Deep Non-Crossing Quantiles through the Partial Derivative

Axel Brando
UB[1] & BSC[2]

Joan Gimeno
UB[1]

José A. Rodríguez-Serrano
BBVA[3]

Jordi Vitrià
UB[1]

Abstract

Quantile Regression (QR) provides a way to approximate a single conditional quantile. To have a more informative description of the conditional distribution, QR can be merged with deep learning techniques to simultaneously estimate multiple quantiles. However, the minimisation of the QR-loss function does not guarantee non-crossing quantiles, which affects the validity of such predictions and introduces a critical issue in certain scenarios. In this article, we propose a generic deep learning algorithm for predicting an arbitrary number of quantiles that ensures the quantile monotonicity constraint up to the machine precision and maintains its modelling performance with respect to alternative models. The presented method is evaluated over several real-world datasets obtaining state-of-the-art results as well as showing that it scales to large-size data sets.

1 INTRODUCTION

Quantile Regression (QR) allows us to approximate a desired quantile – unlike the classical regression that only estimates the mean or the median – of the conditional distribution $p(Y \mid \boldsymbol{X})$. This is useful since we can capture confidence intervals without making strong assumptions about the distribution function to approximate. The formal definition of QR is:

Definition 1 (Conditional quantile regression)**.** *Let $\boldsymbol{X} \in \mathbb{R}^D$ and $Y \in \mathbb{R}$ be respectively a covariate and a response random variable. Given τ in the real interval $[0, 1]$, the conditional quantile regression (QR) consists in finding a function $q_\tau : \mathbb{R}^D \to \mathbb{R}$ which approximates the τ-th quantile of $p(Y \mid \boldsymbol{X})$ by minimising the τ-th quantile regression loss function defined as*

$$\mathcal{L}_{QR}(\boldsymbol{X}, Y, \tau) = \mathbb{E}\Big[\Big(Y - q_\tau(\boldsymbol{X})\Big) \cdot \Big(\tau - \mathbb{1}[Y < q_\tau(\boldsymbol{X})]\Big)\Big], \quad (1)$$

where $\mathbb{1}[c]$ denotes the indicator function that verifies the condition c.

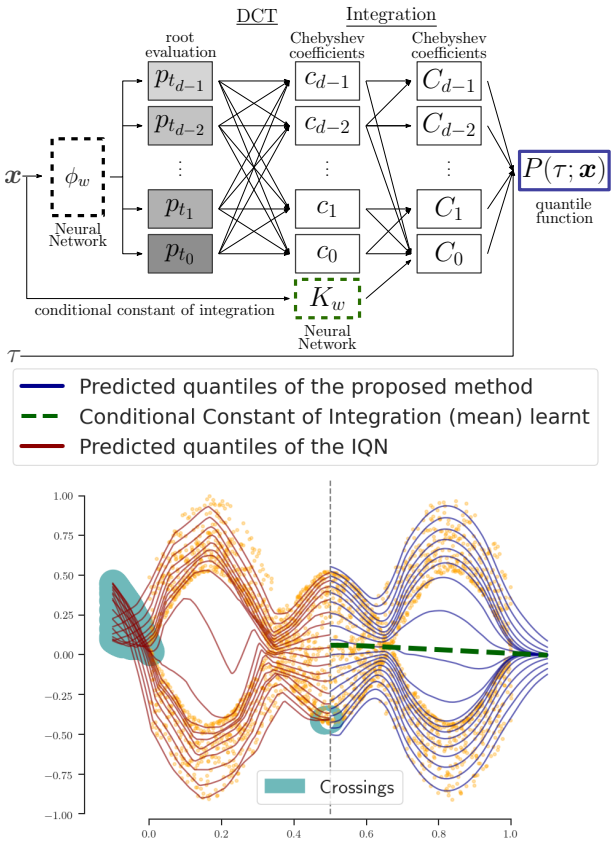

Figure 1: Synthetic data set showing that our proposed method (with the above architecture) avoids crossing quantiles (right), unlike IQN (left).

Generically, the loss function in Eq. 1 is asymmet-

[1] Universitat de Barcelona (UB).
[2] Barcelona Supercomputing Center (BSC).
[3] BBVA AI Factory (BBVA).

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

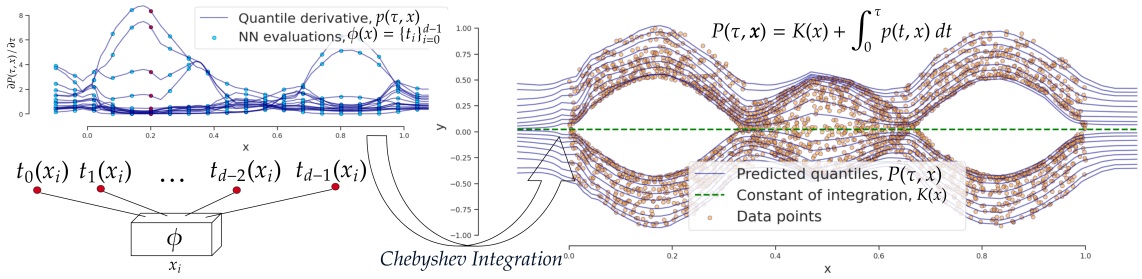

Figure 2: In general terms, the proposed method uses a NN - ϕ at the bottom left - to generate d **positive values** –the red $\{t_j(\boldsymbol{x})\}_{j=0}^{d-1}$ points–, which will be used as roots for computing the coefficients of a Chebyshev polynomial, represented in the top left subfigure. This polynomial will approximate the partial derivative of the quantile function. To do that, we integrate it obtaining a new Chebyshev polynomial, the one in the right subfigure. Thus, for each $\boldsymbol{x}$ we have a Chebyshev polynomial modelling all the quantiles.

ric, convex, and it penalises overestimation errors with weight τ and underestimation errors with weight $1-\tau$.

Based on this loss function, QR models such as quantile regression forests (Meinshausen and Ridgeway, 2006), gradient boosted quantile regression models (Zhang et al., 2018) or even neural network models (Dabney et al., 2018d) can be designed to approximate a discrete set of quantiles.

In all these cases the set of values of each τ is fixed a priori. Thus, the number of quantiles to approximate in a QR scenario constitutes a hyperparameter of the model definition and does not allow to approximate the full quantile function.

Alternatively, following Dabney et al. (2018a); Tagasovska and Lopez-Paz (2019); Brando et al. (2019), if the models are trained stochastically (such as the neural networks which use stochastic gradient descent), they can be defined to learn implicitly the quantile function, i.e. where the quantile to predict is an input parameter, τ. As Dabney et al. (2018a) states, this approach allows to approximate any conditional distribution given sufficient model capacity. In that case, we can extend Eq. 1 to learning the full quantile distribution as follows:

Definition 2 (Conditional quantile function)**.** *A function $\Phi_w \colon [0,1] \times \mathbb{R}^D \to \mathbb{R}$ with parameters w approximates the quantile function when it minimises the conditional quantile regression loss function defined as*

$$\mathcal{L}(\boldsymbol{X}, Y) = \mathbb{E}\left[\int_0^1 \Big(Y - \Phi_w(\tau, \boldsymbol{X})\Big) \cdot \Big(\tau - \mathbb{1}[Y < \Phi_w(\tau, \boldsymbol{X})]\Big)\, d\tau\right], \quad (2)$$

where $\mathbb{1}[c]$ denotes the indicator function that verifies the condition c.

The integral in Eq. 2 is numerically hard to compute because the analytical expression of Φ_w is, generically, not known. Following Brando et al. (2019), we can apply a Monte-Carlo strategy to provide a feasible loss function. This is based on considering a uniform random variable $\tau \sim \mathcal{U}(0,1)$ and for each evaluation of the loss function in Eq. 2, a τ-sample set, $\{\tau_t\}_{t=1}^{N_\tau}$, of N_τ points is generated in each training iteration such that

$$\mathcal{L}(\boldsymbol{X}, Y) \approx \mathbb{E}\left[\frac{1}{N_\tau}\sum_{t=1}^{N_\tau}\Big(Y - \Phi_w(\tau_t, \boldsymbol{X})\Big) \cdot \Big(\tau_t - \mathbb{1}[Y < \Phi_w(\tau_t, \boldsymbol{X})]\Big)\right]. \quad (3)$$

As we highlighted, the Φ_w requires to be a model that can be trained by using a Monte-Carlo approach. Hence, neural networks models will be suitable.

The modellization of quantile functions (shown in Eq. 2) constitutes an important goal, especially when the distribution to be predicted, $p(Y \mid \boldsymbol{X})$, is complex or when we are interested to impose the minimum assumptions about the shape of the distribution. A common practice is to consider an exponential power distribution (e.g. the Normal or Laplace distributions). In such cases, it is assumed unimodality, symmetry, and lose critical information regarding the shape of the distribution.

Importantly, by building a quantile function we are able to approximate - in a discrete manner but with arbitrary precision - the distribution $p(Y \mid \boldsymbol{X})$. Typically, quantiles are used to build confidence intervals. However, when these confidence intervals are tight or when we want to recover the distribution shape from the discretization, we might face with the critical problem described below.

Since the different predicted quantile values are esti-

mated individually, they may not be ordered according to their quantile value, τ. For instance, the predicted quantile 0.1 may not be a value greater than the quantile 0.05. As a consequence, we obtain an non-valid distribution of the response variable. That phenomenon is known as the *crossing quantile* phenomenon (Koenker et al., 2017):

Definition 3 (Crossing quantile phenomenon)**.** *Let $f\colon [0,1]\times\mathbb{R}^D\to\mathbb{R}$ be a conditional quantile function that predicts a certain quantile of a random variable $y\in\mathbb{R}$ given $\boldsymbol{x}\in\mathbb{R}^D$. If there exist $\tau_1,\tau_2\in[0,1]$ being $\tau_1<\tau_2$ and $\boldsymbol{x}\in\mathbb{R}^D$ such that*

$$f(\tau_1,\boldsymbol{x})>f(\tau_2,\boldsymbol{x}),$$

then f suffers the crossing quantile *phenomenon.*

Different solutions have been proposed to overcome this phenomenon. Most of them are based on adding a penalisation term to regularise the optimisation process and "encourage" the model to reduce the number of crossing quantiles Koenker and Hallock (2001); Bondell et al. (2010); Tagasovska and Lopez-Paz (2018). Nevertheless, since the model is not restricted to be **partially monotonic** - i.e. monotonic with respect to τ and not over the other inputs - some quantiles may still cross.

2 IMPOSING A POSITIVE PARTIAL DERIVATIVE

Following the idea of partial monotonicity, we want to estimate a partial derivative and use its integral as a quantile function.

Definition 4 (Partial Derivative of the Conditional Quantile Function)**.** *Let $\Phi_w\colon [0,1]\times\mathbb{R}^D\to\mathbb{R}$ be a quantile function with parameters w and let $\phi_w\colon [0,1]\times\mathbb{R}^D\to\mathbb{R}_+$ and $K_w\colon\mathbb{R}^D\to\mathbb{R}$ be two functions with parameters w. If we assume that*

$$\Phi_w(\tau,\boldsymbol{x})=K_w(\boldsymbol{x})+\int_0^\tau\phi_w(t,\boldsymbol{x})\,dt, \tag{4}$$

then the conditional quantile function can be approximated by minimizing the following loss function,

$$\mathcal{L}(\boldsymbol{X},Y)=\mathbb{E}\Bigg[\int_0^1\left(Y-\left(K_w(\boldsymbol{X})+\int_0^\tau\phi_w(t,\boldsymbol{X})\,dt\right)\right)\cdot\left(\tau-\mathbb{1}\left[Y<\left(K_w(\boldsymbol{X})+\int_0^\tau\phi_w(t,\boldsymbol{X})\,dt\right)\right]\right)d\tau\Bigg], \tag{5}$$

where $\mathbb{1}[c]$ denotes the indicator function that verifies the condition c.

Applying the approach in Definition 4, we are modelling the partial derivative with respect to the quantile function, i.e. $\partial\Phi(\tau,\boldsymbol{x})/\partial\tau =: \phi_w(\tau,\boldsymbol{x})$. Thus, we can impose conditions to that derivative. In our case, we want to impose that it will always be strictly positive to ensure the partial monotonic property of the predicted quantile function, i.e. $\forall\tau_1,\tau_2\in[0,1]$, if $\tau_1<\tau_2$ then $\Phi(\tau_1,\boldsymbol{x})<\Phi(\tau_2,\boldsymbol{x})$.

2.1 Model Definition

Following the Definition 4, the main contribution of this article considers a conditional quantile function $\Phi_w\colon[0,1]\times\mathbb{R}^D\to\mathbb{R}$ that it is obtained by integrating its partial derivative ϕ_w and considering the constant of integration, $K_w(\boldsymbol{x})$. Generically speaking, $K_w(\boldsymbol{x})$ will manage the modelling of the conditional mean or the quantile 0 of $p(Y\mid\boldsymbol{X})$ and $\phi_w(\tau,\boldsymbol{x})$ will approximate the rest of the $p(Y\mid\boldsymbol{X})$ distribution as quantiles of a new $q(Y\mid\boldsymbol{X})$.

To ensure the partial monotonicity of Φ_w, a neural network $\phi_w\colon\mathbb{R}^D\to\mathbb{R}^+\times\overset{d}{\cdots}\times\mathbb{R}^+$ with d positive outputs (as it is shown in Figure 1) will be considered as the partial derivative of Φ_w. Thus, the neural network ϕ_w approximates the derivative of the final quantile function with respect to the quantile τ.

The integration process of ϕ_w will be done by using truncated Chebyshev polynomial expansion of order d. Due to the truncation, we consider a finite mesh of quantile values, called *Chebyshev roots* or *roots*, $\{t_k\}_{k=0}^{d-1}\subset[0,1]$ which are defined as

$$t_k=\frac{1}{2}\cos\left(\frac{\pi(k+\frac{1}{2})}{d}\right)+\frac{1}{2},\quad 0\leqslant k<d. \tag{6}$$

These Chebyshev roots only depend on the degree d and not on the quantile input. Therefore, the roots are fixed once the degree is chosen[1].

The truncated Chebyshev expansion consists in expressing a function as a linear combination of Chebyshev polynomials. These polynomials are defined as mappings $T_k\colon[-1,1]\to\mathbb{R}$ given by the recurrent formula

$$\begin{aligned}T_0(t)&\coloneqq 1,\\ T_1(t)&\coloneqq t,\\ T_{k+1}(t)&\coloneqq 2tT_k(t)-T_{k-1}(t),\quad k\geqslant 1.\end{aligned} \tag{7}$$

In our case, τ is in $[0,1]$ rather than in $[-1,1]$ which slightly modifies the standard polynomial definition and allows us to approximate $\phi_w(\tau,\boldsymbol{x})$ by

$$p(\tau,\boldsymbol{x};d)\coloneqq\frac{1}{2}c_0(\boldsymbol{x})+\sum_{k=1}^{d-1}c_k(\boldsymbol{x})T_k(2\tau-1). \tag{8}$$

[1]The non-dependency of the quantile value will be crucial to ensure the monotonicity of the global predicted quantile function up to the machine precision.

Like the roots, the coefficients $c_j(\boldsymbol{x})$ in Eq. 7 are independent of the quantiles. These quantities are computed for $j = 0, \ldots, d-1$, by

$$c_j(\boldsymbol{x}) \coloneqq \frac{2}{d} \sum_{k=0}^{d-1} \phi_w(t_k, \boldsymbol{x}) \cos\left(\frac{j\pi(k+\frac{1}{2})}{d}\right), \quad (9)$$

Eq. 9 is merely a matrix-vector product from the matrix of cosines and the vector of $\phi_w(t_k, \boldsymbol{x})$ that instead of a complexity $\Theta(d^2)$, it can be computed in logarithmic complexity, $\Theta(d \log d)$.

The polynomials T_k in Eq. 8 do not need to be explicitly computed and, by construction of the coefficients $c_k(\boldsymbol{x})$ in Eq. 9, $p(t_k, \boldsymbol{x}; d)$ is "equal to" $\phi_w(t_k, \boldsymbol{x})$ for all the Chebyshev roots, $\{t_k\}_{k=0}^{d-1}$, in Eq. 6. Note that $\phi_w(t_k, \boldsymbol{x})$ are denoted by p_{t_k} in Figure 1. That equality must, in practice, be understood in terms of machine precision of the numerical representation system, classically $\sim 10^{-16}$ in double-precision or $\sim 10^{-8}$ in single-precision arithmetic. The root evaluation step is illustrated in Figure 1 and its values are denoted as p_{t_k}.

Once $p(t, \boldsymbol{x}; d)$ has been encoded by its $c_j(\boldsymbol{x})$ coefficients in Eq. 9, we can easily compute its integral function $\int_0^\tau p(t, \boldsymbol{x}; d)\, dt$ denoted by $P(\tau, \boldsymbol{x}; d)$. Thus P will be an approximation of the integral of the neural network, ϕ_w. That is,

$$P(\tau, \boldsymbol{x}; d) \approx \Phi_w(\tau, \boldsymbol{x}) = K_w(\boldsymbol{x}) + \int_0^\tau \phi_w(t, \boldsymbol{x})\, dt. \quad (10)$$

Additionally, given that we imposed that the neural network $\phi_w(t, \boldsymbol{x})$ gives only positive values for all $t \in [0, 1]$, then $P(\tau, \boldsymbol{x}; d)$ would be an increasing function with respect to τ as long as d is large enough.

In the above procedure, the integral of ϕ_w in Eq. 10 is, in general, not straightforward to compute. However, the neural network $\phi_w(\cdot, \boldsymbol{x})$ is globally being represented by $p(\cdot, \boldsymbol{x}; d)$ on the quantile interval $[0, 1]$. Therefore, by using its Chebyshev coefficients $c_k(\boldsymbol{x})$, we can encode the integral of P to provide its Chebyshev coefficients $C_k(\boldsymbol{x})$. In fact, according to Clenshaw (1955), the integral of $p(\tau, \boldsymbol{x}; d)$ gives another Chebyshev expansion, say $P(\tau, \boldsymbol{x}; d)$, with coefficients $C_k(\boldsymbol{x})$, i.e.,

$$P(\tau, \boldsymbol{x}; d) = \frac{1}{2} C_0(\boldsymbol{x}) + \sum_{k=1}^{d-1} C_k(\boldsymbol{x}) T_k(2\tau - 1). \quad (11)$$

To deduce the expressions for $C_k(\boldsymbol{x})$ in Eq. 11, we need to recurrently integrate the polynomials $T_k(t)$, whose integral are

$$\begin{aligned}
&\int T_0(t)\, dt = T_1(t) + \text{constant}, \\
&\int T_1(t)\, dt = \frac{T_2(t) + T_0(t)}{4} + \text{constant}, \\
&\int T_k(t)\, dt = \frac{T_{k-1}(t)}{2(k-1)} - \frac{T_{k+1}(t)}{2(k+1)} + \text{constant}, \quad k \geqslant 2.
\end{aligned} \quad (12)$$

By ordering the coefficients $c_j(\boldsymbol{x})$ in Eq. 8 in terms of $C_j(\boldsymbol{x})$ of Eq. 11, we deduce that

$$\begin{aligned}
&C_k(\boldsymbol{x}) \coloneqq \frac{c_{k-1}(\boldsymbol{x}) - c_{k+1}(\boldsymbol{x})}{4k}, \quad 0 < k < d-1, \\
&C_{d-1}(\boldsymbol{x}) \coloneqq \frac{c_{d-2}(\boldsymbol{x})}{4(d-1)},
\end{aligned} \quad (13)$$

and $C_0(\boldsymbol{x})$ depends on the constant of integration $K_w(\boldsymbol{x})$ in Eq. 10 and the other coefficient values in Eq. 11.

On the whole, the proposed method consisting of the following steps: for each $\boldsymbol{x}$ value,

1. A set of values, $\{p_{t_k}(\boldsymbol{x})\}_{k=0}^{d-1}$ is obtained via the neural network ϕ_w at the corresponding roots, $\{t_k\}_{k=0}^{d-1}$.
2. These are transformed to d coefficients, $\{c_k(\boldsymbol{x})\}_{k=0}^{d-1}$ using a fast matrix-vector multiplication algorithm. This results in a truncated Chebyshev polynomial, $p(\tau, \boldsymbol{x}; d)$.
3. This polynomial is then integrated to obtain another Chebyshev polynomial, $P(\tau, \boldsymbol{x}; d)$, whose coefficients are $\{C_k(\boldsymbol{x})\}_{k=0}^{d-1}$ in Eq. 13.

The final output is a Chebyshev polynomial for each $\boldsymbol{x}$ value that approximates the unknown conditional quantiles of $p(Y \mid \boldsymbol{X})$.

2.2 Selecting the Constant of Integration

The formula used to calculate the constant of integration $C_0(\boldsymbol{x})$ (shown in Eq. 13) will depend on which statistic we choose for $K(\boldsymbol{x})$, as shown in the following propositions.

Proposition 5 (Our-q_0)**.** *Let (P, K) be the proposed model and $q(Y \mid \boldsymbol{X})$ the distribution that P produces as quantiles. If K is the* ***lowest quantile*** *($\tau = 0$) of $q(Y \mid \boldsymbol{X})$, then the C_0 coefficient of the Chebyshev polynomial P verifies:*

$$C_0(\boldsymbol{x}) = 2K(\boldsymbol{x}) - 2 \sum_{k=1}^{d-1} C_k(\boldsymbol{x})(-1)^k. \quad (14)$$

Proof. The learning process is subjected to the condition that the lowest quantile value, i.e. $\tau = 0$, must be the K. That is, $P(0, \boldsymbol{x}; d) = K(\boldsymbol{x})$. By taking the value $t = -1$ in Eq. 7, we derive that $T_k(-1) = (-1)^k$. Therefore, using Eq. 11 with $\tau = 0$, we obtain Eq. 14. □

Proposition 6 (Our-Mean)**.** *Let (P, K) be the proposed model and $q(Y \mid \boldsymbol{X})$ the distribution that P produces as quantiles. If K is the **mean** of $q(Y \mid \boldsymbol{X})$, then the C_0 coefficient of the Chebyshev polynomial P verifies:*

$$C_0(\boldsymbol{x}) = 2K(\boldsymbol{x}) - 2 \sum_{\substack{k=1 \\ k \ odd}}^{d-1} \frac{C_k(\boldsymbol{x})}{k^2 - 4}. \tag{15}$$

Proof. Taking into account the definition of the (continous) mean, the condition we must impose is

$$\int_0^1 \tau P(\tau, \boldsymbol{x}; d)\, d\tau = K(\boldsymbol{x}). \tag{16}$$

Then, taking into account the linearity of the integral and after a change of coordinates ($t = 2\tau - 1$), the integral is reduced to compute the mean of the Chebyshev polynomials. That is,

$$\int_{-1}^1 t T_k(t)\, dt. \tag{17}$$

Taking into account the symmetries of the Chebyshev polynomials, we obtain the equality

$$\int_{-1}^1 t T_k(t)\, dt = (1 + (-1)^{k+1}) \int_0^1 t T_k(t)\, dt.$$

Then if k is even, Eq. 17 is zero. If k is now assumed to be an odd integer, then taking into account the recurrent definition of the Chebyshev polynomials, $2T_k(t) = T_{k+1}(t) + T_{k-1}(t)$ and then Eq. 12,

$$\int_{-1}^1 t T_k(t)\, dt = \frac{T_{k-2}(t)}{2(k-2)} - \frac{T_{k+2}(t)}{2(k+2)}\bigg|_0^1 = \frac{2}{k^2 - 4}.$$

From here, we can recover the expression in Eq. 15. □

2.3 Rate of Convergence of the Chebyshev Expansion and Machine Precision

In Chebyshev series theory (Trefethen, 2008; Majidian, 2017), if a function ϕ_w is of class C^1 in $[-1, 1]$, then its Chebyshev expansion converges absolutely and uniformly to ϕ_w in $[-1, 1]$. In case that ϕ_w is in $C^k([-1, 1])$, then its Chebyshev coefficients c_ℓ verify that $|c_\ell| = o(1/\ell^k)$. Moreover, if ϕ_w is now analytic, its Chebyshev expansion will have an exponential rate, i.e. $|c_\ell| = o(\exp(-\rho\ell))$ for some $\rho > 0$ denoting the radius strip in the complex plane whose strip contains all the Chebyshev coefficients c_ℓ. Therefore, smoother mappings are going to require less Cheybshev coefficients. To have a kind of measure about the accuracy of the approximation $p \approx \phi$, one needs to check the decay rate of the Chebyshev coefficients c_ℓ. Most of the times, it will be enough to monitor the last two coefficients $c_{d-1}(\boldsymbol{x})$, and $c_{d-2}(\boldsymbol{x})$ in absolute value.

Due to the discretisation in the root mesh $\{t_k\}_{k=0}^{d-1}$, which acts as a linear transformation, the evaluation of $P(t_k; \boldsymbol{x})$ will have roundoff error (i.e. machine precision) with respect to the integral of $\int_0^{t_k} \phi_w(t; \boldsymbol{x})\, dt$ and for the other values not in the mesh, its (absolute) error will be bounded by the aforementioned convergence rate.

2.4 Ensure Monotonicity for all Quantiles

There is one last detail to be totally confident that the function $P(\tau; \boldsymbol{x}) = f(\tau, \boldsymbol{x})$ is monotonic with respect to τ. It is required to consider that p is an approximation of ϕ_w, i.e. $p(\tau; \boldsymbol{x}) \approx \phi_w(\tau; \boldsymbol{x})$. Thus, although the ϕ_w function has been forced to be strictly positive, the approached Chebyshev polynomial p may not be. According to Section 2.3, we are certain that for the roots values, $\{t_k\}_{k=0}^{d-1}$, the Chebyshev estimation has a negligible error. However, in the middle points between the roots it is important to be careful.

Once the appropriate degree is known, the model with that degree can be considered a monotonic function. However, it is important to note that although the error will be small, the possibility of P ceasing of being monotonic exists in all non-root points. In case that the order must be increase to ensure the monotonicity, the number of parameters to learn will increase as well.

2.5 Calculation Procedure of Coefficients

To evaluate Equation 8 or Equation 11 for a value τ in $[0, 1]$, the Clenshaw's method (Clenshaw, 1955) or its stable numerical error version (Elliott, 1968; Newbery, 1974) works. Let us briefly summarise the evaluation at τ of $p(\tau; \boldsymbol{x})$ in Equation 8 (or $P(\tau; \boldsymbol{x})$ in Equation 11), whose numerical complexity is $\Theta(d)$.

2.6 Chebyshev Polynomial as a Framework

On the whole, the proposed method can be seen as a framework to use any deep learning architecture to build a partial monotonic function. This monotonic function with respect some of the input variables could be applied to solve, with guarantees, the crossing quantile phenomenon of conditional quantile regression models. The implementation of this framework

Algorithm 1 Evaluation of a Chebyshev sum at a given $\tau \in [0,1]$. In particular, useful for Eqs. 8 and 11.

procedure EVAL_CHEB(τ, $c_0(\boldsymbol{x}), \ldots, c_{d-1}(\boldsymbol{x})$)
  $d_1(\boldsymbol{x}) \leftarrow d_2(\boldsymbol{x}) \leftarrow d_3(\boldsymbol{x}) \leftarrow 0$.
  $\sigma \leftarrow 2\tau - 1$.
  for $k = d-1, d-2, \ldots, 1$ **do**
    $d_3(\boldsymbol{x}) \leftarrow d_1(\boldsymbol{x})$.
    $d_1(\boldsymbol{x}) \leftarrow 2\sigma d_1(\boldsymbol{x}) - d_2(\boldsymbol{x}) + c_k(\boldsymbol{x})$.
    $d_2(\boldsymbol{x}) \leftarrow d_3(\boldsymbol{x})$.
  return $\sigma d_1(\boldsymbol{x}) - d_2(\boldsymbol{x}) + 0.5c_0(\boldsymbol{x})$.

Prerequisites 2 Definitions and functions used for next Algorithms.

$\triangleright$ $\boldsymbol{x}$ has batch size and number of features as shape, i.e. $[bs, D]$.
$\triangleright$ RS(*tensor*, *shape*): reshape *tensor* to *shape*.
$\triangleright$ RP($tensor, n$): repeats n times the last dimension of *tensor*.
$\triangleright$ CC(T_1, T_2): concatenate T_1 and T_2 by using their last dimension.

Algorithm 3 Chebyshev coefficients of the integral of a non-negative neural network.

procedure CHEB_CS($\boldsymbol{x}, d, \phi_w, K$)
  $\{o_k(\boldsymbol{x})\}_{k=0}^{d-1} \leftarrow \phi_w(\boldsymbol{x})$ $\triangleright$ Apply any NN, ϕ_w.
  $\{c_k(\boldsymbol{x})\}_{k=0}^{d-1} \leftarrow$ DCT-II($\boldsymbol{o}, d$) $\triangleright$ Eq. 9
  $\{C_k(\boldsymbol{x})\}_{k=1}^{d-1} \leftarrow$ Integration step wrt $\{c_k(\boldsymbol{x})\}_{k=0}^{d-1}$
  $C_0(\boldsymbol{x}) \leftarrow 2K(\boldsymbol{x}) - 2\sum_{k=1}^{d-1} C_k(\boldsymbol{x})(-1)^k$ **return** $\{C_k(\boldsymbol{x})\}_{k=0}^{d-1}, \{c_k(\boldsymbol{x})\}_{k=0}^{d-1}$.

Algorithm 4 How to build the model by using any deep learning architecture for regression.

procedure BUILD_MODEL_GRAPH($\boldsymbol{x}, y, d, \phi, K, N_\tau$)
  $\triangleright$ N_τ is the number of non-roots to evaluate.
  $\{C_k(\boldsymbol{x})\}_{k=0}^{d-1}, \{c_k(\boldsymbol{x})\}_{k=0}^{d-1} \leftarrow$ CHEB_CS($\boldsymbol{x}, d, \phi, K$)
  $\boldsymbol{\tau} \leftarrow \mathcal{U}(0,1)$ $\triangleright$ $\boldsymbol{\tau}$ must has $[bs \cdot N_\tau, 1]$ shape.
  $\boldsymbol{o}_P \leftarrow$ EVAL_CHEB($\boldsymbol{\tau}, C_0(\boldsymbol{x}), \ldots, C_{d-1}(\boldsymbol{x})$)
  $\mathcal{L} \leftarrow (y - \boldsymbol{o}_P) \cdot (\boldsymbol{\tau} - \mathbb{1}[y < \boldsymbol{o}_P])$ $\triangleright$ Eq. 1 loss.
  return $\mathcal{L}$

applying to quantile regression is shown in Algorithm 4 and could be done with any automatic differentiation library such as TensorFlow (Abadi et al., 2016) or PyTorch (Paszke et al., 2017). Additionally, this implementation takes care of performing the matrix-vector product efficiently by using the DCT-II referred previously.

2.7 The Desired Quantile as an Input

It is important to highlight that even if the number of ϕ_w outputs is fixed in the proposed model, we are able to estimate any quantile as it is show in Algorithm 5.

Algorithm 5 How to evaluate the model for any quantile $\tau \in [0,1]$ desired to obtain $P(\tau; \boldsymbol{x})$ and $p(\tau; \boldsymbol{x})$.

procedure EVAL_MODEL($\boldsymbol{x}, d, \phi_w, K, \tau$)
  $\{C_k(\boldsymbol{x})\}_{k=0}^{d-1}, \{c_k(\boldsymbol{x})\}_{k=0}^{d-1} \leftarrow$ CHEB_CS($\boldsymbol{x}, d, \phi_w, K$)
  $\boldsymbol{o}_p \leftarrow$ EVAL_CHEB($\tau, c_0(\boldsymbol{x}), \ldots, c_{d-1}(\boldsymbol{x})$)
  $\boldsymbol{o}_P \leftarrow$ EVAL_CHEB($\tau, C_0(\boldsymbol{x}), \ldots, C_{d-1}(\boldsymbol{x})$)
  return $\boldsymbol{o}_P, \boldsymbol{o}_p$

2.8 Inverse Mapping

For all $\boldsymbol{x} \in \mathbb{R}^D$, when the predicted function is monotonic, then $P(\cdot; \boldsymbol{x})$ is bijective: Given $y \in \mathbb{R}$ in the image of a monotone $P(\cdot; \boldsymbol{x})$, we can get the unique quantile τ, for some fixed $\boldsymbol{x}$ and y, such that $P(\tau; \boldsymbol{x}) = y$. To obtain $\tau^* \approx P^{-1}(y, \boldsymbol{x})$, we can proceed by an iterative scheme given by

$$\tau_{n+1} = \tau_n - p(\tau_n, \boldsymbol{x})^{-1}\big(P(\tau_n; \boldsymbol{x}) - y\big), n \geqslant 1, \quad (18)$$

where the value $p(\tau_n, \boldsymbol{x})$ as well as $P(\tau_n; \boldsymbol{x})$ can be calculated by Equation 8.

3 ABLATION STUDY

To clarify the proposed method, in the following section we will propose a quasi non-crossing quantile model (not up to machine precision), which also estimates the partial derivative and the closest - as far as we know - literature model, but it cannot applicable as a quantile function estimator because it does not provide a uniform global behaviour in $[0,1]$.

3.1 Non-Strictly Positive Partial Derivative

Here we propose a slight variation of the proposed model, which we will denote as *Not Always Monotonic* model (NAM).

Rather than considering the interval of roots fixed, which implies to fix the roots values, an alternative manner to compute the Eq. 4 integral can be to compute the Clenshaw-Curtis formula over the interval $[0, \tau]$. In that case, that formula, considering the quantile input, consists in the following steps:

1. First, we fix an *even* integer d, called degree, and we define the so-called nodes depending on τ:

$$\bar{t}_k^d(\tau) \coloneqq \frac{\tau}{2}\cos\left(\frac{\pi k}{d}\right) + \frac{\tau}{2}, \quad 0 \leqslant k \leqslant d. \quad (19)$$

   These nodes have the property that $\bar{t}_k^{2d}(\tau) = \bar{t}_{k/2}^d(\tau)$, which means that half of them can be reused when the value of d is doubled.

2. Second, we compute the quantities $\bar{c}_j(\tau, \boldsymbol{x})$ defined for $0 \leqslant j \leqslant d$ as

$$\bar{c}_j(\tau, \boldsymbol{x}) \coloneqq \sum_{k=0}^{d} \phi_w(\bar{t}_k^d(\tau), \boldsymbol{x}) \cos\left(\frac{j\pi k}{d}\right), \quad (20)$$

which is just a matrix-vector multiplication. Here, we can use an algorithm, such as the Discrete Cosine Transform of type 1 (DCT-I), which performs the matrix-vector multiplication in Eq. 20 with a complexity $\Theta(d \log d)$ rather than a standard $\Theta(d^2)$ procedure. In general, this algorithm produces the unnormalized coefficients defined in Eq. 20. To normalize them with respect to the degree, we must apply a factor, for instance, $2/d$. Thus, let us redefine the quantities

$$\bar{c}_j(\tau, \boldsymbol{x}) \leftarrow \frac{2}{d} \bar{c}_j(\tau, \boldsymbol{x}).$$

3. Finally, $\Phi_w(\tau; \boldsymbol{x})$ in Def. 4 is approximated by $P(\tau; \boldsymbol{x}, d)$ such that $P(0, \boldsymbol{x}; d) = \Phi_w(0; \boldsymbol{x})$, i.e. the constant of integration in Def. 4 is the conditional quantile $\tau = 0$. All of the above leads us to the final Clenshaw-Curtis expression used in the NAM,

$$P(\tau, \boldsymbol{x}; d) = \tau\left(\frac{\bar{c}_0(\tau, \boldsymbol{x})}{2} - \sum_{k=1}^{d/2} \frac{\bar{c}_{2k}(\tau, \boldsymbol{x})}{4k^2 - 1}\right) + \Phi_w(0; \boldsymbol{x}). \quad (21)$$

Note that Eq. 21 has a τ dependency on all the coefficients of $\bar{c}_k(\tau, \boldsymbol{x})$, which comes from the fact that the P depends on τ, because the nodes in Eq. 19 also originally depend on the quantile τ. This dependency will avoid to have a certain values of quantiles while ours methods can ensure - in machine precision - that the crossing quantile phenomenon does not appear because it approximates ϕ_w independently on τ.

3.2 Related Work

Building generic monotonic functions poses an important problem in several areas (Archer and Wang, 1993; Sill, 1998; Daniels and Velikova, 2010; Gupta et al., 2016; You et al., 2017; Wang et al., 2020). In the econometric field there are works such as (Koenker and Ng, 2005), which precisely explore the incorporation of monotonicity constraints and the estimation of the whole quantile function at once. However, as (Feldman et al., 2021) states, these models, such as the VQR in (Carlier et al., 2020), assume linearity, which in some contexts is a strong assumption. Differently, in the current approach, we are estimating the partial derivative of the quantile function by means of a truncated Chebyshev polynomial. Moreover, the proposed technique allows us to guarantee that the resulting quantile function is always partial monotonous in the desired quantiles even our proposal is using internally a deep learning model. Therefore, our proposal goes beyond modelling the conditional quantile function and not only constitutes a crossing quantile phenomenon solution but it allows us to estimate the partial derivative of the quantile function by means of a neural network.

The closest neural network method in the literature to the proposed method is the one we will explain hereafter. Differently, this literature method approximates the non-partially derivative, which in consequence cannot be applied to approximate a quantile function as we desire, but it has several similarities in the way the derivative is approximated as we will explain hereafter.

Generically, this method is a deep learning approach to building a monotonic function $H\colon \mathbb{R}^D \to \mathbb{R}$, called the Unconstrained Monotonic Neural Network (UMNN) proposed in (Wehenkel and Louppe, 2019). The UMNN estimates the derivative of that function as

$$H(z) = H(0) + \int_0^z h(t)\, dt, \quad (22)$$

where the integral in Eq. 22 is approximated using the Clenshaw-Curtis quadrature (Clenshaw and Curtis, 1960).

Furthermore, this derivative can be approximated by means of a neural network $\hat{h}\colon \mathbb{R}^D \to \mathbb{R}_+$, whose output is restricted to strictly positive values. Therefore, when $h(z) \coloneqq \frac{\partial H}{\partial z}(z) > 0$, the $H(z)$ behaves as a monotone function with respect to z, if $\hat{h}(z) = h(z)$.

The main novelty of our method, compared to the UMNN, is that we build a partial monotonic function, which allows us to approximate a quantile function. Further, the non-dependence of roots regarding τ (as it can be seen comparing Eq. 6 and Eq. 19) ensures that, in these points, the quantile function is always partial monotonic (according to Section 2.3).

On the whole, our goal is to show how the proposed solution solves the crossing quantile phenomenon while maintains (or even improves) the performance with respect to other existing alternatives. This can be extended to other fields where crossing quantiles is critical such as Reinforcement Learning (RL) Dabney et al. (2018c,b); Yang et al. (2019); Zhou et al. (2020) as future work.

4 EXPERIMENTS

In this section, we show the performance of the proposed model compared to the non-always monotonic approaches such as the IQN or the NAM. Furthermore,

Table 1: Minimum and maximum, [min, max], of the crossing quantile numbers over all the test folds proposed in Hernández-Lobato and Adams (2015). * denotes the number of crossing quantiles evaluating the roots quantiles.

	Housing	Concrete	Energy	Kin8nm	Naval	Power	Protein	Wine	Yacht
N	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**
IQN	**[0, 0]**	[0, 1416]	[75, 3007]	[0, 834]	[1673, 104689]	[0, 4058]	[57554, 87096]	**[0, 0]**	[281, 5935]
IQN-P	**[0, 0]**	**[0, 0]**	[11, 2583]	[0, 782]	[1072, 123594]	[0, 2539]	[42461, 77603]	**[0, 0]**	[22, 2852]
IQN-D	[0, 55]	[0, 1541]	[733, 8259]	[0, 3959]	[6246, 263553]	[175, 72813]	[68210, 113867]	[0, 2131]	[931, 7632]
PCDN	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**	**[0, 0]**
NAM	**[0, 0]**	[0, 1201]	[2152, 22760]	**[0, 0]**	[0, 0]	[0, 433]	[2090, 11409]	**[0, 0]**	[150, 2630]
Ours-q_0	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***
Ours-Mean	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***	**[0, 0]***

Table 2: Comparison of the Log-Likelihood sum of the test set over all of the train-test folds proposed in Hernández-Lobato and Adams (2015) for all the QR-based models. The bigger the better.

	Housing	Concrete	Energy	Kin8nm	Naval	Power	Protein	Wine
IQN	-703.2 ± 228.4	-1589.0 ± 228.3	-1092.7 ± 127.9	-9819.8 ± 398.4	-6358.5 ± 1343.0	-6499.3 ± 633.4	-27923.5 ± 614.1	-2040.5 ± 277.3
IQN-P	-613.2 ± 164.3	-1202.6 ± 157.1	-1012.3 ± 106.5	-9579.4 ± 699.5	-7868.2 ± 2072.1	-6320.3 ± 819.7	-29960.0 ± 2204.8	-2094.6 ± 318.8
IQN-D	-1156.7 ± 178.7	-2515.7 ± 192.4	-1839.4 ± 156.0	-18766.6 ± 1116.2	-18329.8 ± 5180.0	-18449.6 ± 1000.6	-90558.1 ± 5334.5	-3990.9 ± 309.9
PCDN	-745.4 ± 139.7	-1398.7 ± 226.4	-1133.6 ± 141.8	-9688.2 ± 540.5	-5664.5 ± 128.6	-8317.0 ± 343.6	-36599.9 ± 2961.3	-1898.7 ± 253.1
NAM	-697.4 ± 216.0	-1571.2 ± 202.7	-1461.7 ± 122.3	-9749.8 ± 483.5	-5372.5 ± 93.1	-8503.4 ± 250.5	-42807.7 ± 3063.3	-1928.7 ± 239.2
Ours-q_0	$\mathbf{-421.9 \pm 114.2}$	$\mathbf{-852.8 \pm 148.9}$	$\mathbf{-722.3 \pm 112.4}$	$\mathbf{-5355.8 \pm 305.3}$	$\mathbf{-4033.0 \pm 139.8}$	$\mathbf{-4521.1 \pm 155.9}$	$\mathbf{-25600.9 \pm 725.1}$	$\mathbf{-1201.6 \pm 157.4}$
Ours-Mean	$\mathbf{-407.9 \pm 111.9}$	$\mathbf{-808.7 \pm 122.8}$	$\mathbf{-736.4 \pm 103.7}$	$\mathbf{-5309.6 \pm 346.4}$	$\mathbf{-4033.5 \pm 102.5}$	$\mathbf{-4494.5 \pm 145.8}$	$\mathbf{-25596.0 \pm 924.6}$	$\mathbf{-1239.1 \pm 154.2}$

in the Appendix section, an always monotonous model based on restricting the weights - denoted as PCDN - is proposed and added too in the comparison. The goal is to verify that imposing the monotonicity constrain does not lead to a clearly worse predictive model.

In short, the different approaches analysed are:

The deep heteroscedastic Normal distribution (N) The conditional normal distribution will be considered. Given that this quantiles are calculated from the parametric conditional distribution as $F(\tau, \boldsymbol{x}) = \mu(\boldsymbol{x}) + \sigma(\boldsymbol{x})\sqrt{2} \cdot \text{erf}^{-1}(2\tau - 1), \tau \in (0, 1)$ where τ is the desired quantile and erf^{-1}, they will not cross. We will denote this approach as *N*.

The Implicit Quantile Network (IQN) Following Section 1, we are going to consider the IQN model. In particular, Tagasovska and Lopez-Paz (2018) proposes to add a regularization term with respect to the derivative as $\max\left(-\frac{\partial \phi_w(\boldsymbol{x},\tau)}{\partial \tau}\right)$ to alleviate the crossing quantile phenomena. We will note this alternative optimisation process as *IQN-D*. On the other hand, we will consider another regularisation term that penalises quantiles when crossing quantile phenomena appears as $\max(0., \phi_w(\tau_1; \boldsymbol{x}) - \phi_w(\tau_2; \boldsymbol{x}))$ for each $\tau_1 < \tau_2$. We will refer to this approach as *IQN-P*.

The Partial Constrained Dense Network (PCDN) The main idea of that model is to force a *selected* subset of *weights* to be only positive combined with only considering ReLU activation functions (Glorot et al., 2011). As it is described in the Appendix Section, the corresponding selected neurons will be increasing functions and can be used to produce a quantile function that is always partial monotonic.

The Not Always Monotonic model (NAM) Similarly to our main proposed model, NAM has two different functions to optimise: The ϕ_w and the K. In that case, K corresponds to the q_0 value as a general shift of all the quantiles. The predicted quantile function does not ensure that always will avoid crossing quantile phenomenon but will be added to the comparison to analyse their difference.

Our model (Our) The main proposal of the article can be defined as a partial monotonous function. All alternative selections of C_0 described in Section 2.2 are considered for the following comparisons.

4.1 Data Sets and Experiment Settings

All experiments are implemented in TensorFlow (Abadi et al., 2015) and Keras (Chollet et al., 2019), running in a workstation with Titan X (Pascal) GPU and GeForce RTX 2080 GPU. To ensure the strictly positive output values of ϕ_w in the proposed model, the final output will have a softplus function (Zheng et al., 2015) with a certain shift, in particular $\phi_w(\tau; \boldsymbol{x}) = 10^{-3} + \texttt{softplus}(NN(\tau; \boldsymbol{x}) + 10^{-5})$ where $NN(\tau; \boldsymbol{x})$ is the output of the neural network. In addition, according to the notation of Section A, to force the *selected* weights to be positive we will apply to it a ReLU function (Glorot et al., 2011) to their values. All internal activation functions will be ReLU. Furthermore,

all experiments will be trained using an early stopping training policy with 200 epochs of patience for all compared methods. All the details of the data sets used are detailed in the Appendix Section.

4.2 Experimental Results

Quantile regression models fit the conditional density distribution $p(Y \mid \boldsymbol{X})$ in a discrete manner. Thus, we want to study the performance between the different models in order to produce that distribution. However, not all models proposed in this work ensure monotonicity. Therefore, we will decide a common procedure to generate the forecasted likelihood for all of them to perform the following comparison. We calculate the number of predictive quantiles that fall within a discretization of the response space to predict for a thousand of equidistant $\tau \in (0, 1)$ points (as it is detailed in the Appendix section).

Evaluating the Number of Crossing Quantiles In Table 1 we show the minimum and maximum number of crossing quantiles taking into account all the folds. We can see that effectively, the normal distribution does not have crossing quantiles due to it comes from the quantile function formula. The PCDN never has any crossing quantile as stated in Appendix section A. Finally, as it is shown in Section 2.4, all the main proposed models (Our-q_0 and Our-Mean) do not have any crossing quantiles in the root quantile values.

Log-Likelihood Estimation In Table 2, we compare the log-likelihood adaptation for all the QR-based presented models for the eight different UCI problems. Each position is reporting the mean and standard deviation over the 20 splits previously defined in Hernández-Lobato and Adams (2015). We observe that the first five options are far from our proposed model in terms of likelihood, that is the best. Additionally, we show that PCDN and NAM are not significantly worst than IQN. However, we need to take into consideration that PCDN and our proposed model are the only presented options that we can be sure that they are monotonic functions when are evaluated in the desired root quantiles.

5 CONCLUSIONS

Quantile regression is an approach to estimate the conditional density distribution of the response variable in a discrete manner. However, when a single model tries to approximate several quantiles at once the order between them matters. When they are not predicted in an increasing way, then the predicted distribution is invalid. This phenomenon is called "crossing quantile". In this work, we introduce a method that solves this issue as shown in Figure 1.

In particular, the proposed method uses a deep learning model to approximate the partial derivative of an ending quantile function with respect to the quantile input. This partial derivative is imposed to be positive, thus, the ending function will be partially monotonic as desired. In Section 2, we described how to calculate this derivative by using a Chebyshev Polynomial approximation ensuring the monotonicity of the quantile function up to any desired precision.

Having computationally guarantees of the monotonicity of these models, our final goal was to verify if the imposed restrictions would adversely affect the performance of the model compared with a non-ensured quantile regression model. As we saw in Section 4, the proposed models outperforms in terms of likelihood estimation and yields non-quantile crossing forecasts with respect the other compared models.

The proposed model constitutes a generic deep learning wrapper for any architecture to build a partial monotonic quantile function avoiding the crossing quantile phenomenon.

ACKNOWLEDGEMENTS

We gratefully acknowledge the Industrial PhD Plan of Generalitat de Catalunya with BBVA Data and Analytics for funding this research. J.G. has been supported by NextGenerationEU within the Spanish national Recovery, Transformation and Resilience plan. The UB recognizes that part of the research was partially funded by RTI2018-095232-B-C21, SGR 1219.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

Archer, N. P. and Wang, S. (1993). Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 24(1):60–75.

Bondell, H. D., Reich, B. J., and Wang, H. (2010). Noncrossing quantile regression curve estimation. *Biometrika*, 97(4):825–838.

Brando, A., Rodriguez-Serrano, J. A., Vitria, J., and Muñoz, A. R. (2019). Modelling heterogeneous distributions with an uncountable mixture of asymmetric laplacians. In *Advances in Neural Information Processing Systems*, pages 8836–8846.

Cannon, A. J. (2018). Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. *Stochastic environmental research and risk assessment*, 32(11):3207–3225.

Carlier, G., Chernozhukov, V., De Bie, G., and Galichon, A. (2020). Vector quantile regression and optimal transport, from theory to numerics. *Empirical Economics*.

Chollet, F. et al. (2019). Keras (2015).

Clenshaw, C. W. (1955). A note on the summation of Chebyshev series. *Math. Tables Aids Comput.*, 9:118–120.

Clenshaw, C. W. and Curtis, A. R. (1960). A method for numerical integration on an automatic computer. *Numer. Math.*, 2:197–205.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018a). Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pages 1104–1113.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018b). Implicit quantile networks for distributional reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1096–1105. PMLR.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. (2018c). Distributional reinforcement learning with quantile regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2018d). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Daniels, H. and Velikova, M. (2010). Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Elliott, D. (1968). Error analysis of an algorithm for summing certain finite series. *J. Austral. Math. Soc.*, 8:213–221.

Feldman, S., Bates, S., and Romano, Y. (2021). Calibrated multiple-output quantile regression with representation learning. *CoRR*, abs/2110.00816.

Gal, Y. and Ghahramani, Z. (2015). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv:1506.02142*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Gupta, M., Cotter, A., Pfeifer, J., Voevodski, K., Canini, K., Mangylov, A., Moczydlowski, W., and Van Esbroeck, A. (2016). Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research*, 17(1):3790–3836.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *ICML*, pages 1861–1869.

Koenker, R., Chernozhukov, V., He, X., and Peng, L. (2017). *Handbook of Quantile Regression*. CRC press.

Koenker, R. and Hallock, K. F. (2001). Quantile regression. *Journal of economic perspectives*, 15(4):143–156.

Koenker, R. and Ng, P. (2005). Inequality constrained quantile regression. *Sankhya: The Indian Journal of Statistics*, 67(2):418–440.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

Majidian, H. (2017). On the decay rate of Chebyshev coefficients. *Appl. Numer. Math.*, 113:44–53.

Meinshausen, N. and Ridgeway, G. (2006). Quantile regression forests. *Journal of Machine Learning Research*, 7(6).

Newbery, A. C. R. (1974). Error analysis for polynomial evaluation. *Math. Comp.*, 28:789–793.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Sill, J. (1998). Monotonic networks. In *Advances in neural information processing systems*, pages 661–667.

Tagasovska, N. and Lopez-Paz, D. (2018). Frequentist uncertainty estimates for deep learning. *Bayesian Deep Learning workshop NeurIPS*.

Tagasovska, N. and Lopez-Paz, D. (2019). Single-model uncertainties for deep learning. In *Advances in Neural Information Processing Systems*, pages 6417–6428.

Trefethen, L. N. (2008). Is Gauss quadrature better than Clenshaw-Curtis? *SIAM Rev.*, 50(1):67–87.

Wang, Y., Xiao, C., Qin, J., Cao, X., Sun, Y., Wang, W., and Onizuka, M. (2020). Monotonic cardinality estimation of similarity selection: A deep learning approach. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1197–1212.

Wehenkel, A. and Louppe, G. (2019). Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems*.

Yang, D., Zhao, L., Lin, Z., Qin, T., Bian, J., and Liu, T.-Y. (2019). Fully parameterized quantile function for distributional reinforcement learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

You, S., Ding, D., Canini, K., Pfeifer, J., and Gupta, M. R. (2017). Deep lattice networks and partial monotonic functions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 2985–2993.

Zhang, W., Quan, H., and Srinivasan, D. (2018). Parallel and reliable probabilistic load forecasting via quantile regression forest and quantile determination. *Energy*, 160:810–819.

Zheng, H., Yang, Z., Liu, W., Liang, J., and Li, Y. (2015). Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4. IEEE.

Zhou, F., Wang, J., and Feng, X. (2020). Non-crossing quantile regression for distributional reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15909–15919. Curran Associates, Inc.

A PARTIAL CONSTRAINED DENSE NETWORK

PCDN has been used as reference method to be compared in the experiments, Section 4. Although it is not in the literature, PCDN is a natural approach to be considered in order to address the problem of crossing quantiles because it considers a classical approach of constraining the weights and activation functions to ensure that fully-connected neural network layers are monotonic. As an extension to this, PCDN constitutes a combination of partial monotone layers to ensure that the global model is partially monotone with respect to the quantile input as we will detail hereafter.

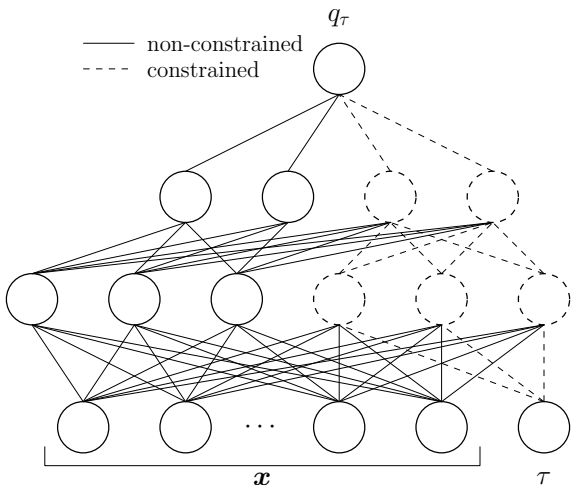

Figure 3: PCDN scheme.

The construction of neural networks with a monotonously increasing function of the inputs is a problem that has been identified in the literature, see, for instance, Sill (1998). These solutions were designed for completely dense models (i.e. those where all the neural network layers are fully-connected). These approaches were based on the following principle: a dense model with positive weights and with monotonically increasing activation functions gives a monotonously increasing function with respect to the input. Although, it is a very easy implementation to carry out, it has been found that, at the practical level, the impossibility of having weights of different signs makes the learning process complicated Wehenkel and Louppe (2019).

According to the notation in the Section 1, to solve the crossing quantile phenomenon it is enough to define a monotonic function f with respect to the input τ (that corresponds to the quantile value).

To built the model proposed in this section, we start with an IQN model, $\psi\colon [0,1] \times \mathbb{R}^D \to \mathbb{R}$, with only fully-connected layers by default.

Then, we select a desired part of the neurons for each hidden layer to ensure that ψ is monotonic with respect to τ; for instance, 50% of the neurons in each layer.

Next, we force that these selected neurons only have positive weights, their activation function are monotonically increasing, and they are only connected with the previous layer neurons that were also selected.

Finally, in the first layer, we only connect the weights that start from the input τ to the previously selected neurons of the following hidden layer as it is shown in Figure 3. We refer to these kind of models as Partial Constrained Dense Network (PCDN).

On the one hand, the inputs $\boldsymbol{x}$ are connected to the output with restricted and non-restricted weights. On the other hand, the τ value is connected with the output of the model only with restricted neurons to ensure it is a partial monotonic increasing function with respect to τ. By applying PCDN we restrict the number of constrained weights and neurons to only selected part of the neural network. This model could be considered as an extension of other proposed models in the literature such as Monotone Composite Quantile Regression Neural Network (MCQRNN) (Cannon, 2018), which predicts a fixed number of quantiles.

B DETAILS OF THE DATA SETS ON THE EXPERIMENTS

In this section, we show the performance of the proposed models compared to the baseline IQN. The goal is to verify that imposing the monotonicity of the PCDN or the main proposed model does not lead to a clearly worse predictive model.

Synthetic Glasses Data set The points in the back of the curves of Figure 1 corresponds to the mixture of two distributions. On the one hand, a noisy evaluation of $5 * \sin(x_i) + 0.5 + \epsilon$ is done where each x_i corresponds to evaluate 3000 equidistant points such that $0 < x_i < 3\pi$. The random noise correspond to a $\epsilon \sim \text{Beta}(\alpha = 0.5, \beta = 1)$. On the other hand, the second distribution of the mixture is the evaluation of 3000 points such that $\pi < x_j < 4\pi$ into $5 * \sin(x_j) + 0.5 + \varepsilon$. This time, the noise is defined by $\varepsilon \sim -\text{Beta}(\alpha = 0.5, \beta = 1)$. After that, all these values are normalized taking into account the maximum value of them. The regression problem consists in predicting, given an assigned value $x \in [0, 1]$, the corresponding $y \in \mathbb{R}$ generated as explained. These data have multi-modalities to encourage crossings of the predicted quantiles. The 50% generated data was considered as test data, 40% for training, and 10% for

validation.

The neural network architecture used for the ϕ_w of IQN and the ϕ_w and K_w of our method consists of 4 dense layers with output dimensions 120, 60, 10, and 1 respectively. Regarding the training time, all models took less than 4 minutes to converge.

UCI data sets In order to satisfy the goal of this section of checking the performance of IQN, PCDN, NAM and the main proposed models, we applied these models to 8 different UCI Machine learning data sets (Dua and Graff, 2017). These data sets are commonly used for various regression tasks. In particular, we used the 20 splits proposed in Hernández-Lobato and Adams (2015) and widely used in later works (Gal and Ghahramani, 2015; Lakshminarayanan et al., 2017).

Regarding the trained models, all of them share the same architecture for all the models: a single dense hidden layer of 200 neurons. However, as NAM and the main proposed model requires the architecture for ϕ_w and K_w, we decided to assign the half part of the neurons of the single hidden layer to each of them.

In addition, we will analyse the number of crossing quantiles that are obtained for each test set considering the 980 quantiles $[0.010, 0.011, 0.012, \ldots, 0.990]$.