambda}^-)$, $\vec{\lambda}^+ = (\lambda_1^+, \cdots, \lambda_N^+)$, $\vec{\lambda}^- = (\lambda_1^-, \cdots, \lambda_N^-)$, and $\vec{\mu} = (\mu_1, \cdots, \mu_N)$.

Learning means looking for values of the parameters such that (i) for all $k = 1, \cdots, K$ we have $\nu(\vec{l}^{\,(k)}) \approx \vec{c}^{\,(k)}$ and, moreover, (ii) *for any other value* $\vec{x} \in [0..1]^{2N}$ we have $\nu(\vec{x}) \approx f(\vec{x})$. This last condition is experimentally validated, as usual in statistical learning, even if there are theoretical results about the way these $\nu()$ functions can approximate as close as desired any $f()$ function having some regularity properties. In order to find an appropriate G-network, the standard approach is to look only for network weights (matrix W), the rest of the parameters being fixed. To do this, we consider the cost function

$$C(W) = \frac{1}{2} \sum_{k=1}^{K} \sum_{o \in \mathcal{O}} \left[\nu_o(W;\, \vec{l}^{\,(k)}) - c_o^{(k)} \right]^2,$$

and we look for minima of $C()$ in the set $\{W \geq 0\}$[c]. A basic way to find good values of W is to follow a gradient descent approach. This process helps in understanding the algebraic manipulations of these models while much more efficient procedures exist (for instance, quasi–Newton meth-

[c]There can be stability issues here, depending on how we deal with the remaining parameters (service rates or departure probabilities), but we do not develop the point further for lack of space. Just observe that fixing the service rates high enough allows to control stability. For instance, $\mu_i = N$ for all neuron i is trivially sufficient for stability (recall we assume all arrival rates in [0..1]).

ods). For the basic approach, we build a sequence of matrices $W(0)$, $W(1)$, ..., (hopefully) converging to a pseudo-optimal one (that is, a matrix $\widehat{W}$ such that $C(\widehat{W}) \approx 0$). For instance, for all $i, j \in \mathcal{N}$, the typical update expressions for the weights between neurons i and j are

$$w^+_{i,j}(m+1) = w^+_{i,j}(m) - \eta \frac{\partial C}{\partial w^+_{i,j}}(W(m)),$$

$$w^-_{i,j}(m+1) = w^-_{i,j}(m) - \eta \frac{\partial C}{\partial w^-_{i,j}}(W(m)).$$

Factor $\eta > 0$ is called the *learning factor*; it allows to tune the convergence process. Observe that the previous expression can lead to a negative value for $w^+_{i,j}(m+1)$ or for $w^-_{i,j}(m+1)$. An usual solution to this is to use the iterations

$$w^+_{i,j}(m+1) = \left[w^+_{i,j}(m) - \eta \frac{\partial C}{\partial w^+_{i,j}}(W(m)) \right] \vee 0,$$

$$w^-_{i,j}(m+1) = \left[w^-_{i,j}(m) - \eta \frac{\partial C}{\partial w^-_{i,j}}(W(m)) \right] \vee 0.$$

Another possible decision is to *freeze* the value of an element of $W(m)$ when it reaches value zero. These are standard points in basic optimization methodology.

It remains how to compute the partial derivative in the previous expression. Writing

$$\frac{\partial C}{\partial w^*_*} = \sum_{k=1}^{K} \sum_{o \in \mathcal{O}} \left[\nu_o(W;\, \vec{l}^{\,(k)}) - c^{(k)}_o \right] \frac{\partial \nu_o}{\partial w^*_*},$$

we see that we still need to calculate the partial derivatives of the outputs, that is, of the activity rates, with respect to the weights.

After some algebra we can write the following relations: if $\vec{\varrho}$ is the vector $\vec{\varrho} = (\varrho_1, \cdots, \varrho_N)$[d],

$$\frac{\partial \vec{\varrho}}{w^+_{uv}} = \vec{\gamma}^+_{uv}(I - \Omega)^{-1}, \quad \frac{\partial \vec{\varrho}}{w^-_{uv}} = \vec{\gamma}^-_{uv}(I - \Omega)^{-1},$$

where

$$\Omega_{jk} = \frac{w^+_{jk} - \varrho_k w^-_{jk}}{\mu_k + T^-_k},$$

[d]All vectors are row vectors in this chapter.

$$\vec{\gamma}^+_{uv} = \frac{\varrho_u}{\mu_v + T^-_v}\vec{1}_v, \quad \vec{\gamma}^-_{uv} = -\frac{\varrho_u\varrho_v}{\mu_v + T^-_v}\vec{1}_v = -\varrho_v\vec{\gamma}^+_{uv},$$

vector $\vec{1}_j$ being the jth vector of the canonical base of $\mathbb{R}^N$.

A compact way of writing these derivatives is as follows. Denote by R the diagonal matrix $R = \mathrm{diag}(\varrho_i)_{i\in\mathcal{N}}$ and by D the diagonal matrix $D = \mathrm{diag}(\mu_i + T^-_i)_{i\in\mathcal{N}}$. Now, let i be fixed and let $A^{(i)} = (A^{(i)}_{uv})$ and $B^{(i)} = (B^{(i)}_{uv})$ be given by

$$A^{(i)}_{uv} = \frac{\partial\varrho_i}{w^+_{uv}}, \quad B^{(i)}_{uv} = \frac{\partial\varrho_i}{w^-_{uv}}.$$

We have:

$$A^{(i)}_{uv} = \frac{\varrho_u M_{vi}}{\mu_v + T^-_v} = \frac{\varrho_u M_{vi}}{D_{vv}}, \qquad B^{(i)}_{uv} = -\frac{\varrho_u\varrho_v M_{vi}}{D_{vv}} = -A^{(i)}_{uv},$$

where $M = (I - \Omega)^{-1}$. Vector $(M_{1i}, \cdots, M_{Ni})^{\mathrm{T}}$, the ith column of M, can be written $M\vec{1}_i^{\mathrm{T}}$. We now can write

$$A^{(i)} = \vec{\varrho}^{\mathrm{T}}\left(M\vec{1}_i^{\mathrm{T}}\right)^{\mathrm{T}} D^{-1} = \vec{\varrho}^{\mathrm{T}}\vec{1}_i M^{\mathrm{T}} D^{-1}.$$

In the same way,

$$B^{(i)}_{uv} = -\vec{\varrho}^{\mathrm{T}}\vec{1}_i M^{\mathrm{T}} D^{-1} R.$$

Resuming, learning means minimizing and minimizing means in this context, computing derivatives. This in turn means, in the general case, inverting a matrix whose dimension is equal to the number of parameters (the variables in the minimization process). In the particular case of feedforward models, the inversion can be done very easily (with an appropriate order in the variables, the matrix to invert is a triangular one).

3.4. *Sensitivity analysis*

One of the consequences of the nice mathematical properties of G-networks is that they allow to perform sensitivity analysis in a systematic way. Sensitivity analysis means here to be able to compute the derivatives of the activity rates with respect to the arrival rates.

After some straightforward algebra, we get the following relations: if $u \neq i$,

$$\frac{\partial\varrho_i}{\partial\lambda^+_u} = \sum_j \frac{\partial\varrho_j}{\partial\lambda^+_u}\Omega_{ji},$$

and

$$\frac{\partial \varrho_i}{\partial \lambda_i^+} = \sum_j \frac{\partial \varrho_j}{\partial \lambda_i^+} \Omega_{ji} + \frac{1}{\mu_i + T_i^-}.$$

In the same way, if $u \neq i$,

$$\frac{\partial \varrho_i}{\partial \lambda_u^-} = \sum_j \frac{\partial \varrho_j}{\partial \lambda_u^-} \Omega_{ji},$$

and

$$\frac{\partial \varrho_i}{\partial \lambda_i^-} = \sum_j \frac{\partial \varrho_j}{\partial \lambda_i^-} \Omega_{ji} - \frac{\varrho_i}{\mu_i + T_i^-}.$$

In matrix form, let us denote $F = (F_{iu})$ and $G = (G_{iu})$ where $F_{iu} = \partial \varrho_i / \partial \lambda_u^+$ and $G_{iu} = \partial \varrho_i / \partial \lambda_u^-$. If $u \neq i$, we have

$$F_{iu} = \sum_j F_{ju} \Omega_{ji} = \sum_j \Omega_{ij}^{\mathrm{T}} F_{ju} \quad \text{and} \quad G_{iu} = \sum_j G_{ju} \Omega_{ji} = \sum_j \Omega_{ij}^{\mathrm{T}} G_{ju}.$$

When $u = i$,

$$F_{ii} = \sum_j \Omega_{ij}^{\mathrm{T}} F_{ju} + \frac{1}{\mu_i + T_i^-} \quad \text{and} \quad G_{ii} = \sum_j \Omega_{ij}^{\mathrm{T}} G_{ju} - \frac{\varrho_i}{\mu_i + T_i^-}.$$

Using the same notation than in previous subsection, this leads to the expressions

$$F = D^{-1} M^{\mathrm{T}} \quad \text{and} \quad G = D^{-1} R M^{\mathrm{T}}.$$

As we see, the general formulæ for performing sentitivity analysis in the general case need the inversion of the same matrix as for the learning process. As before, a feedforward network structure simplifies considerably the computations (only a triangular matrix must be inverted).

4. Applications

In the case of the example described in Section 2, we use a RNN having 4 input nodes (corresponding to the 4 selected variables, BR, FR, LR and MLBS) and one output node, corresponding to the quality of the flow. Some details about the performance of RNN in the learning phase, their relative insensitivity with respect to the size of the subset of hidden nodes (in a reasonable range), and in general, their main properties, can be seen in [3]. For audio flows, see [4].

The first direct application of the PSQA technology is to analyze the impact of the selected variables on perceived quality. Since once the RNN

has learnt from data we have a function giving quality values for any configuration in the space we defined at the beginning of the process, we are able to analyze this quality function as a function of a part of the input vector, to compute its sensitivity with respect to the input variables, etc. Again, we refer to the given papers to see details about these results.

A second and richer set of results is as follows. Suppose you are interested in analyzing the impact of some specific part of the communication structure on quality. For instance, consider again our video example, and assume you want to analyze the impact of the size H of a specific buffer on quality. If you have a model that is able to evaluate the impact of H on the network parameters you selected when applying PSQA (LR and MLBS in our example), then you can map H into quality and perform your analysis efficiently. An example of this is [6], where we analyze the effect of FEC (Forward Error Correction) on the quality of voice flows. The basis of the approach are in [5]. Let us explain here this approach with more details, but keeping our video example of Section 2.

Assume we send a video stream using packets of constant length B bits through the Internet, from some source S to a receiver R. Applying the PSQA methodology, we get an explicit function $Q = \nu(\mathrm{BR}, \mathrm{FR}, \mathrm{LR}, \mathrm{MLBS})$ where BR is the bit rate of the connection, FR its frame rate, LR is its end-to-end loss probability and MLBS the average size of its bursts of losses.

The connection has a bottleneck with capacity c bps, and it is able to store up to H packets. For simplicity, assume that a packet contains exactly one frame. The flow of packets arriving to this bottleneck is Poisson with rate λ pps (packets per sec). With the standard assumptions, this node is a $M/M/1/H$ queue (at the packet level), leading to simple expressions of the loss probability p and (see [5]) mean loss burst size M. We have $p = (1-\varrho)\varrho^H/(1-\varrho^{H+1})$ and, after analyzing the Mrkov chain, $M = 1+\varrho$, where the load $\varrho = \lambda B/c$ is assumed to be $\neq 1$. If our flow is the only one using the bottleneck node, its quality can be written $Q = \nu(\lambda B, \lambda, p, M)$. More explicitly, this gives

$$Q = \nu\left(\lambda B, \lambda, \frac{(1-\lambda B/c)(\lambda B/c)^H}{1-(\lambda B/c)^{H+1}}, 1+\lambda B/c\right).$$

If other flows share the same bottleneck, then the difficulty lies in analyzing the loss rate and the mean loss burst size of our connection at that node. For instance, consider the case of K different flows sharing the node, with troughputs $\lambda_1, \cdots, \lambda_K$. Our flow is number 1. For simplicity, let us consider that all packets sizes are the same, B bits, and that the node can handle up

to H packets of all types. With the standard exponential assumptions, we have now a multiclass FIFO $M/M/1/H$ model with input rates $\lambda_1, \cdots, \lambda_K$ and service rate $\mu = c/B$. We need now to compute the loss probability for class 1 customers and the mean loss burst size of these customers. Since we assume Poisson arrivals, the loss probability is the same for all classes and thus can be computed from the $M/M/1/H$ single class formula with $\lambda = \lambda_1 + \cdots + \lambda_K$. It remains the problem of the mean burst loss size for class 1 customers. The first sub-problem is to define a burst in this multiclass context. Following [6], we define a burst in the following way. Assume a packet of class 1 arrives at a full queue and is lost, and assume that the previous class 1 packet that arrived was not loss. Then, a burst of class 1 losses starts. The burst ends when a packet is accepted, whatever its class is. See [6] for a discussion about this definition.

Let us denote by LBS the loss burst size for class 1 units. We have that $\Pr(\text{LBS} > n) = q^n$, where

$$q = \sum_{m \geq 0} \left(\frac{\lambda - \lambda_1}{\lambda + \mu} \right)^m \frac{\lambda_1}{\lambda + \mu} = \frac{\lambda_1}{\lambda_1 + \mu}.$$

The last relationship comes from the fact that we allow any number of class $k \neq 1$ units to arrive as far as their are lost, between two class 1 losses (that is, while the burst is not finished, no departure from the queue is "allowed"). Using the value of q, we have

$$\text{E(LBS)} = \sum_{n \geq 0} \Pr(\text{LBS} > n) = \cdots = 1 + \frac{\lambda_1}{\mu}.$$

With a straightforward extension of the results in [11], the analysis can be extended to the case of packets with different sizes for different flows. The discussion's aim was to show the kind of difficulty we may have to face at, when coupling PSQA to standard performance models. In some more complex cases the mapping from the variables the users wants to analyze and the inputs to the $\nu()$ function will need a simulator.

5. Conclusions

Using RNN, we have developed a technology allowing to put the *perceived* quality of a video (or audio, or multimedia) stream (that is, a subjective object, by definition) into numbers, after the flow arrives at the receiver through the Internet, and *automatically*. This evaluation process is not time consuming and can be therefore done *in real time*, if necessary.

When coupled with a standard performance model, this approach can allow the analyst to dimension a system adressing directly quality and not indirected metrics (such as those related to losses, delays, etc.). For instance, he/she will find the optimal channel speed (or buffer size, or number of terminals) that allows keeping quality over some threshold, instead of doing the same with loss rates, or delays, or jitter. The approach allows to work directly with quality, the "ultimate target".

Ongoing work with this technique includes applying it to control experiments, or to quality assessment in an operating network. Concerning the methodology itself and in particular the RNN tool, ongoing efforts are being done in order to explore efficient learning algorithms together with analyzing the mathematical properties of these specific dynamic models.

References

1. ITU-T Recommendation P.800, "Methods for subjective determination of transmission quality" (`http://www.itu.int/`).
2. ITU-R Recommendation BT.500-10, "Methodology for the subjective assessment of the quality of television pictures" (`http://www.itu.int/`).
3. Mohamed, S. and Rubino, G., "A study of real–time packet video quality using Random Neural Networks" *IEEE Transactions On Circuits and Systems for Video Technology*, **12** (12), (Dec. 2002), 1071-1083.
4. Mohamed, S. and Rubino, G. and Varela, M., "Performance evaluation of real-time speech through a packet network: a Random Neural Networks-based approach" *Performance Evaluation*, **57**(2), (2004), 141-162.
5. Rubino, G. and Varela, M., "A new approach for the prediction of end-to-end performance of multimedia streams",IEEE CS Press, 2004, 110-119. (1st International Conference on Quantitative Evaluation of Systems (QEST'04),University of Twente, Enschede, The Netherlands , Sept. 2004)
6. Rubino, G. and M. Varela. "Evaluating the utility of media–dependent FEC in VoIP flows", in Quality of Service in the Emerging Networking Panorama, LNCS 3266, 31-43 (2004).
7. Gelenbe, E., "Random Neural Networks with negative and positive signals and product form solution" *Neural Computation*,**1**(4), (1989), 502–511.
8. Gelenbe, E., "Stability of the Random Neural Network model" ISBN 3540522557, 1990, 56-68. (Proc. of Neural Computation Workshop, Berlin, Germany, Feb. 1990)
9. Gelenbe, E., "Learning in the recurrent Random Neural Network" *Neural Computation*, **5**(1), (1993), 154-511.
10. Bakircioglu, H. and Kocak, T., "Survey of Random Neural Network applications" *European Journal of Operational Research*, **126**(2), (2000), 319-330.
11. Marie, R. and Rubino, G., "Semi-explicit formulas for the M/M/1 Multiclass Queue" *Journal of the Theoretical Computer Science Institut*, Polish Academy of Sciences, **2**(1-2), (1990), 7-18.