Learning of an XOR Problem in the Presence of Noise and Redundancy

Denis Cousineau
Département de psychologie
Université de Montréal
C.P. 6128, Succ. Centre-Ville
Montréal, H3C 3J7 CANADA
E-mail: denis.cousineau@umontreal.ca

Abstract- **Recently introduced time-based networks represent an alternative to the usual strength-based networks. In this paper, we compare two instances of each family of networks that are of comparable complexity, the Perceptron and the race network when faced with uncertain input. Uncertainty was manipulated in two different ways, within channel by adding noise and between channels by adding redundant inputs. For the Perceptron, results indicate that if noise is high, redundancy must be low (or vice versa), otherwise learning does not occur. For the race network, the opposite is true: If both noise and redundancy increase, learning remains both fast and reliable. Asymptotic statistic theories suggest that these results may be true of all the networks belonging to these two families. Thus, redundancy is a non trivial factor.**

1. INTRODUCTION

Many connectionist networks are built around a strength-based representation in which 0 means an absence of input and 1 an input present. Recently, there was a demonstration that a network could be built around a time-based representation [1]. In this framework, stimuli are coded according to the moment they become accessible. Therefore, 0 means immediately available whereas ∞ means never presented. A time-based network tries to respond as fast as possible whether relevant information is present or not. The purpose of the learning rule is to discover the priority level of the inputs by increasing delays of connections.

We showed in [2] the similarities that exist between the Perceptron with the delta rule and a simple time-based network, the race network. As was demonstrated, the transmission rules and the learning rules are nearly identical in both versions. In one sense, this is not surprising because the two networks have exactly the same architecture (two layers, feed-forward). The surprise however came from the fact that the time-based version learned nearly twenty times faster a linearly separable problem, compared to a Perceptron with no hidden layer.

One objective of this paper is to see whether the race network also learns faster a non-linearly separable problem, compared to its equivalent counterpart, the Perceptron. However, a more important objective of this paper is to see if this speed of learning advantage persists in more realistic situations.

A physical system placed in a real-life situation is going to be fed with imprecise input. This imprecision has two different flavors. The first one is related to the quality of the input. There are few chances that perfect zeros and ones (or infinities and zeros, respectively, in the time-based representation) are sampled from the outside world. Instead, any real value number between these limits may be accessed. This first form of imprecision, which occurs within the connections (or equivalently, within the channels) will be called noise in the remainder of the paper. A simple way to simulate noise is to take the quantity that the connection would have sampled in the absence of noise (either 0 or 1 in a strength-based network) and blur this quantity by adding a random value to it.

The second form of imprecision can be referred to as between-channel uncertainty. In a complex enough system, the inputs are likely to consist of a large array of detectors. Numerous inputs are an advantage from an engineer point of view because it increases the sensitivity to external signals and reduces the sensitivity to internal failures. A disadvantage of having a large array of detectors is that only a subset of them might register the presence of a stimulus and that subset can be different from one experience with the stimulus to the other. As an analogy, think of the eye as a large array of light detectors. A static square will affect different parts of the retina, with possible overlap between stimulated

regions [3]. The key aspect behind all this is the notion of redundancy. Redundant inputs are a set of inputs whose purpose is to detect a specific attribute and many or all of them turn on when this attribute is present. Redundancy can be simulated by first getting the quantity that one connection would have sampled in the absence of noise and duplicate it a number of time. The corresponding network must have one input unit per duplicated signal. If noise is also present, then in a second step, each signal is blurred independently.

The main objective of this paper is to see whether strength-based and time-based networks can learn input-output associations when the inputs are redundant and noisy. The impact of redundancy was discussed in [4] for the strength-based networks; some of the following simulations will replicate their results. However, they used non noisy inputs. In what follows, we briefly review the race network and then perform simulations in which the amount of noise and the amount of redundancy are manipulated.

1.1. Perceptron and race network

As said, the transmission and the learning rule of the Perceptron and the race network are very similar. The transmission rules are given by $\mathbf{o} = \mathbf{a}.\mathbf{W}$ and $\mathbf{o} = \mathbf{a} \backsim \mathbf{W}$ respectively, which means:

$$o_j = \sum (a_i \times w_{ij}) \quad \text{vs.} \quad o_j = \bigvee (a_i + w_{ij}) \qquad (1)$$

where $\boldsymbol{o}$ is the output vector of the network, $\boldsymbol{a}$ is the input vector, and $\mathbf{W}$ is the connection matrix of the network. In the case of the Perceptron, $\mathbf{W}$ contains the weights of the connections. The operation . (dot) represents an inner product, that is, an operation where pairs of values are joined using × and where columns are aggregated using Σ. For the race network, connections represents "waits", a wait w_{ij} of 0 meaning that the i^{th} input is highly important for the j^{th} output and has high priority whereas ∞ has the opposite meaning. The operation $\backsim$ is a modified inner product that joins pairs of values using + and aggregates columns using $\bigvee$, the k_j^{th} smallest element. A component of the race network that is not visible in the above notation is the threshold vector $\boldsymbol{k}$ of the same size as the output vector. Thresholds are discussed next.

In general (and in the simulations that follows), a sigmoid function is applied to the output of a Perceptron so that the outputs are bounded between 0 and 1 and still derivable [5].

Learning in the race network occurs by penalizing the connections that contributed most heavily to the error, as indicated by a "teacher". Thus, it implements an algorithm similar to the Δ rule. The rule stipulates that $\Delta\mathbf{W} = \alpha\, \mathbf{a})+(\mathbf{o} - \mathbf{e})$ where α is a learning rate parameter and $\mathbf{e}$ is the expected output vector. The operation $)+($ is a modified outer product where pairs of values are joined using +. The update of the connection waits are then performed with $\mathbf{W} \xleftarrow{update} \mathbf{W} \bigvee (\mathbf{W} + \Delta\mathbf{W})$.

So far, the two networks have very similar mechanisms, resulting in very similar equations. In the following, we discuss the dissimilarities.

First, the race network uses a hard threshold in its decision rule, a quantity k_j for each output unit. Its role is to indicate how many channels must send information before the output unit is triggered. This quantity is set to one at the beginning of training and is adjusted only when an output missed, i. e. when it was not triggered but should have according to $\mathbf{e}$: $\Delta k = \omega\, Sign(\#\mathbf{a} - k)$, where $Sign(x)$ returns +1, -1 or 0 depending on whether $x > 0$, $x < 0$, or $x = 0$ respectively, $\#\mathbf{a}$ returns the number of inputs that were active at the time the error was detected and ω is a learning rate parameter for the thresholds ($0 < \omega < 1$).

The second difference is related to how the architecture must be modified so that these networks can learn non-linearly separable problems. For the Perceptron, the solution is well-known and consists in adding a layer of hidden units. The error given by the teacher is back propagated in the network so that the residual error is used on preceding layers.

For the race network, the solution consists in adding "clock" units at the input layer. These units are in no way influenced by the stimuli. However, at some time, they will turn on. The moment when this happens is arbitrary and we used 0 in all the simulations. In essence, these units simply indicate that time is passing and a decision can be based on this fact. As an illustration, we indicate in Figure 1 how the race network could solve an XOR problem [6].

The Perceptron has nice properties. It reduces the energy using gradient descent techniques. It is also an optimal classifier in the sum of square sense. By contrast, the properties of the race network are not known. It might perform some form of Lagrangian descent learning, but a demonstration is still to come.

2. SIMULATIONS

We first describe the training problem and then the networks and how they were made comparable. Afterwards, we describe how noise and redundancy were manipulated.

2.1. The training problem and its representation

The networks were all tested on the same problem, the XOR problem. It was chosen because it is well known and non-linearly separable. For a strength-based network, True is represented by 1 whereas False is represented by 0. For a time-based network, True is

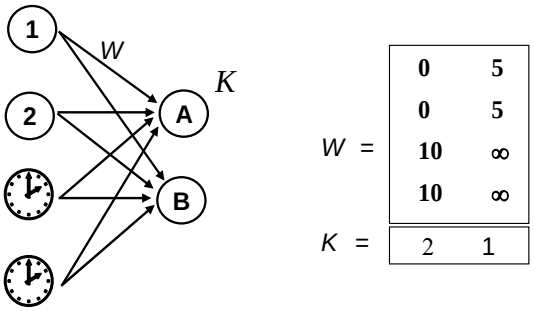

Fig. 1. How a race network can solve the XOR problem. The clocks are on at the onset of a trial. If present, the network immediately detects the presence of both input since the connections imposes no extra delays ($w_{11} = w_{21} = 0$ and threshold $k_1 = 2$). If only one input is on, the output B will be triggered by it after a delay of 5 (arbitrary units of time) imposed by the connection ($w_{12} = 5$ or $w_{22} = 5$). Finally, if no inputs are present, the clock units will activate a response after a delay of 10.

represented by 0 and False by ∞. This ∞ is meant to represents "never" but because it reduces the comparability of the two networks, we chose to use 1 instead of ∞. This means that an input unit that does not receive stimulation will nevertheless turn on after 1 arbitrary unit of time. This can be seen as a "spurious activation" or a false alarm.

To signal the two possible responses, two output units were used so that when one had to turn on, the other had to remain off, and vice-versa.

Because of the symmetry of the XOR problem, the exact same input-output sets can be used to train both networks. For example, the input {1,1} is seen as {True, True} for the Perceptron but as {False, False } for the race model. The only difference is that the responses are inverted: the Perceptron had to learn to activate the second output for the above input whereas the race network has to activate the first input.

2.2. The networks

In all the simulations, the race network used had two clock units but no hidden layer and two units on the output layer. The learning rate parameters were $\alpha = 0.5$ and $\omega = 0.1$. We tried various αs with no qualitative changes in the results. The Perceptron used had one hidden layer composed of 6 units. The learning rate parameter α was 0.5. We also tested a Perceptron with two hidden units so that the total number of units in the two networks was identical. However, such a Perceptron could not learn the XOR problem on nearly 25% of the simulations.

In the simulations, the order of presentation of the instances was random. Training was arbitrarily divided in epoch of ten trials. In all cases, it was interrupted after 5000 trials. To test that learning was successful, we used the following criterion: learning occurred if the sum of square error (SSE) between the network's output and the desired solution was reduced below 0.1 [7]. Because the race network tends to produce a very erratic learning curve (as seen in the following figures), the SSE obtained in a simulation were smoothed using a moving windows of 50 trials: The smoothed performance at trial *i* is the average of that trial, the 24 trials that follows and the 25 that precedes it. All the simulations were replicated a hundred times.

2.3. Redundancy and noise

Two levels of redundancy, noted with the letter ρ, were tested: either no redundancy ($\rho = 1$) or high redundancy ($\rho = 10$). In the no redundancy condition, the two input dimensions were not duplicated and so the networks tested had only two input units. In the high redundancy condition, the two inputs were duplicated ten times. Thus, there were 20 input units. For the race network, there were always two additional input units, the clock units, which were set to 0 on all trial, irrespective of the input presented. We also ran simulations where the clock units were also duplicated ρ times, but this did not change the results presented in the next section.

Noise was added independently to each input using an exponential distribution. When the input was 0, an exponentially distributed random value was added. It was subtracted when the input value was 1 so that the resulting input value tented to be between 0 and 1. Values that exceeded 1 were truncated to 1 and values below 0 were truncated to 0. The exponential distribution is specified by a single parameter η, which is both the mean and the standard deviation of the population. It is lower bounded at zero but extend to ∞ (hence, the truncation procedure described above).

We tested two noise conditions ($\eta = 0.05$ and $\eta = 0.10$). The $\eta = 0.10$ condition will be called the high noise condition, even though the average value is close to zero (being 0.1) and the perturbation is between 0 and 0.2 on 86% of the trials.

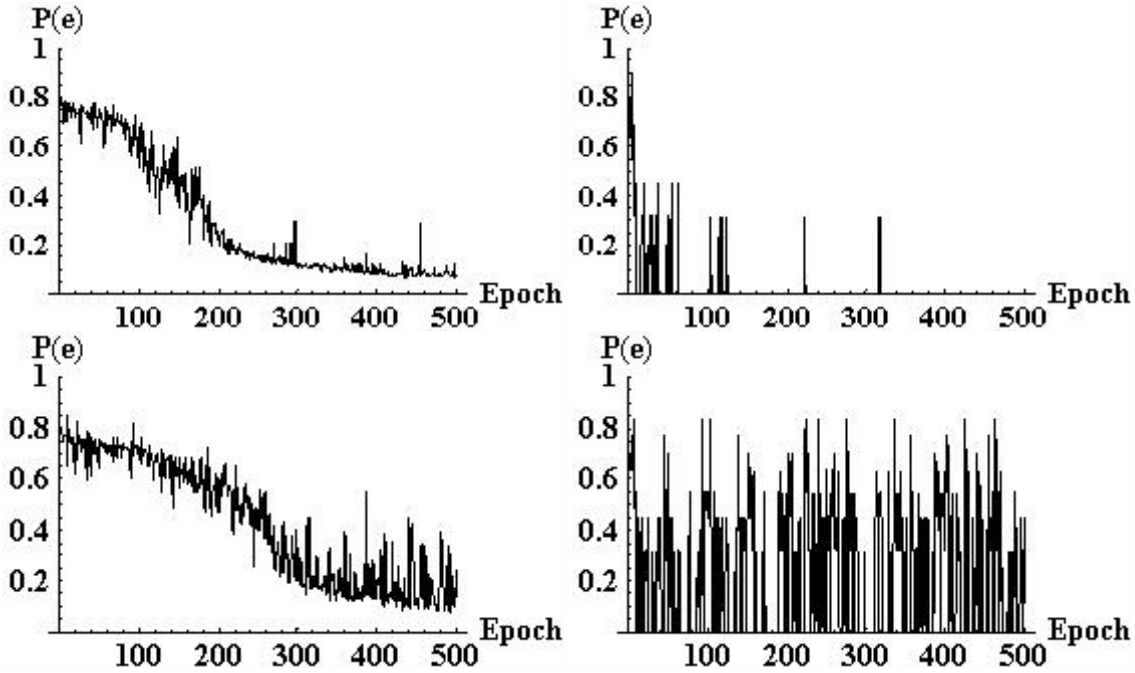

Fig. 2. Comparison of the Perceptron (left column) and the race network (right column) on the XOR problem when there is no redundancy (ρ = 1). Noise is either small (η = 0.05, top row) or high (η = 0.10, bottom row). One epoch represents 10 trials.

3. Results of the simulations

We present the results in the no redundancy conditions (ρ = 1) followed by the high redundancy conditions (ρ = 10).

3.1. Learning with noise but no redundancy

Figure 2 presents the networks' performances for a typical simulation. The vertical axis represents the mean SSE for a given epoch. Table 1 shows the average performance across a hundred replications.

We first look at the low noise conditions (top row): For the Perceptron, the average number of iterations to reach the criterion is near 2300 and the percent of successful learning is 98%. As of the race network, it does a few scattered errors, as seen in Figure 2. As training extend further, these errors became rarer. Learning is very rapid (270 trials) and very reliable (100% of successful learning). Because the race network is a winner-take-all network, the pikes are often the results of a single erroneous response.

The major result of this section is in the bottom row: with high level of noise, the race network suddenly stops learning. Various learning parameters were tested with no change. The percent of successful learning dropped to 10%. By contrast, the performance of the Perceptron was unaffected by this amount of noise. As will be seen next, a totally different picture emerges when redundancy is introduced.

The fact that the race network does not learn an XOR problem with high noise, whatever the parameters, suggests that it is a limit of the whole framework rather than an accidental limit of our simulations. In the general discussion, we suggest an approach to understand this limit.

3.2. Learning with noise and redundancy

Figure 3 presents the results of a typical simulation when redundancy is high (ρ = 10). As seen, the results are very different from those of Figure 2.

First, the race network learned in all noise conditions. Going from low noise (η = 0.05) to high noise (η = 0.10) did slow down learning: the number of iterations roughly doubled as noise was doubled. However, learning was very robust: the criterion was reached in over 99% of the simulations. By contrast, the Perceptron behave in a totally different manner. For low noise, learning is faster (average of 1400 trials) and moderately reliable (76% of the simulations found a solution). Faster learning in the no noise condition was predicted by [4] who studied the role of redundancy in the presence of non noisy input. In the high noise condition, only 51% of the simulations found a solution in less than 5000 trials (in which case learning was moderately fast with an average near 1700 trials).

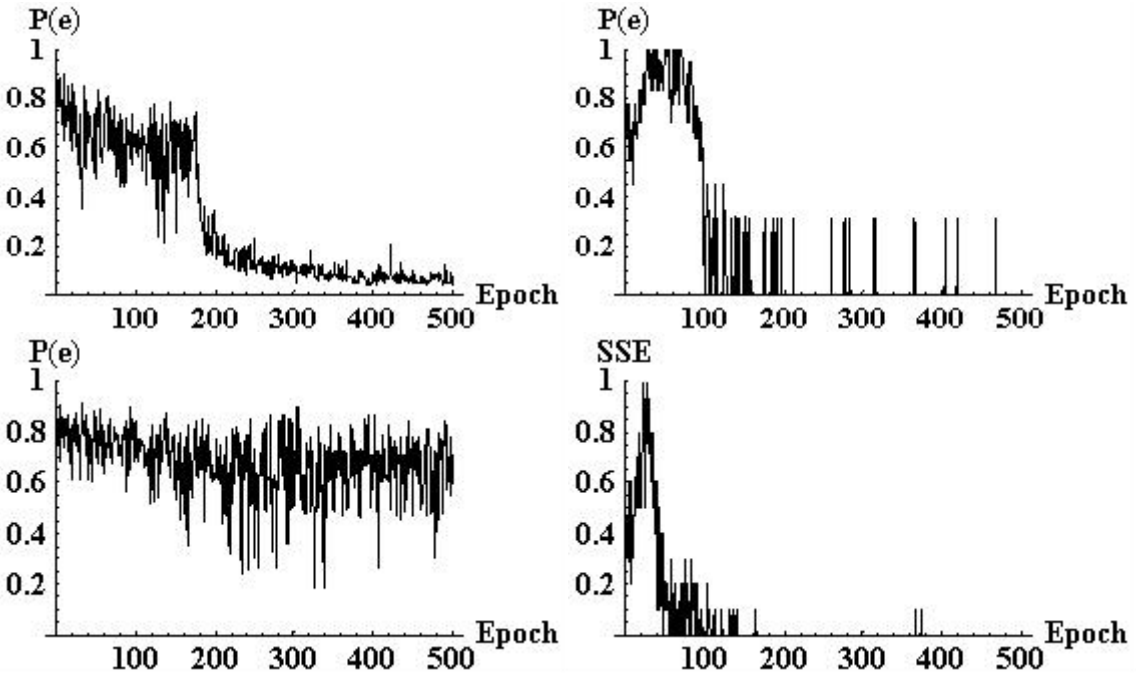

Fig. 3. Comparison of the Perceptron (left column) and the race network (right column) on the XOR problem when redundancy is high ($\rho = 10$). Noise is either small ($\eta = 0.05$, top row) or high ($\eta = 0.10$, bottom row). One epoch represents 10 trials.

These simulations raise the role of uncertainty in understanding neural networks' behavior. It does have a quantitative impact on speed of learning: the Perceptron learned almost two times faster in the high redundancy condition whereas the race network was three times slower. More importantly however is the finding that learning was unreliable in certain cases involving both noise and redundancy. Given a high level of noise, the race network was unreliable when redundancy was low whereas the Perceptron was unreliable when redundancy was high. This interaction is maybe the key difference between the two types of networks. We next look at theorems related to asymptotic distributions of noise than might suggest an explanation.

4. General Discussion

The role of many neural networks is to find a separation between the stimuli. Non-linearly separable problems are difficult because there is no single separation in the original input space. This is why hidden layers and non linear functions are required. The race network, owing to its clock units, can implement more than one separation, so it does not need a hidden layer to learn the XOR problem.

When the input is noisy, its representation in the input space is changed from a point to a cloud whose density is higher near the center but which may extend far in all directions. The separations are no longer absolutely reliable but if the extend of the cloud is not too large,

Table 1. Mean number of iteration required (max. 5000) to reach an amount of errors (SSE) below 0.01 plus or minus the standard error of the mean and percentage of successful learning between parenthesis.

	$\rho = 1$		$\rho = 10$	
	Perceptron	*Race Network*	*Perceptron*	*Race Network*
$\eta = 0.05$	2346 ± 75 (98%)	268 ± 56 (100%)	1724 ± 79 (76%)	717 ± 49 (99%)
$\eta = 0.10$	2715 ± 51 (100%)	4461 ± 65 (10%)	1665 ± 130 (51%)	1205 ± 55 (100%)

the separations may be reliable most of the time.

We can implement the clouds with a distribution function and measure its extent using the standard deviation. In the above simulations, we specified the standard deviation of the noise at the input (η). We thus want to know the variance at the output and check whether the limited domain of the representation adopted [0..1] is large enough to contain one (or two for the race network) useful separation.

Following [8], let $L(\mathbf{o})$ be the distribution function of the outputs. From Eq. 1, we have for the Perceptron that $L(\mathbf{o}) = L(\mathbf{a} \cdot \mathbf{W})$. Thus, we are looking for the distribution of a weighted sum, so that the distribution function L is with respect to summation, that we note $L_{\Sigma}(\mathbf{a} \times \mathbf{W})$. Assuming that the number of connections is large (this was not quite true in the simulations so the following has only a heuristic value), the question is thus to find the asymptotic distribution with respect to summation. The solution has been known for decades and is given by the Central Limit Theorem [9]. More importantly, it states that the total variance will be the sum of the input variances. If all the inputs have the same standard deviation η and their number is given by ρ, then the final standard deviation is

$$\eta_{\Sigma} = \eta \times \sqrt{\rho} \qquad (2)$$

Hence, the size of the cloud increases with the number of redundant input. Because the range of values is limited to be between 0 and 1, there is a point where no separation is possible.

This informal reasoning explains why the Perceptron could not learn the XOR when redundancy was high: the output standard deviation was $\sqrt{\rho}$ times larger than in the no redundancy condition, probably spanning the whole range.

For the race network, things are different. $L(\mathbf{o})$ is given by $L(\mathbf{a} \curlyvee \mathbf{W})$ which is the distribution with respect to minima $L_{\vee}(\mathbf{a} + \mathbf{W})$. Assuming that the number of connections is large, the solution to this problem is given by the Extreme Limit Theorem [10]. It states that the final standard deviation has the following relation to the standard deviation of each input (assumed to be all equal) and the redundancy:

$$\eta_{\vee} = \frac{\eta}{\sqrt[\gamma]{\rho}} \qquad (3)$$

where γ depends on the specific nature of the noise distribution (its "signature", [10]; in the simulations with exponential noise, γ is 1). The important point is that ρ cancels the effect of noise. Noise can be increased, as long as redundancy is also increased, the net effect is equivalent to a low noise, low redundancy condition. In this framework, redundancy acts like a filter.

The race network could not learn in the $\eta = 0.10$, $\rho = 1$ condition because it had to maintain two separations, one more than the Perceptron. However, in the $\rho = 10$ conditions, noise was reduced ten times.

This paper suggested how noisy input could be modeled in a large scale network using between-channel and within-channel uncertainty. Faced with redundancy and noise, the race network and the Perceptron behaved in drastically different ways. This was demonstrated with simulations, and Equations 2 and 3 taken from asymptotic statistics seem to indicate the generality of this finding. If this network is to be used as a model of the human cognition, then it shows that a central question that ought to be examined is whether there are redundancy in the brain or not.

ACKNOWLEDGMENTS

This research was supported by the *Fonds de Recherche sur la Nature et les Technologie du Québec* and the *Conseil de la Recherche en Sciences Naturelles et en Génie du Canada*.

REFERENCES

[1] D. Cousineau, "Merging race models and adaptive networks: A parallel race network", *Psychonomic Bulletin & Review*, vol. 11, pp. 807-825, Nov. 2004.

[2] D. Cousineau, G. L. Lacroix, and S. Hélie, "Redefining the rules: Providing race models with a connectionist learning rule", *Connection Science*, vol. 15, pp. 27-43, Mar. 2004.

[3] F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of the brain mechanisms", Washington, DC: Spartan, 1963.

[4] Y. Izui, and A. Pentland, "Analysis of neural networks with redundancy", *Neural Computation*, vol. 2, pp. 226-238, Jan. 1990.

[5] G. E. Hinton, "How neural networks learn from experience", *Scientific American*, pp. 145-151, Sep. 1992.

[6] R. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry*, Cambridge, Mass: MIT Press, Jun. 1969.

[7] R. C. O'Reilly, "Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm", *Neural Computation*, vol. 8, 895-938, Nov. 1996.

[8] J. Galambos, *The Asymptotic Theory of Extreme Order Statistics*, New York: John Wiley and Sons, 1978.

[9] W. Feller, *An introduction to probability theory and its application*, 2nd ed, Vol. 1, New York: John Wiley and son, 1957.

[10] D. Cousineau, V. Goodman and R. M. Shiffrin, "Extending statistics of extremes to distributions varying on position & scale", *Journal of Mathematical Psychology*, vol. 46: 431-454, Jun. 2002.