Learning with monotonicity requirements for optimal routing with end-to-end quality of service constraints

Antoine Mahul, Alexandre Aussem

▶ **To cite this version:**

Antoine Mahul, Alexandre Aussem. Learning with monotonicity requirements for optimal routing with end-to-end quality of service constraints. European Symposium on Artificial Neural Networks, ESANN'06, 2006, Bruges, Belgium. pp. ?. hal-00266057

HAL Id: hal-00266057
https://hal.science/hal-00266057

Submitted on 20 Mar 2008

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.

Learning with monotonicity requirements for optimal routing with end-to-end quality of service constraints

Antoine Mahul[1] and Alexandre Aussem[2]

1- Université Blaise Pascal Clermond-Ferrand II - LIMOS / UMR 6158
Campus des Cézeaux, B.P.10125, 63173 Aubière - FRANCE

2- Université Claude Bernard Lyon I - Laboratoire PRISMa
8, boulevard Niels Bohr, 69622 Villeurbanne cedex - FRANCE

Abstract. In this paper, we adapt the classical learning algorithm for feed-forward neural networks when monotonicity is required in the input-output mapping. Monotonicity can be imposed by adding of suitable penalization terms to the error function. This yields a computationally efficient algorithm with little overhead compared to back-propagation. This algorithm is used to train neural networks for delay evaluation in an optimization scheme for optimal routing in a communication network.

1 Introduction

The multiplication of services in communication networks and the increasing demand for quality and reliability generate new challenges for the researchers. We consider here the optimization problems arising typically in Traffic Engineering where the overall operational performance of the communication network is to be maximized while satisfying some predefined Quality of Service (QoS) requirements. Unfortunately, the QoS, usually expressed in terms of average response time and/or loss rate, is difficult to express analytically in terms of the incoming traffic characteristics. It is also particularly appealing to train a feed-forward neural network as a "black-box" for the evaluation of the QoS values. As delay and loss are monotonically increasing with respect to the incoming traffic rates, inclusion of this prior knowledge into the training procedure can lead to better generalization. Moreover this monotonicity is also a stringent requirement for the optimization algorithm to converge.

To include the monotonicity as prior knowledge during learning, in [1], authors add virtual examples (*hints*) in the learning base. At the opposite, a specific neural architecture which is intrinsically monotonic is presented in [2]. We propose here a different strategy: to impose the monotonic property to neural networks by adding constraints in the learning problem.

First, we briefly present our traffic engineering problem and our resolution strategy using feed-forward neural networks as evaluation function of the QoS. Then, we propose a computationally efficient learning algorithm for feed-forward neural networks that takes into account monotonicity requirements. Finally, we present some numerical results.

2 Optimal routing problem with end-to-end constraints

We focus on the problem of routing a set of demands with distinct QoS requirements in an MPLS backbone in order to distribute the load equally in the network. We investigate how the classical M/M/1 assumption for delay evaluation could be relaxed by considering a slightly more accurate neural model in a multicommodity flow problem.

2.1 Problem description and resolution

In this paper, we restrict ourselves to the end-to-end delay defined, for each commodity, as the mean delay of packets between the departure time at the origin and the arrival time at the destination. The optimization problem is to route each commodity on the network in order to minimize the overall load on the network, defined as the overall sum of the arc loads, while respecting all QoS requirements.

The problem can be formulated as a multicommodity flow problem (see [3] for more details and a complete formulation) with a supplementary constraint for QoS on every active path. We propose the following nonlinear QoS constraints:

$$x_p^k \left(\sum_{a \in p} \psi_a^{s_k}(f_a) - \overline{D}_{s_k} \right) \leqslant 0 \qquad \forall k \in \mathcal{K}, \forall p \in \mathcal{P}_k \tag{1}$$

where $\mathcal{K}$ is the set of demands, $\mathcal{P}_k$ the set of all paths for demand $k \in \mathcal{K}$, x_p^k the quantity of flow for the demand $k \in \mathcal{K}$ routed on a path $p \in \mathcal{P}_k$, f_a the bandwidth vector (aggregated by class of service) for each class of an arc a. The evaluation function $\psi_a^s(f_a)$ represents the delay for class of service $s \in \mathcal{S}$ on arc $a \in A$ and $\overline{D}_s$ is the upper bound on the end-to-end delay for every demand associated with the service class s.

Therefore, our problem is an optimization problem with a nonlinear objective function, linear flow constraints and an exponential number of nonlinear non-convex QoS constraints. These constraints can be deported in the objective function by a lagrangian relaxation. We need to use an augmented lagrangian scheme in order to deal with the non-convexity of QoS constraints. In [4], we propose to adapt the flow deviation method (cf. [5]) for solving the inner problem. This algorithm converges toward a local optimum of the initial problem. However, this procedure is valid only if the evaluation function is increasing.

2.2 Neural approximation of quality of service

Typically, the M/M/1 approximation is used as the delay function ψ. But it ignores the class of service and therefore it is not valid in the case of differentiated services. A more realistic evaluation function of delay on an arc can be obtain with neural networks trained on discrete-event simulations (see [6]).

In the optimization process, input traffics are only characterized by bandwidth. So, from these rates aggregated per class, a neural network will be able

to estimate the mean delay for each class. The neural network was trained to approximate the function: $\{f^s\}_{s\in\mathcal{S}} \rightarrow \{d^s\}_{s\in\mathcal{S}}$, where f^s is the aggregated traffic of class s and d^s is the average delay for class s. The neural network was trained on examples generated by simulation of a router output interface.

However we have to ensure the monotonicity of evaluations for using the neural network into the optimization algorithm. We propose in the following to force the monotonicity during the learning of the neural network.

3 Learning with monotonicity requirements

3.1 Definitions and notations

In the following, we consider feed-forward network of arbitrary topology. Let $\mathcal{N}$ be the set of neurons. $\mathcal{N}_{\text{in}} \subset \mathcal{N}$ is the subset of input neurons and $\mathcal{N}_{\text{out}} \subset \mathcal{N}$ is the subset of output neurons. $\mathcal{A}$ is the set of arcs, and $w_c = w_{ij}$ the weight of an arc $c = (i, j) \in \mathcal{A}$. For given input vector x and weight vector w, $s_i(x, w)$ and $a_i(x, w)$ are respectively the input value and the activity of a neuron $i \in \mathcal{N}$. f_i is the activation function of a neuron $i \in \mathcal{N}$. The value of the input neurons are set to $x = \{s_i, i \in \mathcal{N}_{\text{in}}\}$. Hence, input and output of neurons are determined by the relations:

$$s_i(x, w) = \sum_{k\in\mathcal{N},(k,i)\in\mathcal{A}} w_{ki}\, a_k(x, w) \quad i \notin \mathcal{N}_{\text{in}}$$
$$a_i(x, w) = f_i(s_i(x, w)) \quad i \in \mathcal{N}$$

We also note $a'_i(x, w) = f'_i(s_i(x;))$ and $a''_i(x, w) = f''_i(s_i(x, w))$.

3.2 Learning problem subject to monotonicity constraints

Learning is classically achieved by a gradient descent method. Generally, we seek to minimize the quadratic error of estimation E on a given base $\mathcal{B}$ of examples. We propose to impose monotonicity during the learning process by adding constraints in the learning problem.

An output value y_i ($i \in \mathcal{N}_{\text{out}}$) of the neural network is increasing (resp. decreasing), on a compact set $\mathcal{K} \subset \mathbb{R}^{|\mathcal{N}_{\text{in}}|}$, according to an input value x_j ($j \in \mathcal{N}_{\text{in}}$) if the corresponding element $J_{ij}(x, w) = \frac{\partial y_i}{\partial x_j} = \frac{\partial a_i}{\partial s_j}(x, w)$ of the jacobian matrix is positive. However, the monotonicity is difficult to enforce on the whole domain $\mathcal{K}$. So we consider constraints only on examples given by the learning base $\mathcal{B}$. We suppose that the neural network will be able to generalize the monotonicity property on the whole domain. The learning problem (L) is then a nonlinear optimization problem subject to a finite number of nonlinear constraints:

$$(L)\begin{cases} \text{Min} & E(w) = \displaystyle\sum_{(x,y)\,\in\,\mathcal{B}} \sum_{i\,\in\,\mathcal{N}_{\text{out}}} \big(a_i(x, w) - y_i\big)^2 \\ \text{s.t.} & J_{ij}(x, w) \geq 0 \qquad \forall (x, \cdot) \in \mathcal{B},\ \forall i \in \mathcal{N}_{\text{out}},\ \forall j \in \mathcal{N}_{\text{in}} \quad (C) \\ & w \in \mathbb{R}^{|\mathcal{A}|} \end{cases}$$

3.3 Penalty method

We propose to use penalty methods (see [7] for details) to solve problem (L). Penalty methods solve a sequence of unconstrained subproblems which approach iteratively the infinite penalty function ($\sigma(x) = 0$ if all constraints are valid and $\sigma(x) = \infty$ otherwise). The penalty function $\Phi(w, \mu)$ associated to problem (L) can be written ($w \in \mathbb{R}^{|\mathcal{A}|}$ is the vector of the weights of the neural network) as:

$$\Phi(w,\mu) = \sum_{(x,y)\,\in\,\mathcal{B}} \left(\underbrace{\sum_{i\,\in\,\mathcal{N}_{\text{out}}} \big(a_i(x,w) - y_i\big)^2}_{E(x,w)} + \mu \underbrace{\sum_{i\,\in\,\mathcal{N}_{\text{out}}} \sum_{j\,\in\,\mathcal{N}_{\text{in}}} \varphi\big(J_{ij}(x,w)\big)}_{P(x,w)} \right)$$

where φ is a function from $\mathbb{R}$ to $\mathbb{R}$. We use the quadratic penalty function $\varphi(x) = \frac{1}{2}\min(0,x)^2$. At an iteration k of the penalty method, we have to solve the subproblems (P_k) $\min_w \Phi(w, \mu_k)$ where the sequence μ_k is chosen such that $\lim_{k\to\infty} \mu_k = \infty$. In order to apply a gradient descent method for solving these subproblems, we must be able to compute the gradient $\nabla\Phi(w, \mu)$. The term $\nabla E(w, \mu)$ can be computed by the classical back-propagation algorithm. We focus on the penalty term $\nabla P(w, \mu)$ in the next section.

4 Forward-backward algorithm for gradient computation

We propose in this section an algorithm to compute the penalty gradient ∇P. We proceed in a way similar to the curvature-driven learning proposed by [8] in the case of regularization.

Let x be an input vector of the neural network and w a weight vector. In order to simplify the notations, we will omit x and w in the sequel. For a given weight w_{kl}, $(k,l) \in \mathcal{A}$, the corresponding component of ∇P is

$$\frac{\partial P}{\partial w_{kl}} = \sum_{i\,\in\,\mathcal{N}_{\text{out}}} \sum_{j\,\in\,\mathcal{N}_{\text{in}}} \frac{\partial J_{ij}}{\partial w_{kl}}\, \varphi'\big(J_{ij}\big) \tag{2}$$

We generalize the definition of $J_{ij} = \frac{\partial a_i}{\partial s_j}$ for all $i \in \mathcal{N}$. When j is an input neuron, s_j is independant of any weights, so we can write:

$$\frac{\partial J_{ij}}{\partial w_{kl}} = \frac{\partial}{\partial w_{kl}}\left[\frac{\partial a_i}{\partial s_j}\right] = \frac{\partial}{\partial s_j}\left[\frac{\partial a_i}{\partial w_{kl}}\right] = J_{il} J_{kj} + a_k \frac{\partial J_{il}}{\partial s_j}$$

If we define $J^+_{ij} = \sum_{k\,\in\,\mathcal{N}_{\text{out}}} J_{ki}\, \varphi'\big(J_{kj}\big)$ and $\nu^+_{ij} = \sum_{k\,\in\,\mathcal{N}_{\text{out}}} \frac{\partial J_{ki}}{\partial s_j}\, \varphi'\big(J_{kj}\big)$, for all $(i,j) \in \mathcal{N} \times \mathcal{N}_{\text{in}}$ then equation (2) becomes:

$$\frac{\partial P}{\partial w_{kl}} = \sum_{j\,\in\,\mathcal{N}_{\text{in}}} \Big(J_{kj}\, J^+_{lj} + a_k\, \nu^+_{lj} \Big) \tag{3}$$

Let $\kappa_{ij} = \frac{\partial s_i}{\partial s_j}$, defined for $(i,j) \in \mathcal{N} \times \mathcal{N}_{\text{in}}$. The calculation of jacobian elements can be performed with the following forward relations:

$$J_{ij} = a'_i \kappa_{ij} \quad \text{and} \quad \kappa_{ij} = \begin{cases} \delta^i_j & \text{if } i \in \mathcal{N}_{\text{in}}, \\ \displaystyle\sum_{k,(k,i)\in\mathcal{A}} w_{ki}\, J_{kj} & \text{otherwise.} \end{cases} \tag{4}$$

where δ^i_j is the Kronecker symbol. And we can establish back-propagation formula for the calculation of J^+_{ij} and ν^+_{ij} (for $i \notin \mathcal{N}_{\text{out}}$):

$$J^+_{ij} = \sum_{\substack{k\in\mathcal{N} \\ (i,k)\in\mathcal{A}}} w_{ik}\, a'_i\, J^+_{kj} \quad \text{and} \quad \nu^+_{ij} = \sum_{\substack{k\in\mathcal{N} \\ (i,k)\in\mathcal{A}}} w_{ik} \left(a''_i\, \kappa_{ij}\, J^+_{kj} + a'_i\, \nu^+_{kj} \right) \tag{5}$$

Then when $i \in \mathcal{N}_{\text{out}}$, we have $J^+_{ij} = a'_i\, \varphi'(J_{ij})$ and $\nu^+_{ij} = a''_i\, \kappa_{ij}\, \varphi'(J_{ij})$.

Finally, the gradient $\nabla P(x)$ can be computed for a given input vector x with the following forward-backward algorithm: first compute, in the topological order (from inputs to outputs), κ_{ij} and J_{ij} using (4); then compute, in the reverse topological order (from outputs to inputs), J^+_{ij} and ν^+_{ij} using (5); finally, use (3) to compute the overall gradient. Like the standard back-propagation algorithm for error gradient computation, the complexity of this algorithm is linear according to the number of synaptic weights.

5 Numerical results

In these numerical experiments, we consider a DiffServ router with 3 classes of services and a simplified scheduler (EF a priority class; AF1 and AF2 two symmetrical classes scheduled with a round robin algorithm). We build the learning base by discrete-event simulation and examples are chosen randomly. We report learning results in Table 1 with classical back propagation learning and with constrained learning, and we have considered two models of traffic: poissonian and On/Off traffic. The first columns are the normalized mean squared error (NMSE) for each class of service, the fourth column is the mean squared of negative jacobian elements measuring the monotonicity error and the last column is the percent of decreasing patterns. All values are measured on validation sets of examples with a cross-validation technique. We observe that constrained learning gives negligible error of monotonicity on unknown examples.

		NMSE ($\times 10^{-3}$)			Monotonicity	
Traffic	Learning	EF	AF1	AF2	error	%
Poisson	back prop.	1,10	6,21	6,22	$1,95 \cdot 10^{-3}$	10,0%
	penalty	3,01	20,01	14,85	0.0	0.0%
On/Off	back prop.	1,11	4,70	3,29	$1,14 \cdot 10^{-2}$	28,8%
	penalty	9,25	20,00	13,98	$9,94 \cdot 10^{-6}$	1,7%

Table 1: Learning results with a 10 hidden units MLP

Then we use these models of delay in our optimization scheme, considering a network instance of 10 nodes, 44 arcs and 20 demands. Results are reported in Table 2 with three models of delay. Columns are respectively the network load at the optimal solution (f^*), the maximal violation ($\| g \|_\infty$), n_{vio} the number of violating paths and n_{tot} the number of active paths.

Delay model	f^*	$\| g \|_\infty$	n_{vio}/n_{tot}	Iterations	CPU (sec.)
M/M/1	22,483721	0.	0 / 43	28351790	3320
MLP:Poisson	22,456047	0.	0 / 40	11537021	13312
MLP:On/Off	22,730098	$8,08 \cdot 10^{-8}$	1 / 61	4355476	6575

Table 2: Optimal routing with delay constraints and 3 functions of delay.

These results are useful in proving that the way the QoS is approximated has a major impact on the routing solution obtained by the optimization procedure. According to the QoS model used, the optimal routes differ significantly, in particular with the neural model based on On/Off model.

6 Conclusion

We presented a learning algorithm satisfying monotonicity requirements for feed-forward neural networks and we uses this learning approach to solve a routing problem with delay constraints in communication networks. The neural network was trained to estimate delays induced by the load of the network, which is an unknown increasing function of the traffic rates. This neural estimator was then used in a routing optimization scheme for which the monotonous condition of the delay function is a fundamental condition.

References

[1] Joseph Sill and Yaser S. Abu-Mostafa. Monotonic hints. *Advances in Neural Information Processing Systems*, 9:634, 1997.

[2] Joseph Sill. Monotonic networks. *Advances in Neural Information Processing Systems*, 10:661–667, 1998.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Networks Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[4] Antoine Mahul. *Apprentissage de la Qualité de Service dans les réseaux multiservices: applications au routage optimal sous contraintes*. PhD thesis, Université Blaise Pascal, 2005.

[5] L. Fratta, M. Gerla, and L. Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3:97–133, 1973.

[6] Antoine Mahul and Alexandre Aussem. Distributed neural networks for QoS estimation in communication network. *International Journal of Computational Intelligence and Applications*, 3(3):297–308, 2003.

[7] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.

[8] Christopher M. Bishop. Curvature-driven smoothing: a learning algorithm for feed-forward networks. *IEEE Transactions on Neural Networks*, 4(5), September 1993.