Efficient training of neural networks with interval uncertainty

Jonathan C. Sadeghi, Marco De Angelis and Edoardo Patelli,*

Institute for Risk and Uncertainty, University of Liverpool, United Kingdom
*Corresponding author: edoardo.patelli@liverpool.ac.uk

Abstract

In this paper we attempt to build upon past work on Interval Neural Networks, and provide a robust way to train and quantify the uncertainty of Deep Neural Networks. Specifically, we propose a back propagation algorithm for Neural Networks with constant width predictions. In order to maintain numerical stability we propose minimising the maximum of the batch of errors at each step. Our approach can accommodate incertitude in the training data, and therefore adversarial examples from a commonly used attack model can be trivially accounted for. We present preliminary results on a test function example. The reliability of the predictions of these networks are guaranteed by the non-convex Scenario approach to chance constrained optimisation. A key result is that, by using minibatches of size M, the complexity of our approach scales as $\mathcal{O}(MN_{iter})$, and does not depend upon the number of training data points as with other Interval Predictor Model methods.

Keywords: Machine Learning, Imprecise Probability, Uncertainty Quantification, Neural Networks, Interval Predictor Models

1 Introduction

In recent years Deep Learning using Artificial Neural Networks has emerged as a generalised Machine Learning tool which has revolutionised supervised learning, reinforcement learning, as well as finding many applications in the field of Engineering, most often as efficient surrogates for large models [1]. In all fields, but particularly in safety critical engineering applications, it is essential to quantify the uncertainty of the Neural Network. The simplest and most widely used approaches attempt to quantify this uncertainty by analysing the mean squared error or explained variance (r^2) of the Neural Network on a test set. However, this approach does not robustly bound the predictions of the Neural Network. Subsampling techniques like bootstrapping can improve the uncertainty quantification, but the results are in general still unsatisfactory [2]. Bayesian Neural Networks (assisted by variational inference), where the weights are modelled as random variables, have emerged as the most popular tool for giving a prediction of the uncertainty of the Neural Network which is acceptable to statisticians [3]. Popular implementations in PyMC3 and Edward, amongst others, have made this approach viable in an industrial context [4]. However, many assumptions are necessary in order to apply this approach, for example the weights are commonly assumed to have a Gaussian prior and the likelihood function is also assumed to be Gaussian. In addition, Variational Inference is more complex to implement than the Backpropagation which is used by most Machine Learning engineers. Robust Neural Networks apply Bayesian model selection to discrete sets of Neural Networks, and whilst a prior is not required explicitly for the weights, other assumptions are made [5].

Interval Predictor Models are a recently developed machine learning technique for supervised learning which make interval predictions with guaranteed accuracy [6]. In other words, for every input, x_i an Interval Predictor Model would predict $\bar{y}(x_i)$ and $\underline{y}(x_i)$ instead of just y_i. The technique relies upon the solution of chance constrained convex optimisation programs by the scenario technique [7]. The scenario technique is a method of approximately solving optimisation problems with probabilistic constraints by considering a random sample. Scenario Optimisation is easier to use in practice than similar methods in statistical learning theory since no knowledge of the VC-dimension is required. The first published software implementation of Interval Predictor Models was made available in the open source OpenCossan software [8]. Crucially the scenario approach only applies when the data distribution of the input variables is stationary. This is an important limitation of the work currently. A key advantage over other machine learning techniques is that interval training data fits into the scenario optimisation framework coherently [9]. This can be seen as similar to the attack model of adversarial examples considered in [10], however our framework also permits robustness against uncertainty in training outputs.

Neural Networks with interval outputs were first proposed in [11], and further described in [12]. In

these papers the learning takes place by identifying the weights W, which solve the following program:

$$\arg\min_{\bar{W},\underline{W}} [\bar{y}(x_i) - \underline{y}(x_i) : \bar{y}(x_i) > y_i > \underline{y}(x_i) \forall i], \tag{1}$$

where $\bar{y}(x)$ and $\underline{y}(x)$ are obtained from two independent Neural Networks, such that $\bar{y}(x)$ and $\underline{y}(x)$ are the output layers of networks, where layer i is given by f_i

$$f_i = \tanh(W_i f_{i-1}), \tag{2}$$

where f_i is a vector (which is the input vector when $i = 0$) and W_i is the ith weight matrix. In practice this problem is solved by using a mean squared error loss function with a simple penalty function to model the constraints. Our experiments revealed that this penalty method requires careful choice of hyper-parameters to guarantee convergence, and hence we struggled to repeat the results from the paper. These Neural Networks act in a similar way to Interval Predictor Models, however the Interval Neural Networks do not include an assessment of the prediction accuracy (although this can be predicted to a reasonable degree of accuracy by using a test set). In [13] the Scenario Approach was extended to non-convex optimisation programs, and hence applied to a single layer Neural Network, with a constant width interval prediction, which was trained using the interior-point algorithm in Matlab. In other words the following program is solved:

$$\arg\min_{W,h} [h : |y_i - \hat{y}(x_i)| < h \forall i], \tag{3}$$

where h is a real number, and $\hat{y}$ represents the central line of the prediction obtained from the same network specified by Eqn. 2. In [14] the non-convex scenario approach is extended to problems solved with the sampling and discarding technique where the most restrictive constraints are optimally removed [15].

In this paper we propose a back propagation algorithm for Neural Networks with constant width predictions, and show how this can be efficiently used to create deep Interval Neural Networks. In order to obtain maintain numerical stability we avoid using the penalty method proposed by [11], and instead propose minimising the maximum squared error of the whole batch of gradients at each step. We present preliminary results on a test function example. On modern architectures using larger batches is desirable as this allows the step size to be increased, whilst maintaining a constant computation time for each step by making use of parallelisation. The predictions of these networks are guaranteed by the non-convex Scenario approach. We explain how interval training data can be accommodated in this paradigm [9].

2 Proposed Architecture

2.1 Overview

Our architecture is based on the model proposed in [13]. By solving

$$\arg\min_{W,h} [h : |y_i - \hat{y}(x_i)| < h \forall i], \tag{4}$$

we are trying to find the Neural Network which minimises the maximum error rather than the mean squared error. In order to solve this problem efficiently we propose a modified backpropagation method for the Neural Network, i.e. gradient based optimisation. Eqn. 4 is chance constrained but is much easier to solve than Eqn. 1. To evaluate the gradient we compute the mean squared error for each data point in the training set, and then follow the gradient for the point with maximum error. Our algorithm is described in further detail in Algorithm 1.

It is immediately apparent that our algorithm is more costly than the standard back propagation method, since our method costs $\mathcal{O}(NN_{iter})$, compared to a standard stochastic gradient descent cost of $\mathcal{O}(N_{iter})$. Fortunately, the cost is not as high as it would initially seem as modern GPU architectures allow the some of the calculations to be parallelised. However, the largest GPU architectures have several thousand cores, so for data sets with millions of data points our approach would not be scalable. We should also add that the N_{iter} required for convergence in both algorithms is not necessarily the same, as this depends on the variance of the gradient at each step.

In order to increase the speed of convergence the weights could be initialised to those obtained by training the network with a mean squared error loss function.

Algorithm 1 Maximum error backpropagation method

Input: Training data pairs (x_i,y_i for $i = 1, ..., N$)
  Randomly initialise weight tensor and h.
  for $i = 1, .., N_{iter}$ **do**
    for $j = 1, .., N$ **do**
      Compute Prediction $\hat{y}(x_j)$
      Compute error$_j = |y_j - \hat{y}(x_j)|$
    end for
    Set $k = \arg\max_j \text{error}_j$
    Use gradient of loss function to update W and h ($W_i \leftarrow W_i + \eta \frac{\partial (y_k - \hat{y}(x_k))^2}{\partial W}$)
  end for
  for $j = 1, .., N$ **do**
    Compute Prediction $\hat{y}(x_j)$
    Compute error$_j = |y_j - \hat{y}(x_j)|$
  end for
  Set $h = \max_j \text{error}_j$
Output: Weight tensor and h

2.2 Scalability Improvement

In order to reduce the computational cost of the algorithm we propose the use of minibatch stochastic gradient descent [16]. In other words, M training data points are randomly selected for each step. Our algorithm is described in Algorithm 2.

Algorithm 2 Maximum error backpropagation method, using minibatches

Input: Training data pairs (x_i,y_i for $i = 1, ..., N$)
  Randomly initialise weight tensor and h.
  for $i = 1, .., N_{iter}$ **do**
    Generate set, B, of M random numbers, sampled without replacement between 1 and N
    Set $k = \arg\max_{j \in B} |y_j - \hat{y}(x_j)|$
    Use gradient of loss function to update W and h ($W_i \leftarrow W_i + \eta \frac{\partial (y_k - \hat{y}(x_k))^2}{\partial W}$)
  end for
  Generate set, B, of M random numbers, sampled without replacement between 1 and N
  Set $h = \arg\max_{j \in B} |y_j - \hat{y}(x_j)|$
Output: Weight tensor and h

It is clear that using minibatches reduces the cost of our algorithm to $\mathcal{O}(MN_{iter})$, which is a potentially vast improvement when $N >> M > 1$. However there is no guarantee that the minibatch selected at each step contains the true maximum of the error, and therefore we may not be following the true gradient, and hence may be solving the wrong optimisation program. However it turns out that this is not problematic, as small minibatch sizes can closely replicate the expectation of the true loss function (see Appendix A for proof). For large N we find that by using a minibatch of size M, we solve a problem equivalent to minimising the $\frac{1}{M}$-th percentile of the empirical cumulative distribution function of the error for the whole training data set. This is not as problematic as it seems, due to developments which are presented later in the paper.

3 Reliability Assessment

3.1 Convex case

For the benefit of readers not familiar with the scenario approach to chance constrained optimisation we will first present an overview of the theory of scenario optimisation in the convex case.

A chance constrained optimisation program is an optimisation program of the following form

$$\arg\min c^T x : P\{f(x, \delta) > 0\} \leq \epsilon, \tag{5}$$

where δ is a random variable, x is the design variable, c is a constant, and ϵ is a parameter which constrains how often the constraints may be violated. This program can be approximately solved using

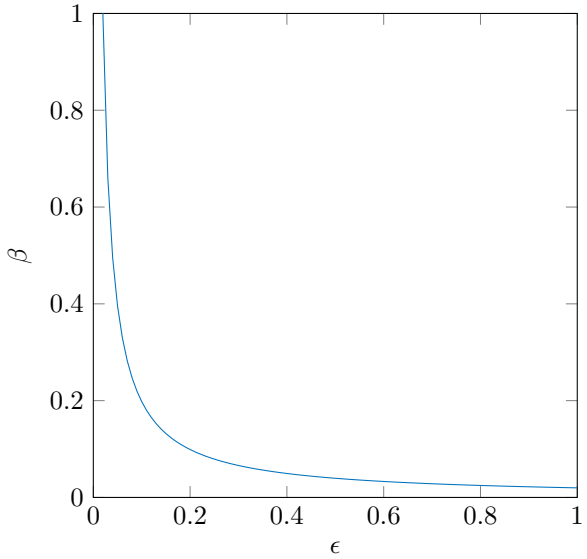


Figure 1: Plot of Eqn. 8 for $N = 100$ and $n = 2$.

the scenario approach: the problem is solved for a random sample of the constraints, i.e. we solve

$$\arg\min c^T x : f(x, \delta^i) \leq 0, i = 1, ..., N, \tag{6}$$

where δ^i represents the ith sampled value of the constraints [7].

Intuition tells us that the solution will be most accurate when the dimensionality of the design variable is low and we take as many samples of the constraints as possible (in fact, an infinite number of sampled constraints would allow us to reliably estimate $P\{f(x, \delta) > 0\}$, and hence solve the program exactly). However, in practice obtaining these samples is often an expensive process. Luckily the theory of scenario optimisation provides robust bounds on the robustness of the obtained solution. The bounds generally take the following form:

$$P^N(V(\hat{x}_N) > \epsilon) < \beta. \tag{7}$$

This equation states that the probability of observing a bad set of data (i.e. a bad set of constraints) in future, such that our solution violates a proportion greater than ϵ of the constraints ($V(\hat{x}_N) > \epsilon$ where $V(\hat{x}_N) = \frac{1}{N}\sum_i^N V_i$ and $V_i = 1$ only if $f(x, \delta^i) > 0$), is no greater than β. The scenario approach gives a simple analytic form for the connection between ϵ and β in the case that the optimisation program is convex:

$$\beta = \frac{1}{\epsilon}\frac{n}{N+1}, \tag{8}$$

where N is the number of constraint samples in the training data set used to solve the scenario program, and n is the dimensionality of the design variable, x. Other tighter bounds exist in the literature [17]. For a fixed N and n we obtain a plot as shown in Fig. 1. The plot demonstrates that by decreasing ϵ slightly, $1-\beta$ can be made to be insignificantly small. Crucially our assessment of the solution is a-priori, although other techniques exist [18]. In the convex case the a-priori assessment is made possible by the fact that the number of support constraints (the number of constraints which if removed result in a more optimal solution) for a convex program is always less than the dimensionality of the design variable. For a non-convex program this is not necessarily the case, and therefore a new approach is required.

3.2 Non-convex case

Fortunately, another approach is possible. In [13] the following bound is given for the non-convex case:

$$P^N(V(\hat{x}_N) > \epsilon(s)) < \beta, \tag{9}$$

where

$$\epsilon(s) = \begin{cases} 1, & \text{for } s = N, \\ 1 - \sqrt[N-s]{\frac{\beta}{N\binom{N}{s}}}, & \text{otherwise,} \end{cases} \tag{10}$$

and s is the cardinality of the support set (in other words, the number of support constraints). The behaviour of this bound is similar to the convex case since in general increasing n should increase the size of the support set.

3.3 Finding The Cardinality of the Support Set

So far we have shown how we can efficiently solve the scenario program which is required to train a Neural Network with interval predictions, provided that we know the number of support constraints. The only practical challenge which remains is to calculate the cardinality of the support set. This is in general a computationally expensive task since the scenario program must be solved N times. In [13] an efficient algorithm is presented which only requires that the scenario problem is solved s times. However in this section we propose a more efficient algorithm which can be derived from our approach.

If the maximum error is plotted with respect to iterations (see example in Fig. 2) then we will be able to spot when the Neural Network training process has converged by noticing a cyclic behaviour in the plot. This signifies that our optimiser is orbiting around a point in the design space. A simple modification to our algorithm allows us to record which points in the training data set contribute to this orbit, as they are the points with the maximum error at each step. If a point which is not a member of this set is removed then the orbit will not change. Therefore we propose that the cardinality of the support set is bounded by the cardinality of the set of points which contribute to the converged orbit.

When using minibatches the set of points which contribute to the orbit will be an over estimate due to statistical fluctuations. Therefore it will usually be necessary to run a number of refining iterations where the batch gradient is computed by finding the maximum error of the whole support set. This allows us to avoid the error resulting from using minibatches whilst avoiding a computational cost of order $\mathcal{O}(NN_{iter})$. The refining iterations also allow us to avoid unsatisfied constraints in the scenario program (as discussed in Section 2.2), which may occur due to choosing a minibatch size which is not sufficiently large.

3.4 Incertitude in Training Data

We have proposed an algorithm for use with crisp training data. However one of the main advantages of the Interval Predictor Model frame work is that training data with incertitude (i.e. interval training data or fuzzy data) fits coherently into the paradigm [9]. An example of incertitude in training data is the adversarial attack model given in [10]. In fact, the proposed attack model places each training data point in an uncertain hyper-sphere (ℓ_2 ball) rather than the intervals with which many uncertainty practitioners are accustomed to. However, both cases are convex sets and therefore the conceptual challenge of accommodating this training data is similar. Since our model is more complex than that proposed in [9] the computations required to accommodate interval data are also more complex.

For the case of interval imprecision in the output variables (i.e. we observe pairs x_i and $[\underline{y}_i, \bar{y}_i]$) we can simply modify Eqn. 4 as follows:

$$\arg\min_{W,h} [h : \max\left(|\bar{y}_i - \hat{y}(x_i)|, |\underline{y}_i - \hat{y}(x_i)|\right) < h \forall i], \tag{11}$$

which can be written in simplified form if the width of interval $[\underline{y}_i, \bar{y}_i]$ is constant for all data points.

For interval incertitude in the input training data the situation is more complex, and since the sum of squares approach used in [9] is not directly applicable our algorithm will be more costly. If we observe pairs $[\underline{x}_i, \bar{x}_i]$ and $[\underline{y}_i, \bar{y}_i]$ then we must solve

$$\arg\min_{W,h} [h : \max_{x \in [\underline{x}_i, \bar{x}_i]} \left(|\bar{y}_i - \hat{y}(x)|, |\underline{y}_i - \hat{y}(x)|\right) < h \forall i], \tag{12}$$

where the nested optimisation in the constraints is clearly to blame for the inefficiency of the algorithm. One approach to solving this problem would be to attempt to brute force the nested optimisation (i.e. discretise along the upper 'edge' of the incertitude box), however if the incertitude is large or the dimensionality of the training data is high then this becomes impractical. Another possibility is assuming the prediction of the Neural Network is approximately linear locally and using the gradient of the Neural Network with respect to the inputs (which is known analytically) to find an approximate solution to the nested optimisation problem. This is similar to the approaches proposed in [19] and [20], where the gradient is used to search within a set close to the original training data for points which maximise the loss function of the Neural Network. The crucial difference is that in our formulation we need only search on the surface of the set, since we aim to enclose the whole set in the Interval Neural Network.

Since neither of these methods is completely satisfactory we leave the problem of interval uncertainty (ℓ_∞ ball, ℓ_2 ball and other cases) in the input training data as an unsolved problem.

4 Numerical Experiments

In order to illustrate the developed techniques we will demonstrate the Interval Neural Network on a modified version of a simple problem from [13]. We will train a Neural Network with 1 layer containing 10 neurons on 1250 samples from the following test function:

$$y = 0.3 * (15 * u * \exp(-3 * u) + w) \tag{13}$$

where w is a normal distributed random variable with zero mean and standard deviation $\sigma = 0.025$. The input variable u will be uniformly distributed between 0 and 1, since input data can be transformed into this range trivially. For clarity the algorithm used is described in Algorithm 3.

Algorithm 3 Maximum error backpropagation method as used in example

Input: Training data pairs (x_i,y_i for $i = 1, ..., N$)
  Randomly initialise weight tensor and h.
  for $i = 1, .., 5000$ **do**
    Generate set, B, of M random numbers, sampled without replacement between 1 and N
    Set $k = \arg\max_{j \in B} |y_j - \hat{y}(x_j)|$
    Use gradient of loss function to update W and h ($W_i \leftarrow W_i + \eta \frac{\partial (y_k - \hat{y}(x_k))^2}{\partial W}$)
    if $i > 4000$ **then**
      $S_{i-4000} = k$
    end if
  end for
  Set $B = S$
  for $i = 1, .., 100$ **do**
    Set $k = \arg\max_{j \in B} |y_j - \hat{y}(x_j)|$
    Use gradient of loss function to update W and h ($W_i \leftarrow W_i + \eta \frac{\partial (y_k - \hat{y}(x_k))^2}{\partial W}$)
    if $i > 4000$ **then**
      $R_i = k$
    end if
  end for
  Set $B = S$
  Set $h = \arg\max_{j \in B} |y_j - \hat{y}(x_j)|$
  Use Eqn. 10 to calculate a bound on the violation probability of the identified Neural Network, with $\beta = 10^{-6}$ and s equal to the cardinality of R.
Output: Weight tensor and h

The trained Neural Network is shown in Fig. 3. An annotated plot of the convergence (i.e. the maximum error at each step) is shown in Fig. 2. A minibatch size of 200 was used, and hence the support set (measured from assumed convergence after 4000 iterations) was an overestimate ($s = 181$). Running 100 refinement iterations allowed the size of the support set to be reduced to only 9 points.

Using Eqn. 10 we calculate

$$P^N(V(\hat{x}_N) > 0.057) < 10^{-6} \tag{14}$$

for the trained Interval Neural Network.

5 Conclusions

We have shown how to create Neural Networks which quantify their uncertainty with interval predictions. Our approach converges reliably and allows some of the data to be ignored to reduce the width of the prediction interval. Our approach is not restricted to a specific architecture and makes use of many modern techniques in deep learning.

Our approach is limited by the fact that this interval is a constant width. In a future paper we will extend the approach to demonstrate how our work can be extended to a non-constant interval prediction width.

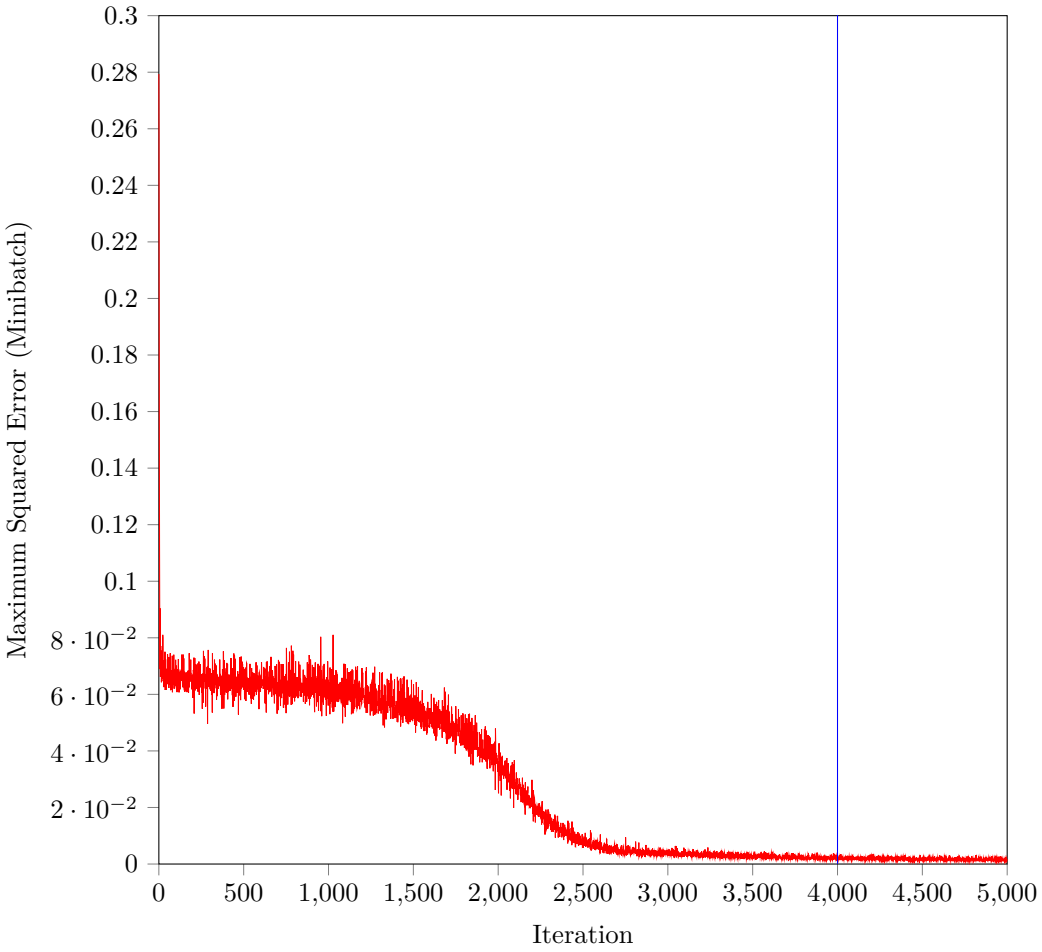
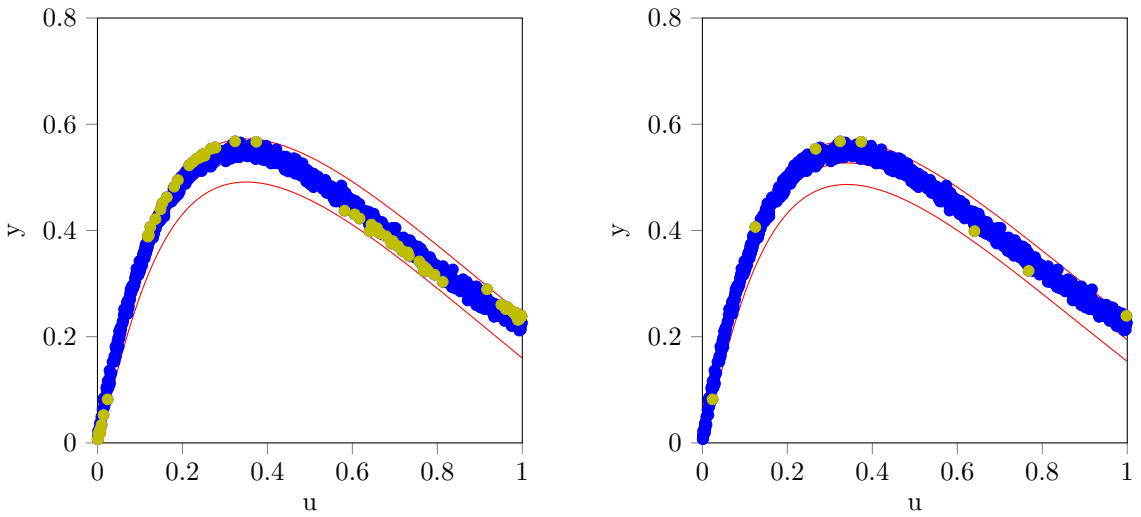


Figure 2: Plot of convergence of the Neural Network. The support set was determined by recording which training data points contribute to the change in weights after 4000 iterations (shown as a blue line).



(a) Upper bound to support set, before refinement.

(b) Upper bound to support set, after refinement.

Figure 3: Plot of trained Interval Neural Network. The support set is shown in yellow.

References

[1] S. Tolo, T. V. Santhosh, G. Vinod, U. Oparaji, and E. Patelli, in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (2017).

[2] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1 (Springer series in statistics New York, 2001).

[3] R. M. Neal, *Bayesian learning for neural networks*, vol. 118 (Springer Science & Business Media, 2012).

[4] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, Probabilistic programming in python using pymc3, *PeerJ Computer Science* **2**, e55 (2016).

[5] U. Oparaji, R.-J. Sheu, M. Bankhead, J. Austin, and E. Patelli, Robust artificial neural network for reliability and sensitivity analysis of complex non-linear systems, *Neural Networks* (2017).

[6] M. C. Campi, G. Calafiore, and S. Garatti, Interval predictor models: Identification and reliability, *Automatica* **45**, 382 (2009).

[7] G. Calafiore and M. C. Campi, Uncertain convex programs: randomized solutions and confidence levels, *Mathematical Programming* **102**, 25 (2005).

[8] E. Patelli, M. Broggi, S. Tolo, and J. Sadeghi, in *Proceedings of the 2nd ECCOMAS thematic conference on uncertainty quantification in computational sciences and engineering, UNCECOMP* (2017).

[9] M. J. Lacerda and L. G. Crespo, in *American Control Conference (ACC), 2017* (IEEE, 2017), 1487–1492.

[10] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, Towards deep learning models resistant to adversarial attacks, *arXiv preprint arXiv:1706.06083* (2017).

[11] H. Ishibuchi, H. Tanaka, and H. Okada, An architecture of neural networks with interval weights and its application to fuzzy regression analysis, *Fuzzy Sets and Systems* **57**, 27 (1993).

[12] L. Huang, B.-L. Zhang, and Q. Huang, Robust interval regression analysis using neural networks, *Fuzzy sets and systems* **97**, 337 (1998).

[13] M. C. Campi, S. Garatti, and F. A. Ramponi, in *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on* (IEEE, 2015), 4023–4028.

[14] S. Grammatico, X. Zhang, K. Margellos, P. Goulart, and J. Lygeros, A scenario approach for non-convex control design, *IEEE Transactions on Automatic Control* **61**, 334 (2016).

[15] M. C. Campi and S. Garatti, A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality, *Journal of Optimization Theory and Applications* **148**, 257 (2011).

[16] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (2011), 713–720.

[17] G. C. Calafiore, Random convex programs, *SIAM Journal on Optimization* **20**, 3427 (2010).

[18] J. Barrera, T. Homem-de Mello, E. Moreno, B. K. Pagnoncelli, and G. Canessa, Chance-constrained problems and rare events: an importance sampling approach, *Mathematical Programming* **157**, 153 (2016).

[19] A. Kurakin, I. Goodfellow, and S. Bengio, Adversarial machine learning at scale, *arXiv preprint arXiv:1611.01236* (2016).

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples, *arXiv preprint arXiv:1412.6572* (2014).

A Proof of statements in Section 2.2

We will solve the Optimisation Problem in Eqn. 4 approximately using Algorithm 1. Explicitly we wish to minimise the loss function

$$J = \max_{j\in[1,\dots,N]} (y_j - \hat{y}(x_j))^2. \tag{15}$$

Consider that the probability of selecting the true maximum of the squared error in a minibatch by random sampling without replacement is $\frac{M}{N}$. The probability that the maximum point selected in the minibatch is the i-th largest in the training set is

$$P(i) = \frac{\binom{N-i}{M-1}}{\binom{N}{M}}. \tag{16}$$

Then to find the expectation of i we calculate

$$\mathbb{E}(i) = \sum_{i=1}^{i=N-M+1} i \frac{\binom{N-i}{M-1}}{\binom{N}{M}} = \frac{N+1}{M+1}. \tag{17}$$

In the case that $\frac{N}{M} >> 1$ we find that the expression for the expected percentile reduces to

$$\frac{\mathbb{E}(i)}{N} \approx \frac{1}{M} \tag{18}$$

as promised. The variance of the percentile is

$$\mathrm{Var}\,(\frac{i}{N}) = \frac{M(N-M)(1+N)}{N^2(1+M)^2(2+M)}, \tag{19}$$

which becomes

$$\mathrm{Var}\,(\frac{i}{N}) \approx \frac{1}{M^2} \tag{20}$$

in the large N limit. Therefore we see that the minibatch technique performs best when the size of the training set is large, but it is also necessary to increase the minibatch size to avoid the gradient having a large variance.