Efficient Training of Interval Neural Networks for Imprecise Training Data

Jonathan Sadeghi[1], Marco De Angelis[1], and Edoardo Patelli[1]

[1]edoardo.patelli@liverpool.ac.uk, Institute for Risk and Uncertainty, Chadwick Building, University of Liverpool, Peach Street, Liverpool L69 7ZF, United Kingdom

July 8, 2019

Abstract

This paper describes a robust and computationally feasible method to train and quantify the uncertainty of Neural Networks. Specifically, we propose a back propagation algorithm for Neural Networks with interval 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 results on a test function example, and a more realistic engineering test case. The reliability of the predictions of these networks are guaranteed by the non-convex Scenario approach to chance constrained optimisation, which takes place following training, and is hence robust to the performance of the optimiser. A key result is that, by using minibatches of size M, the complexity of the proposed approach scales as $\mathcal{O}(M \cdot N_{iter})$, and does not depend upon the number of training data points as with other Interval Predictor Model methods. In addition, troublesome penalty function methods are avoided. To the authors' knowledge this contribution presents the first computationally feasible approach for dealing with convex set based epistemic uncertainty in huge data sets.

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 [43]. In all fields, but particularly in safety critical engineering applications, it is essential to quantify the uncertainty of the Neural Network. The simplest 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, these approaches do not robustly bound the predictions of the Neural Network. In general, Uncertainty Quantification can be achieved by probabilistic or non-probabilistic methods, although some hybrid techniques exist. Probabilistic models provide a richer model of uncertainty, however interval or hybrid techniques usually require fewer assumptions and provide theoretical guarantees on the reliability and robustness of the results. Bayesian Neural Networks (assisted by variational inference), where the weights are modelled probabilistically 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 [35]. However, many assumptions are necessary in order to apply this approach. For example, the weights are commonly assumed to have a Gaussian prior (or mixture of Gaussians) and the likelihood function is also assumed to be Gaussian. In addition, Variational Inference is more complex to implement than the maximum likelihood loss functions which are used by most machine learning engineers, and it is not explained in the literature how robust guarantees on the reliability of the results can be obtained [5]. In other words, if the optimiser performs poorly the predicted probability density may not accurately represent the training data .

In this paper we propose a back propagation algorithm for Neural Networks with interval predictions, and show how this can be efficiently used to create Interval Neural Networks. The proposed constant width interval predictions can be seen as a robust homoscedastic bound on the uncertainty of the trained network, though we also propose a method for obtaining non-constant width predictions. In order to maintain numerical stability we avoid using the penalty method proposed by Ishibuchi et al. [28], and instead propose minimising the maximum absolute error of the whole batch of gradients at each step. This is advantageous as the number of difficult-to-tune hyper-parameters is reduced. We present results for a test function example, and an engineering problem. 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. We explain how convex set training data can be accommodated in this paradigm, in the spirit of Lacerda and Crespo [32].

2 Related Work

Osband [37] and Kendall and Gal [29] discuss epistemic and aleatory uncertainty in Deep Neural Networks from a Bayesian perspective. Aleatory or irreducible uncertainty describes the natural stochastic behaviour contained within a model. Epistemic uncertainty or incertitude is uncertainty relating to a lack of knowledge, which can in principle be reduced by collecting more data. Interestingly, Hsu et al. [26] suggests that the brain processes incertitude in a different way to variability (i.e. aleatory uncertainty). Bayesian Neural Networks are becoming widely used due to popular implementations in PyMC3 and Edward, amongst others, which have made this approach viable in an industrial context [41].

However, several other probabilistic approaches to quantifying the uncertainty in Neural Networks exist. Subsampling techniques like bootstrapping can be applied to the training data to create multiple Neural Networks which improve the uncertainty quantification compared to mean squared error approaches, but the results are in general still unsatisfactory [22]. Such approaches require ensembles of Neural Networks to be trained, which is inefficient in space and time. Similarly, Robust Neural Networks apply Bayesian model selection and model averaging to sets of Neural Networks (without subsampling the training data), and whilst a prior is not required explicitly for the weights, other assumptions are made regarding the probabilistic distribution of the output of the model [36].

Probabilistic techniques are not the only method of modelling epistemic uncertainty [20]. In fact Balch et al. [3] claim that modelling epistemic uncertainty with probability distributions may cause risk to be underestimated, and lead to bizarre conclusions regarding when engineers try to reduce epistemic uncertainty in measurements. Interval Predictor Models are a recently developed machine learning technique for supervised learning which make interval predictions with guaranteed accuracy [12]. In other words, for every input example, $x^{(i)}$ an Interval Predictor Model would predict bounds on the output, $\bar{y}(x^{(i)})$ and $\underline{y}(x^{(i)})$, instead of just $y(x^{(i)})$. The first published software implementation of Interval Predictor Models was made available in the open source OpenCossan software [38]. The technique relies upon the solution of chance constrained convex optimisation programs by the scenario technique [6, 40].

The scenario technique is a method of approximately solving optimisation problems with probabilistic constraints by considering a random sample of the constraints. Scenario Optimisation is easier to use in practice than similar methods in statistical learning theory, since no knowledge of the Vapnik-Chervonenkis dimension (a measure of the capacity of the model, which is difficult to determine exactly) is required. Carè et al. [15] apply the scenario technique to supervised classification machine learning problems. A key advantage over other machine learning techniques is that interval training data (i.e. where the training data inputs are given in the form $x^{(i)} \in [\underline{x^{(i)}}, \bar{x}^{(i)}]$ due to epistemic uncertainty or some other reason) fits into the scenario optimisation framework coherently [32]. This can be seen as equivalent to defending against the attack model of adversarial examples considered by Madry et al. [33], where the network is trained to produce the same outputs for small perturbations of the input data. The framework also permits robustness against uncertainty in training outputs, i.e. $y^{(i)} \in [\underline{y^{(i)}}, \bar{y}^{(i)}]$, where $y^{(i)}$ is a single training example output.

Prior to the development of Scenario Optimisation, Neural Networks with interval outputs were proposed by Ishibuchi et al. [28], and further described by Huang et al. [27]. In these papers the learning takes place by identifying the weights W, which solve the following program:

$$\underset{\bar{W},\underline{W}}{\arg\min} \, [\mathbb{E}_x(\bar{y}(x) - \underline{y}(x)) : \bar{y}(x^{(i)}) > y^{(i)} > \underline{y}(x^{(i)}) \; \forall \; i], \tag{1}$$

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

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

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. This approach is somewhat similar to quantile regression [18]. In general penalty methods require careful choice of hyper-parameters to guarantee convergence. These Neural Networks act in a similar way to Interval Predictor Models, however the Interval Neural Networks do not include a robust assessment of the prediction accuracy from the training data set. Freitag et al. [21] define similar networks with fuzzy parameters to describe fuzzy data, which is closely related to interval data since operations in fuzzy arithmetic can be decomposed into repeated interval arithmetic operations. The fuzzy Neural Networks are trained by minimising a least square loss function (a set inclusion constraint is not used), which can also be applied to time series data sets.

Campi et al. [13] extended the Scenario Approach to non-convex optimisation programs, and hence applied the approach 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:

$$\operatorname*{arg\,min}_{W,h}\, [h : |y^{(i)} - \hat{y}(x^{(i)})| < h \ \forall\ i], \tag{3}$$

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. Grammatico et al. [25] extend the non-convex scenario approach to problems solved with the sampling and discarding technique where the most restrictive constraints are optimally removed [10].

All of the discussed techniques are summarised in Table 1. The aim of this paper is closest to the model proposed in Campi et al. [13]. However, in this paper we concentrate on efficient methods to solve Eqn. 3, and interpretations of the inferred uncertainty.

2.1 Comparison with other methods

Interval Neural Networks offer a principled framework for dealing with imprecision in training data. This paper describes a novel framework for training Neural Networks which output a specific type of convex set prediction: super-ellipsoids, which are mathematically parameterised as ellipsoids in a space equipped with an ℓ_p norm (this acts essentially as a transformation enabling continuous deformation between hyper-spheres and hyper-cubes). The hyper-ellipsoidal case, representing correlated uncertainty between outputs, and hyper-rectangular case, representing no correlation between outputs, are discussed in detail in Section 3.4. Bayesian techniques, and other probabilistic techniques

Uncertainty Model	Typical Application
Traditional Neural Network with Backpropagation	• Input and output: crisp data • Abundant data (ideally)
Traditional Interval Neural Network [28]	• Input: crisp Output: crisp, fuzzy or interval data (ℓ_∞ ball) • Abundant data (ideally)
Fuzzy Recurrent Neural Network [21]	• Input and output: fuzzy data, possibly time dependant • Abundant data (ideally)
Dropout as a Bayesian Approximation [29] with Adversarial Training [33]	• ℓ_2 ball input, precise output • Can quantify epistemic uncertainty, but in practice works best with huge data sets • Deterministic or stochastic 'true' model
Approach proposed in this paper	• Input and output: interval data, ℓ_2 ball, other convex sets. • Abundant data (ideally), but also possible to quantify confidence with small data sets.

Table 1: A summary of the models of uncertainty in Neural Networks described in Section 1 and their typical applications.

for Neural Networks are not capable of handing the set inclusion constraints required for the neural network output, since the output is a point value or probability distribution rather than a set. The Interval Neural Networks in this paper have several advantages over those proposed in previous literature. The main advantage is that the training algorithms allow more complex network architectures to be trained. Specifically, the gradient descent algorithm can be used and no penalty or barrier functions are required in the loss function. In addition, ℓ_2 ball training data is considered while other papers (e.g. [32]) only consider ℓ_∞ training data.

Like other set-based and interval uncertainty models, the networks do not indicate the relative likelihood of different outputs within the prescribed output interval, because an interval communicates less information than a fuzzy number or probability distribution; it indicates complete uncertainty within the defined range. Although the interval output could be seen as a disadvantage because it is less expressive than a probability distribution, it enables simple guarantees to be made on the performance of the network (e.g. Section 4.3). Furthermore, the loss functions proposed in this paper currently only apply to regression problems; no attempt has been made to generalise typical classification loss functions, e.g. the logistic loss function or cross entropy loss function.

3 Interval Neural Network Training

3.1 Overview

Firstly, note that one can solve Eqn. 3 by finding the Neural Network weights which minimise the loss:

$$\mathcal{L}_{\text{max-error}} = \max_i |y^{(i)} - \hat{y}(x^{(i)})|, \tag{4}$$

where h is the minimum value of the loss. It is trivial to show this is true, since the set inclusion constraint in Eqn. 3 requires that h is larger than the absolute error for each data point in the training set. Eqn. 4 will be referred to as the maximum absolute error loss (as opposed to the mean absolute error loss, which is most commonly used in machine learning). For the avoidance of doubt, Eqn. 4 takes the maximum over each point in the training data set, rather than each component of a multi-output Neural Network (though this is discussed in Section 3.4). In order to minimise the loss in Eqn. 4 stochastic gradient descent is used. To obtain an accurate estimate of h (the minimum value of the loss), the loss function

$$\mathcal{L}_{\text{actual}} = \mathcal{L}_{\text{max-error}} + (h - \mathcal{L}_{\text{max-error}})^2 \tag{5}$$

is minimised with respect to the weights and h, which is beneficial as the estimate for h is effectively averaged by the gradient descent algorithm and is hence more accurate than simply setting h to the value of the loss at any iteration in particular, whilst the minimum of the loss for W remains unchanged. This technique can be trivially applied for every subsequent loss function described in this paper, and is used in all of our numberical experiments. Our algorithm is described in further detail in Algorithm 1. Note that Algorithm 1 could also be initialized with the weights obtained by training the network with a mean squared error loss function, if these were already available.

Algorithm 1 is more costly than the standard back propagation method, since the proposed method costs $\mathcal{O}(N \cdot N_{iter})$, compared to a standard stochastic gradient descent cost of $\mathcal{O}(N_{iter})$. The algorithm is amenable to parallelisation, since at each step the N evaluations of the absolute error can be made simultaneously. However, the largest GPU architectures have several thousand cores, so for data sets with millions of data points Algorithm 1 would not be tractable. Note also that the N_{iter} required for convergence in both algorithms

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
    Set $k = \arg\max_{j \in [1,...,N]} |y^{(j)} - \hat{y}(x^{(j)})|$
    Use gradient of loss function to update $W$ and $h$ ($W \leftarrow W + \eta \nabla_W |y^{(k)} - \hat{y}(x^{(k)})| + (h - |y^{(k)} - \hat{y}(x^{(k)})|)^2$ $h \leftarrow h + \eta \frac{\partial}{\partial h}(h - |y^{(k)} - \hat{y}(x^{(k)})|)^2$)
  end for
Output: Weight tensor and $h$
```

is not necessarily the same, as this depends on the variance of the gradient at each step.

3.2 Scalability Improvement

We propose the use of minibatch stochastic gradient descent to reduce the computational cost of the algorithm [19], whereby a randomly selected subset of size M of the training data is selected at each step and used to evaluate the maximum absolute error loss, Eqn. 4. This procedure is described in further detail 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 $h$ ($W \leftarrow W + \eta \nabla_W |y^{(k)} - \hat{y}(x^{(k)})| + (h - |y^{(k)} - \hat{y}(x^{(k)})|)^2$ $h \leftarrow h + \eta \frac{\partial}{\partial h}(h - |y^{(k)} - \hat{y}(x^{(k)})|)^2$)
  end for
Output: Weight tensor and $h$
```

Using minibatches reduces the cost of the proposed algorithm to $\mathcal{O}(M \cdot N_{iter})$, which is a potentially vast improvement when $N >> M > 1$. However, we are now only minimising an *approximation* of Eqn. 4. Fortunately, in A we show that the true loss function can be approximated well for reasonably small minibatch sizes. For large N we find that using a minibatch of size M is equivalent to minimising the $\frac{1}{M}$-th percentile of the empirical cumulative distribution function of the error for the whole training data set. In addition, by minimising the k-th largest error in a minibatch of size M, one is actually minimising the $\frac{k}{M}$-th percentile of the empirical cumulative distribution function of the error for the whole training data set. If desired this can be checked after training by passing the entire dataset through the model once and checking the identified value of h against the data (this check will not be too costly if $N_{iter}M >> N$).

Minimising the empirical percentiles of the error on the training set, with the minibatch approximation of Eqn. 4, allows us to control the training of the Neural Network but does not by itself provide a statistical guarantee on performance on the test set. To statistically guarantee performance of the model it is therefore necessary to use the techniques in Section 4, and in particular Section 4.3.

3.3 Incertitude in Training Data

Algorithm 1 and Algorithm 2 are described for use with crisp training data. However one of the main advantages of the Interval Predictor Model framework is that training data with incertitude (i.e. interval training data or fuzzy data) fits coherently into the paradigm [32]. An example of incertitude in training data is a common defence against the adversarial attack model from Madry et al. [33]. The proposed attack model places each training data point in an uncertain hyper-sphere (ℓ_2 ball). Typically, in the context of uncertainty quantification, incertitude is characterised with intervals (ℓ_∞ ball). However, both cases are convex sets and therefore the conceptual challenge of accommodating this training data is similar. Since the neural network model is more complex than that proposed in Lacerda and Crespo [32] the computations required to accommodate interval data are also more complex.

For the case of interval imprecision in the output variables (i.e. pairs $x^{(i)}$ and $[\underline{y}^{(i)}, \bar{y}^{(i)}]$ are observed) Eqn. 3 can be modified as follows:

$$\arg\min_{W,h} [h : \max(|\bar{y}^{(i)} - \hat{y}(x^{(i)})|, |\underline{y}^{(i)} - \hat{y}(x^{(i)})|) < h \;\forall\; i], \tag{6}$$

which can be written in simplified form if the width of interval $[\underline{y}^{(i)}, \bar{y}^{(i)}]$ is constant for all data points. The optimisation program in Eqn. 6 can be solved by minimising the loss

$$\mathcal{L}_{\text{output incertitude}} = \max_i \max(|\bar{y}^{(i)} - \hat{y}(x^{(i)})|, |\underline{y}^{(i)} - \hat{y}(x^{(i)})|), \tag{7}$$

with respect to the weights, W, where h becomes the value of the loss at the minimum.

For interval incertitude in the input training data the situation is more complex, and since the sum of squares approach used in [32] is not directly applicable, and hence the algorithm with neural networks will be more costly. If the pairs $[\underline{x}^{(i)}, \bar{x}^{(i)}]$ and $[\underline{y}^{(i)}, \bar{y}^{(i)}]$ are observed then one must solve

$$\arg\min_{W,h} [h : \max_{x \in [\underline{x}^{(i)}, \bar{x}^{(i)}]} (|\bar{y}^{(i)} - \hat{y}(x)|, |\underline{y}^{(i)} - \hat{y}(x)|) < h \;\forall\; i], \tag{8}$$

where the nested optimisation in the constraints significantly increases the complexity 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 [31] and [24], 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 the formulation proposed in this paper only the surface of the set must be searched, since the aim is to enclose the whole set in the Interval Neural Network. Therefore we propose that Eqn. 8 is best solved by minimising

$$\mathcal{L}_{\text{input incertitude}} = \max_i \max\left(|\bar{y}^{(i)} - (\hat{y}(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2}) - \epsilon^{(i)})|, |\underline{y}^{(i)} - (\hat{y}(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2}) + \epsilon^{(i)})|\right), \tag{9}$$

with respect to the parameters W, where $\epsilon^{(i)} = |\frac{\bar{x}^{(i)} - \underline{x}^{(i)}}{2} \cdot (\text{sign}(\nabla_x \hat{y}(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2})) \circ \nabla_x \hat{y}(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2}))|$ ($\circ$ denotes component-wise multiplication of vectors), and h becomes the value of the loss at the minimum. This loss will provide an accurate solution to Eqn. 8 when the output of the neural network ($\hat{y}(x)$) is locally linear for a Taylor series expansion in the training data intervals, such that $|\hat{y}(x + \delta x) - (\hat{y}(x) + \delta x \cdot \nabla_x \hat{y}(x))| < \omega$, where ω is an arbitrarily small constant representing the accuracy of the solution and δx is a constant at the length scale of the interval width. Of course, higher order Taylor expansions can be used to construct more complex loss functions, or the assumption of monotonicity can be made ($\max_{x \in [\underline{x}^{(i)}, \bar{x}^{(i)}]} \hat{y}(x) = \hat{y}(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2} + \frac{\bar{x}^{(i)} - \underline{x}^{(i)}}{2} \circ \text{sign}(\nabla_x \hat{y}(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2}))))$.

This approach provides a computationally feasible approximate solution to many practical problems involving the ℓ_∞ ball and ℓ_2 ball uncertainty models. However, the authors believe there is certainly progress to be made in finding exact solutions to these problems, as well as generalisations to more complex uncertainty models, e.g. hyper ellipsoid models and super ellipsoid models.

3.4 Multi-output Neural Networks

A key advantage of Neural Networks over other machine learning techniques is the ease with which correlation between model outputs can be expressed. For example, this is applicable when the output layer is a one hot encoder for classification tasks, or an image for computer vision tasks. It is also of use for multi-task learning [2, 46]. So far Algorithm 1 and Algorithm 2 have been described in the context of supervised learning from data with only one output dimension. We generalise the algorithms in the previous sections to multi-output Neural Networks by predicting an ℓ_p ball, with radius h, around the output of the Neural Network in the output space. The ℓ_p norm is defined as $\|z\|_p =$

$(|z_1|^p + |z_2|^p + \cdots + |z_n|^p)^{1/p}$ (with z representing a vector with components z_1, z_2, etc.). An ℓ_p ball is simply a shape with ℓ_p norm equal to 1 on the surface. For readers unfamiliar with convex set models of uncertainty, it may be useful to note that $p = \infty$ corresponds to no correlation between outputs (intervals, or hypercubes), and $p \to 0$ corresponds to the case of completely correlated outputs. p becomes a hyper-parameter which can be optimised to express the dependence between outputs in the proposed model. A weighted norm can be used to form more complex shapes like super-ellipsoids, or hyper-rectangles (as opposed to hyper-cubes).

To train the Interval Neural Network the optimisation program

$$\arg\min_{W,h} [h : \left(\sum_j \Big| \frac{y_j^{(i)} - \hat{y}_j(x^{(i)})}{\hat{\sigma}_j} \Big|^p \right)^{\frac{1}{p}} < h \; \forall \; i] \tag{10}$$

should be solved, where the weights $\hat{\sigma}$ are normalised such that $\|\hat{\sigma}\|_2 = 1$, which is ensured by setting $\hat{\sigma}_i = \frac{\sigma_i}{\sqrt{\sum_j \sigma_j^2}}$, where σ_i are parameters to be found during training.

In practice, training takes place by replacing the absolute error in Algorithm 2 with the appropriate ℓ_p distance in the output space, i.e.

$$\mathcal{L}_{\text{multi-output}} = \max_i \left(\sum_j \Big| \frac{y_j^{(i)} - \hat{y}_j(x^{(i)})}{\hat{\sigma}_j} \Big|^p \right)^{\frac{1}{p}}, \tag{11}$$

which reduces to the case of a ℓ_p ball when $\sigma_i = 1 \; \forall \; i$.

For example, if the analyst believes there is no dependency between outputs they might minimise Eqn. 11 with $p = \infty$ for minibatches of training data. The network would then predict intervals (hyper-rectangles) with radius h. Explicitly the training loss for neural networks predicting hyper-rectangles is given by:

$$\mathcal{L}_{\text{hyper-rectangle}} = \max_i \max_j \Big| \frac{y_j^{(i)} - \hat{y}_j(x^{(i)})}{\hat{\sigma}_j} \Big|. \tag{12}$$

3.5 Heteroscedastic Interval Uncertainty

So far the Interval Neural Networks discussed (i.e. from solving Eqn. 3) have made predictions with constant interval width, or constant convex set width in the case of multi-output models. There may be some situations where a richer description of uncertainty is desired. Therefore in this section we describe how to generalise the results from the previous sections to the case of non-constant width interval prediction. Rather than solving the original Interval Neural Network optimisation program (Eqn. 1), we propose a modified model:

$$\arg\min_{W,h} [h : \frac{|y^{(i)} - \hat{y}(x^{(i)})|}{\hat{\sigma}(x^{(i)})} < h \; \forall \; i], \tag{13}$$

where the Neural Network provides $\hat{y}(x^{(i)})$ (the central line prediction of the interval), and $\hat{\sigma}(x^{(i)})$ (the interval half-width), and the other symbols have the same meanings as in Eqn. 3. In order for the optimisation program to yield a plausible Interval Neural Network it is required that $\hat{\sigma}(x^{(i)}) > 0$ and $\mathbb{E}_x(\hat{\sigma}(x)) = 1$. These constraints can be enforced by setting $\hat{\sigma}(x^{(i)}) = \frac{\sigma(x^{(i)})}{\mathbb{E}_x(\sigma(x))}$, where $\sigma(x^{(i)})$ is output from a neural network layer with positive only activation function (e.g. ReLU or Softplus, or in the case of a multi-output neural network, as in the previous section, Softmax). Then the Neural Network can be trained by minimising the loss given by

$$\mathcal{L}_{\text{heteroscedastic}} = \max_i \frac{|y^{(i)} - \hat{y}(x^{(i)})|\mathbb{E}_x(\sigma(x))}{\sigma(x^{(i)})}, \tag{14}$$

again h is obtained from the minimum value of the loss function. The loss is evaluated on minibatches, and therefore $\mathbb{E}_x(\sigma(x))$ is computed using the Monte Carlo estimator of the expectation on the minibatch. The trained network makes interval predictions with centre $\hat{y}$ and half-width $h\hat{\sigma}(x)$ (the normalising factor $\mathbb{E}_x(\sigma(x))$ should be precomputed and stored).

4 Interval Neural Network Reliability Assessment

4.1 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

$$\arg\min c^T z : P\{f(z, \delta) > 0\} \leq \epsilon, \tag{15}$$

where δ is a random variable, z 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 the scenario approach: the problem is solved for a random sample of the constraints, i.e. one solves

$$\arg\min c^T z : f(z, \delta^{(i)}) \leq 0, i = 1, ..., N, \tag{16}$$

where $\delta^{(i)}$ represents the i-th sampled value of the constraints [6].

Intuition indicates that the solution will be most accurate when the dimensionality of the design variable is low and one takes as many samples of the constraints as possible (in fact, an infinite number of sampled constraints would allow us to reliably estimate $P\{f(z, \delta) > 0\}$, 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:

$$P^N(V(\hat{z}_N) > \epsilon) < \beta. \tag{17}$$

This equation states that the probability of observing a bad set of data (i.e. a bad set of constraints) in future, such that the solution violates a proportion greater than ϵ of the constraints (i.e. $V(\hat{z}_N) > \epsilon$ where $V(\hat{z}_N) = \frac{1}{N}\sum_i^N V^{(i)}$ and $V^{(i)} = 1$ only if $f(z, \delta^i) > 0$), 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:

$$\beta = \frac{1}{\epsilon}\frac{n}{N+1}, \tag{18}$$

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, z. A plot is shown in Fig. 1 for a fixed N and n . The plot demonstrates that by decreasing ϵ slightly, $1 - \beta$ can be made to be insignificantly small. Other tighter bounds exist in the more recent Scenario Optimisation literature, e.g. [7, 9, 1], for example

$$\beta = \sum_{i=0}^{n-1} \binom{N}{i} \epsilon^i (1-\epsilon)^{N-i}. \tag{19}$$

Crucially the assessment of $V(\hat{z}_N)$ is possible a-priori, although other techniques exist [4]. Care et al. [16] analyse the reliability of solutions of the maximum error loss from Eqn. 4 in the scenario framework when $\hat{y}(x^{(i)})$ is convex in $x^{(i)}$ and the function weights.

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. Campi and Garatti [11] explore this connection for convex programs in further detail, by analysing the number of support constraints after a solution is obtained. This idea has a deep connection with the concept of regularisation in machine learning [8]. For a non-convex program, the number of support constraints is not necessarily less than the dimensionality of the design variable, and therefore a new approach is required, which we describe in the following section.

4.2 Non-convex case

Campi et al. [14] provides the following bound for the non-convex case:

$$P^N(V(\hat{z}_N) > \epsilon(s)) < \beta, \tag{20}$$

where

$$\epsilon(s) = \begin{cases} 1, & \text{for } s = N, \\ 1 - \sqrt[N-s]{\frac{\beta}{N\binom{N}{s}}}, & \text{otherwise,} \end{cases} \tag{21}$$

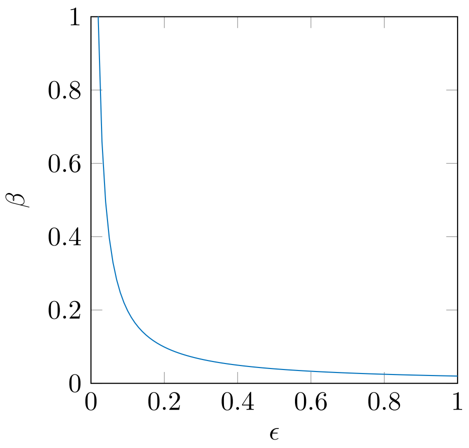

Figure 1: Plot of Eqn. 18 for $N = 100$ and $n = 2$.

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.

Finding the cardinality of the support set is in general a computationally expensive task, since the scenario program must be solved N times. Campi et al. [13] presents an efficient algorithm which only requires that the scenario problem is solved s times.

4.3 A Posteriori Frequentist Analysis

When data is available in abundance, as is typically the case in most machine learning tasks where a Neural Network is currently used, $V(\hat{z}_N)$ can be evaluated more easily by using a test set to collect samples from $V(\hat{z}_N)$. Estimating $V(\hat{z}_N)$ is similar to estimating a probability of failure in the well known reliability theory. Therefore one can construct a Monte Carlo estimator of $V(\hat{z}_N)$, or use more advanced techniques from reliability analysis if it is possible to interact with the data generating mechanism. For example if the number of test data points is large one can use the normal approximation Monte Carlo estimator of $V(\hat{z}_N)$ with $V(\hat{z}_N) \approx \frac{N_v}{N_t}$ and standard deviation $\sqrt{\frac{\frac{N_v}{N_t}(1-\frac{N_v}{N_t})}{N_t}}$, on a test set of size N_t, where N_v data points fall outside the interval bounds of the neural network.

A particularly robust method of estimating the probability of a binary outcome involves using the binomial confidence bounds. In this case specifically, one can bound $V(\hat{z}_N)$ with the desired confidence using the binomial confidence bounds:

$$\sum_{i=0}^{N_t-N_v} \binom{N_t}{i} (1-\underline{p})^i \underline{p}^{N_t-i} = \frac{\beta}{2} \tag{22}$$

and

$$\sum_{i=N_t-N_v}^{N_t} \binom{N_t}{i} (1-\bar{p})^i \bar{p}^{N_t-i} = \frac{\beta}{2}, \tag{23}$$

where $P(V(\hat{z}_N) < \bar{p} \cap V(\hat{z}_N) > \underline{p}) = \beta$. Estimating $V(\hat{z}_N)$ using a test set also offers the advantage that when the Neural Network is used for predictions on a different data set $V(\hat{z}_N)$ can be evaluated easily. If the value of $V(\hat{z}_N)$ obtained on the test set is higher than that on the training dataset then one can apply regularisation in order to implicitly reduce the size of the support set and increase $V(\hat{z}_N)$ on the test set (e.g. dropout regularisation, or ℓ_2 regularisation on the weights).

This methodology is ideal for models with a complex training scheme, where determining the support set would be prohibitively expensive. Note that the probabilistic assessment of the reliability of the model takes place separately from the training of the Neural Network, such that it is still robust, even if there is a problem with the Neural Network training. This is an important advantage over Variational Inference methods which are often used with Neural Networks.

5 Numerical Experiments

All experiments were timed on TensorFlow on a Google Colaboratory session equipped with an NVIDIA Tesla T4. In all experiments TensorFlow's ADAM optimiser was used with exponential gradient decay, i.e. learning rate = initial learning rate $*$ decay rate$^{\frac{\text{global step}}{\text{decay steps}}}$ [30].

5.1 Simple Numerical Example

5.1.1 Description

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

$$y = 0.3 * (15 * x * \exp(-3 * x) + w * x) \tag{24}$$

where w is a normal distributed random variable with zero mean and standard deviation $\sigma = 0.025$. The data is generated by sampling from the input variable x uniformly between 0 and 1. We perform the following experiments:

Experiment	1 Const. Width	2 Const. Width	3 Heteroscedastic	4 MSE
Minibatch Size, M	200	20	200	200
Initial Learning Rate	0.1	0.1	0.1	0.1
Learning Rate Decay Rate	0.96	0.96	0.96	0.96
Number of Training Epochs	6000	6000	6000	6000
Decay Steps	100	100	100	100

Table 2: Hyper-parameters used in the numerical experiments for the simple analytical function.

1. We train a constant width Neural Network using the loss from Eqn. 4, using a minibatch size of $M = 200$;
2. We repeat the previous experiment with a minibatch size of $M = 20$ to demonstrate the effect of using a smaller minibatch size;
3. We train a Neural Network with heteroscedastic uncertainty using the loss from Eqn. 14;
4. We train a Neural Network using the mean squared error (MSE) loss as a comparison.

For clarity, the algorithm used is described in Algorithm 2. The hyper-parameters used are shown in Table 2. Note that an epoch is defined as one pass of the whole dataset through the model, so training runs with smaller batch sizes require more iterations to complete the same number of training epochs, and hence will require more training time. These optimiser hyper-parameters were tuned manually by inspecting the loss curves, and the Minibatch size was chosen to be large enough to benefit from the properties discussed in A. The weights were initialised using the TensorFlow defaults (Glorot Uniform Initializer [23] for the kernel and Zeros for the bias), and h was initialised at zero.

5.1.2 Results

The training loss curves for Experiments 1, 2, 3 and 4 are shown in Figs. 2, 4, 6 and 8 respectively. The trained Neural Networks for Experiments 1, 2, 3 and 4 are shown in Figs. 3, 5, 7, and 9 respectively.

Using a train-test split ratio of 0.2, and the approach from Section 4.3 we calculate bounds on the violation probability, $\bar{v}$ and $\underline{v}$, from

$$P(V(\hat{z}_N) < \bar{v} \cap V(\hat{z}_N) > \underline{v}) = 10^{-3} \tag{25}$$

for each trained Interval Neural Network (with the test set size, $N_t = 250$). In this case the solution $\hat{z}_N$ is the obtained weights and model width of the Interval Neural Network. The results are summarised in Table 3, which also displays the

Experiment	1	2	3	4
	Const. Width	Const. Width	Heteroscedastic	MSE
Test Points, N_t	250	250	250	250
Bound Violating Test Points, N_v	1	5	2	N/A
$\bar{v}$	3.6×10^{-2}	6.4×10^{-2}	4.4×10^{-2}	N/A
$\underline{v}$	4.0×10^{-6}	3.0×10^{-3}	1.8×10^{-4}	N/A
Model Half-width, h	1.5×10^{-2}	1.0×10^{-2}	9.7×10^{-3}	N/A
Root Mean Squared Error	N/A	N/A	N/A	4.3×10^{-3}
Runtime (seconds)	66	525	75	68

Table 3: Results from the numerical experiments with the simple analytical function.

model half-width h for each trained network and the number of bound violating test points, N_v.

Note that as expected, the number of violating test points, N_v, and hence the bounds on the violation probability, $V(\hat{z}_N)$, are higher in Experiment 2 than Experiment 1, as the minibatch size, M, is lower. In addition we observe that as expected, the model half-width, h, is much lower in Experiment 3 than in Experiment 1. This indicates that the model in Experiment 1 is far too simple for the dataset, which we know to be true because in reality the training data contains heteroscedastic additive noise. For comparison we observe that the Neural Network trained with the mean squared error loss function (Experiment 4), has a root mean squared error on the test set of 4.3×10^{-3}, and a fitted model which is comparable to those in the Interval Model experiments (since if the strong assumption is made of a fitted Gaussian probability density then 99.7% data points would fall within 3 standard deviations of the mean).

5.2 Simple Numerical Example with Uncertain Training Data

5.2.1 Description

In order to demonstrate the developments in Section 3.3, we train an Interval Neural Network on a modified version of the previous example, where the training data consists of ℓ_∞ balls (intervals). The centre of the intervals $(x^{(i)}, y^{(i)}) = (\frac{\bar{x}^{(i)}+\underline{x}^{(i)}}{2}, \frac{\bar{y}^{(i)}+\underline{y}^{(i)}}{2})$ is generated by Eqn. 24. The incertitude in both the inputs and outputs will be given by the interval radius,

$$\frac{\bar{y}-\underline{y}}{2} = \frac{\bar{x}-\underline{x}}{2} = \frac{1}{160 * (|x-0.5|+0.1)}. \tag{26}$$

In order to allow for heteroscedasticity we train the network with the loss

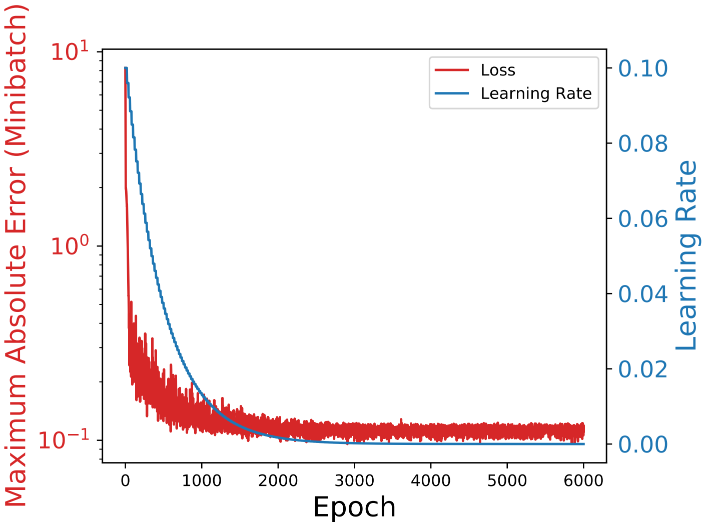

Figure 2: Plot of convergence of the Neural Network for Experiment 1.

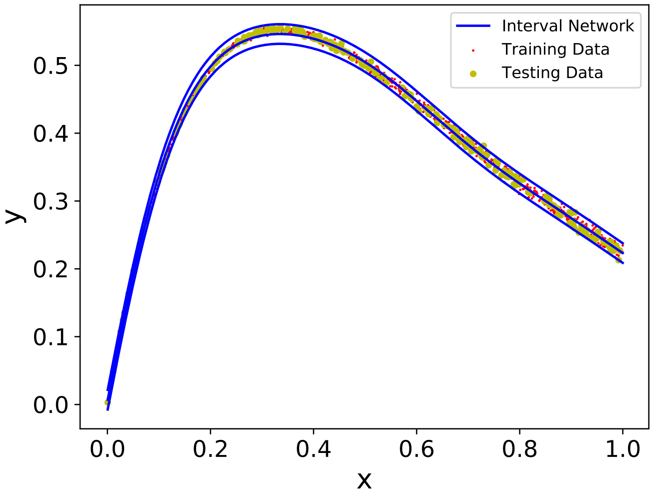

Figure 3: Plot of trained Interval Neural Network for Experiment 1. Training set shown in red, test set shown in yellow.

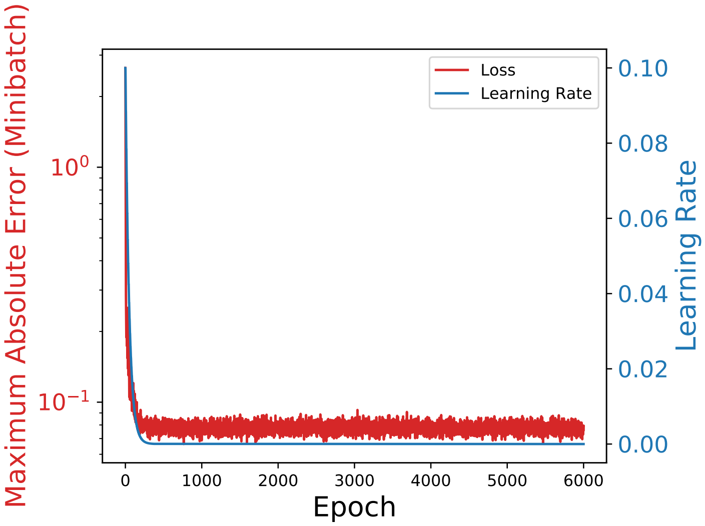

Figure 4: Plot of convergence of the Neural Network for Experiment 2.

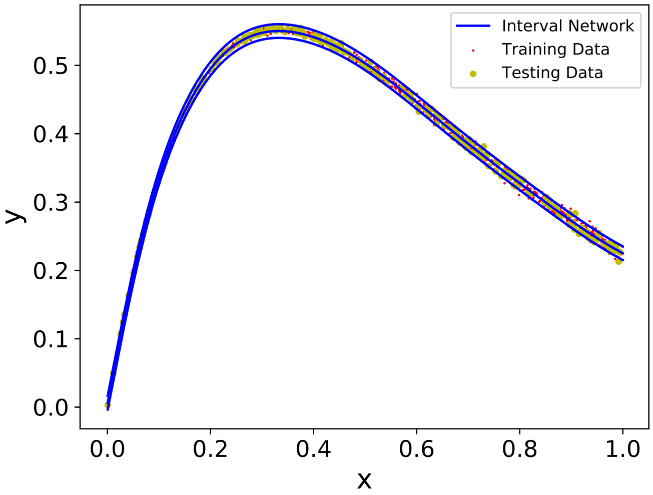

Figure 5: Plot of trained Interval Neural Network for Experiment 2. Training set shown in red, test set shown in yellow.

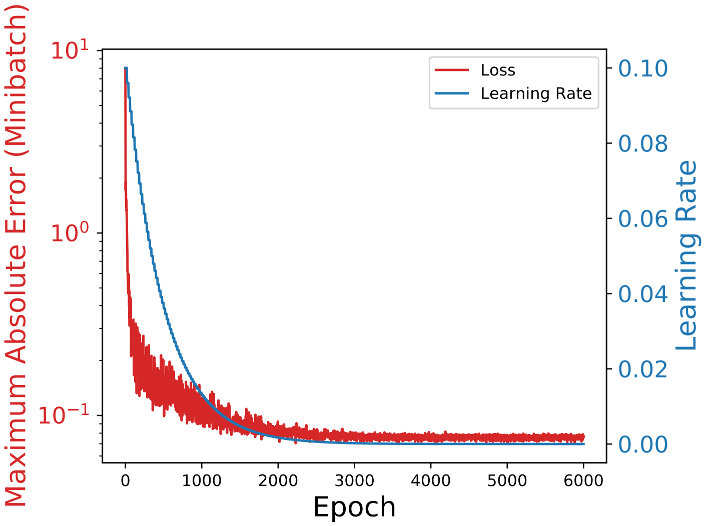

Figure 6: Plot of convergence of the Neural Network for Experiment 3.

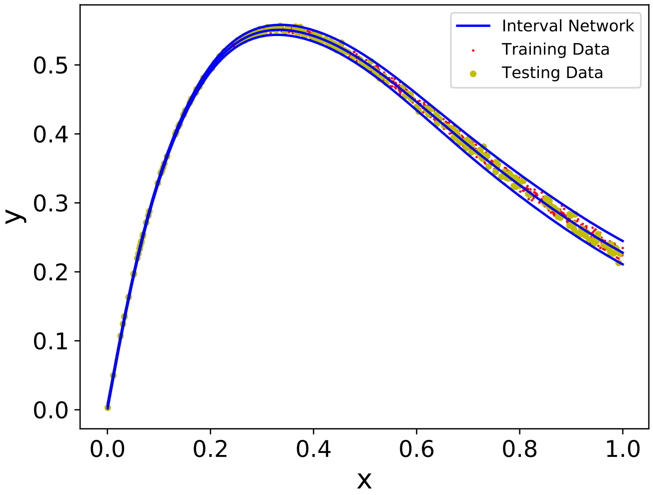

Figure 7: Plot of trained Interval Neural Network for Experiment 3. Training set shown in red, test set shown in yellow.

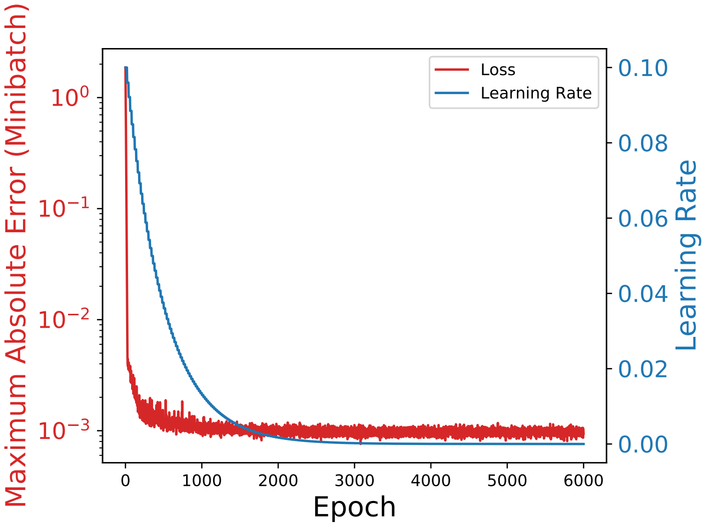

Figure 8: Plot of convergence of the Neural Network for Experiment 4 (Mean Squared Error Loss).

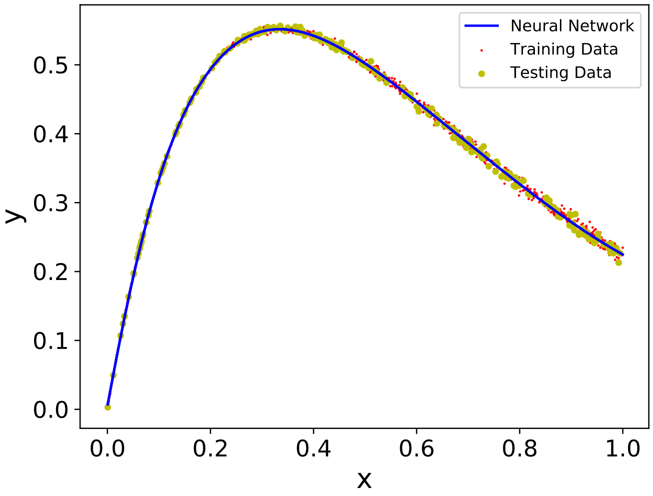

Figure 9: Plot of trained Interval Neural Network for Experiment 4 (Mean Squared Error Loss). Training set shown in red, test set shown in yellow.

Experiment	1	2
	Const. Width	MSE
Minibatch Size, M	200	200
Initial Learning Rate	0.1	0.1
Learning Rate Decay Rate	0.96	0.96
Number of Training Epochs	6000	6000
Decay Steps	100	100

Table 4: Hyper-parameters used in the numerical experiments with interval training data.

from Section 3.3 with the scaling in Section 3.5, i.e.

$$\mathcal{L}_{\text{experiment}} = \mathcal{L}_{\text{heteroscedastic}} + \frac{\bar{x}^{(i)} - \underline{x}^{(i)}}{2} \cdot |\nabla_x \mathcal{L}_{\text{heteroscedastic}}| \tag{27}$$

where σ and ϵ take the same meanings as in previous chapters, and the gradient of the loss, $|\nabla_x \mathcal{L}_{\text{heteroscedastic}}|$, is evaluated at the centre of the intervals $(\frac{\bar{x}^{(i)} + \underline{x}^{(i)}}{2})$. The same Neural Network architecture was used as in the previous example (10 neurons in hidden layer). In order to make a comparison, another network with two hidden layers with 10 and 20 neurons was used. This is summarised in Table 4.

In this case the Interval Network could not be compared with a traditional neural network, as a traditional neural network would not be able to represent the set inclusion constraint required to train with interval data.

5.2.2 Results

The training loss curves and trained single Interval Neural Network are shown in Fig. 10 and Fig. 11. The corresponding plots of the Neural Network with two hidden layers are shown in Fig. 12 and Fig. 13. Using a train-test split ratio of 0.2, and the approach from Section 4.3 we calculate bounds on $V(\hat{z}_N)$ with confidence 0.999. Although the single layer Interval Neural Network encloses the expected proportion of data based on the minibatch size, the interval appears overly large in places. This indicates that the chosen Neural Network may be too simple and hence the complexity (number of neurons) of the model could be increased, as in the neural network with two hidden layers. The results are summarised in Table 5.

5.3 Multi-output Test Function

5.3.1 Description

In order to test the multi-output loss function proposed in Section 3.4, we train an Interval Neural Network with the loss function Eqn. 12 on a test function

Experiment	1 Single Layer	2 Two Layers
Test Points, N_t	250	250
Bound Violating Test Points, N_v	1	2
$\bar{v}$	3.6×10^{-2}	4.4×10^{-2}
$\underline{v}$	4.0×10^{-6}	1.8×10^{-4}
Model Half-width, h	0.066	0.058
Runtime (seconds)	182	191

Table 5: Results from the numerical experiments with interval training data.

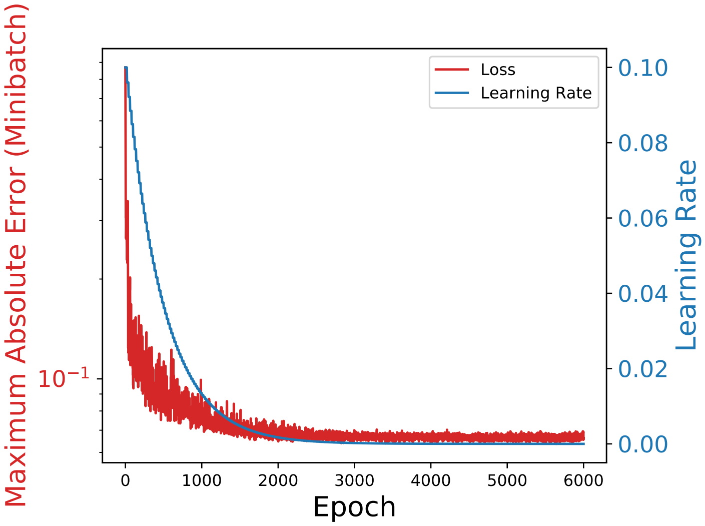

Figure 10: Plot of convergence of single hidden layer Interval Neural Network trained on uncertain data.

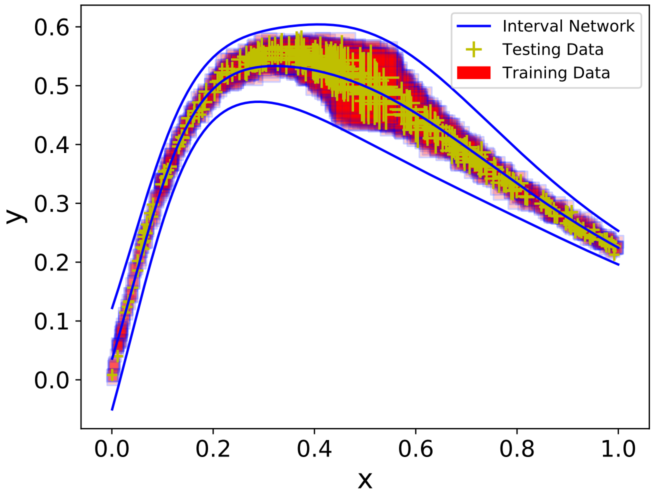

Figure 11: Plot of trained single hidden layer Interval Neural Network trained on uncertain data. Training set shown in as red squares, test set shown as yellow crosses.

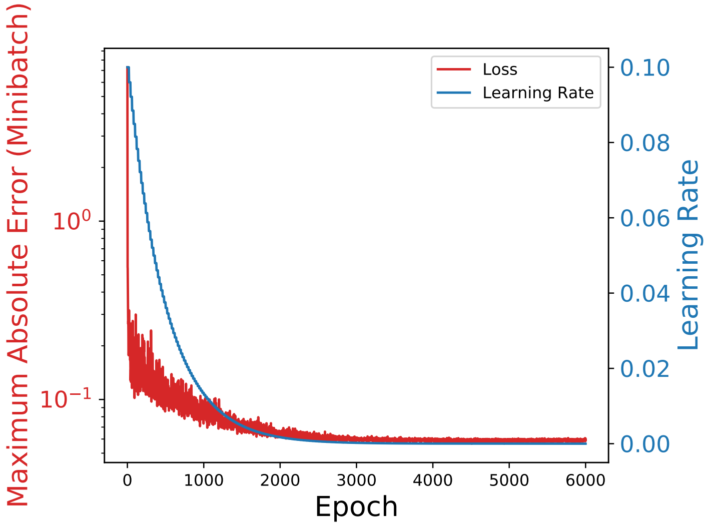

Figure 12: Plot of convergence of the Interval Neural Network with two hidden layers trained on uncertain data.

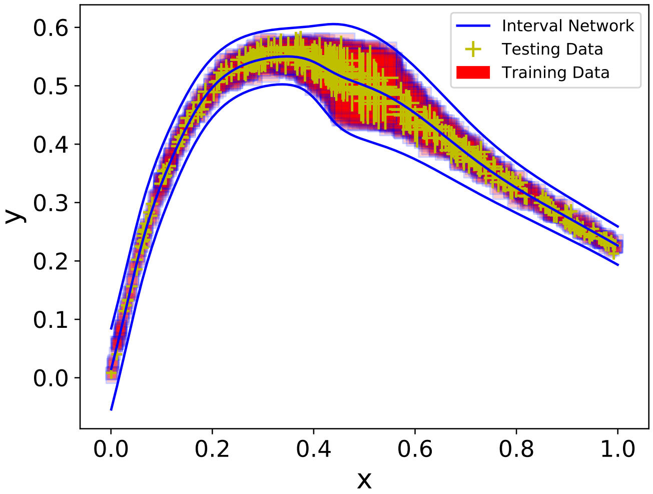

Figure 13: Plot of trained Interval Neural Network with two hidden layers trained on uncertain data. Training set shown in as red squares, test set shown as yellow crosses.

from [42], which was used to test multi-output emulators. The test function is given by

$$y_1 = 3x_1^3 + \exp(\cos(10x_2)\cos^2(5x_1)) \ + \exp(\sin(7.5x_3)) + w_1 \tag{28}$$

and

$$y_2 = 2x_1^2 + \exp(\cos(10x_1)\cos^2(5x_2)) \ + \exp(\sin(7.5x_3^2)) + 1.5w_2, \tag{29}$$

where w_1 and w_2 are uniformly distributed random numbers between 0 and 1. The model is trained on 1000 samples from the test function, made by sampling each component of x uniformly between 0 and 1, with a 0.2 train test split ratio. The Neural Network has 1 hidden layer with ReLU activation and 100 neurons. The hyper-parameters are summarised in Table 6. The TensorFlow default initialisers are used, except for σ which is initialised to ones.

5.3.2 Results

The plots of residuals for the network outputs are shown in Figs. 15 and 16. The training loss curve is shown in 14. Using a train-test split ratio of 0.2, and the approach from Section 4.3 we calculate bounds on $V(\hat{z}_N)$ with confidence 0.999 for the trained Interval Neural Network. The model half-widths were $h\hat{\sigma}_1 = 0.64$ and $h\hat{\sigma}_2 = 0.83$. Encouragingly, the model has identified a noise in each output similar to the true value from the test function. This is comparable with the

Experiment	1	2
	Const. Width	MSE
Minibatch Size, M	200	200
Initial Learning Rate	0.01	0.01
Learning Rate Decay Rate	0.99	0.99
Number of Training Epochs	10000	10000
Decay Steps	200	200
ℓ_2 regularisation scale	1.5×10^{-3}	1.5×10^{-3}

Table 6: Hyper-parameters used in the numerical experiments for the multi-output test function.

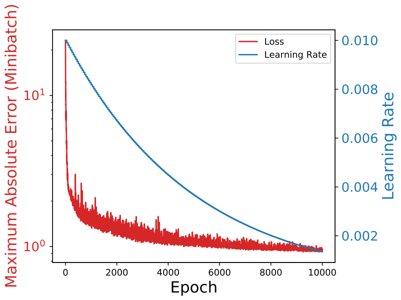

Figure 14: Plot of convergence of the multi-output Neural Network.

result obtained by training the same network with the MSE loss. The results are summarised in Table 7.

5.4 Realistic Engineering Test Case

5.4.1 Description

The compressive strength of concrete is a nonlinear function of age and ingredients. Yeh [45] provides a database with 1030 experimental measurements of the compressive strength of concrete as a function of age and composition in kg/m^3 (cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate)[1]. No information is provided about incertitude in the measurements, and therefore we are forced to process the data as it is given.

We wish to obtain robust bounds for the compressive strength of the concrete. This can be used for a worst case structural reliability analysis calculation. We replicate the architecture from Yeh [45] with our proposed algorithm and train a Neural Network with 1 hidden layer containing 8 neurons with hyperbolic tangent activation functions on the normalised data set (transformed to have mean zero and unit variance).

[1] Copyright Prof. I-Cheng Yeh

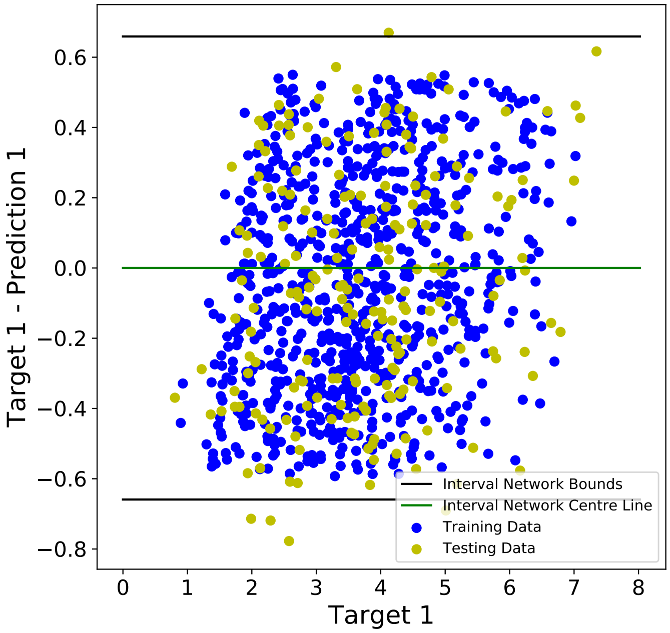

Figure 15: Plot of Residuals for output 1 of multi-output Interval Neural Network. Training set shown in as blue, test set shown in yellow.

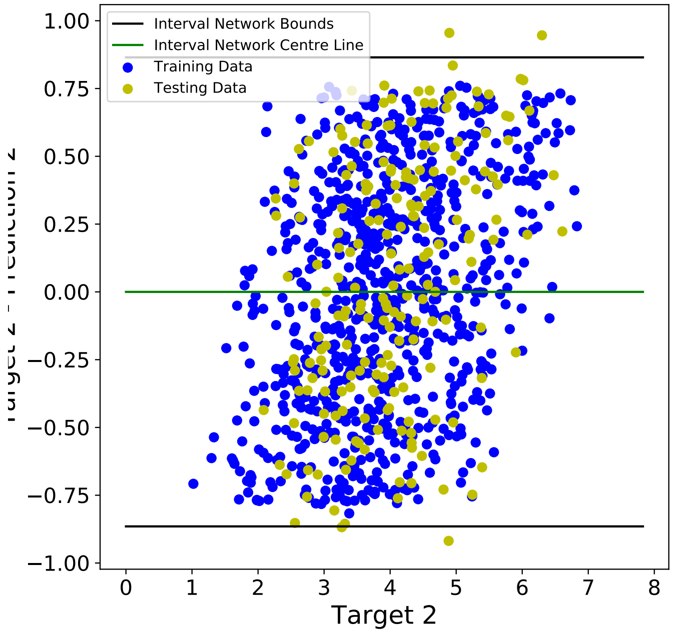

Figure 16: Plot of Residuals for output 2 of multi-output Interval Neural Network. Training set shown in as blue, test set shown in yellow.

Experiment	1 Const. Width	2 MSE
Test Points, N_t	200	200
Bound Violating Test Points, N_v	8	N/A
$\bar{v}$	1.0×10^{-1}	N/A
$\underline{v}$	1.0×10^{-2}	N/A
Model Half-width Output 1, $h\hat{\sigma}_1$	0.66	N/A
Model Half-width Output 1, $h\hat{\sigma}_2$	0.85	N/A
Root Mean Squared Error Output 1	N/A	0.34
Root Mean Squared Error Output 1	N/A	0.50
Runtime (seconds)	115	88

Table 7: Results from the numerical experiments with the multi-output test function.

We apply Algorithm 2 with the constant width loss from Eqn. 4 and $M = 200$. The weights are again initialised with the TensorFlow defaults. The hyper-parameters are summarised in Table 8

5.4.2 Results

Annotated plots of the convergence for the upper and lower bounds (i.e. the maximum error at each step) are shown in Fig. 17. The absolute error for the upper and lower bound (i.e. the 'residuals') is plotted in Fig. 18, and corresponds to an error width of $h = 14.6$ MPa, so the bounds had width 29.2 MPa. Using a train-test split ratio of 0.2, and the approach from Section 4.3

Experiment	1 Const. Width	2 MSE
Minibatch Size, M	200	200
Initial Learning Rate	0.01	0.01
Learning Rate Decay Rate	0.9	0.9
Number of Training Epochs	15000	15000
Decay Steps	1000	1000
ℓ_2 regularisation scale	N/A	N/A

Table 8: Hyper-parameters used in the numerical experiments for the concrete test dataset.

Experiment	1 Const. Width	2 MSE
Test Points, N_t	206	206
Bound Violating Test Points, N_v	8	N/A
$\bar{v}$	1.0×10^{-1}	N/A
$\underline{v}$	9.7×10^{-3}	N/A
Model Half-width, h /MPa	14.6	N/A
Root Mean Squared Error /MPa	N/A	5.85
Runtime (seconds)	145	146

Table 9: Results from the numerical experiments for the concrete test dataset.

we calculate bounds on $V(\hat{z}_N)$ with confidence 0.999 for the trained Interval Neural Network. The results compare favourably with other machine learning techniques [44]. The results are summarised in Table 9.

5.5 Outaouais Benchmark Dataset

5.5.1 Description

The Outaouais dataset was introduced in the Evaluating Predictive Uncertainty Challenge [39]. The dataset is for a regression problem with 37 features and 1 target variable. The dataset consists of 20000 training examples and 9000 test examples.

To predict the target a Heteroscedastic Interval Neural Network was trained with Eqn. 14. The network architecture had two hidden layers with 150 and 20 neurons with ReLU activation functions. The weights were initialised with the TensorFlow defaults.

This was compared to a heteroscedastic maximum likelihood perceptron network (heteroscedastic MLP) [17], trained with the same network architecture and hyper-parameters. The hyper-parameters for both models are summarised in Table 10

5.5.2 Results

Using the test data set with the approach from Section 4.3 we calculate bounds on $V(\hat{z}_N)$ with confidence 0.999 for the trained Interval Neural Network. The results are summarised in Table 11. The confidence bound on $V(\hat{z}_N)$ of the Heteroscedastic Interval Network is superior to that of the Heteroscedastic MLP when Chebyshev's inequality is used to produce a confidence bound from the Mean Squared Error of the MLP.

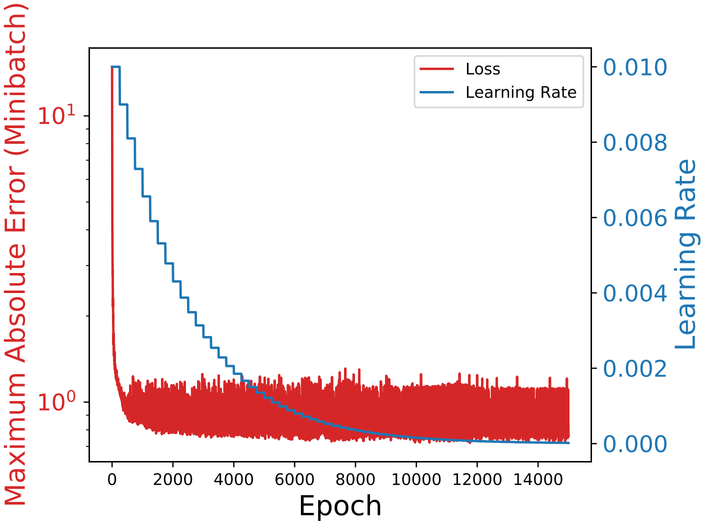

Figure 17: Plot of convergence of the Interval Neural Network to predict Concrete Compressive Strength.

Experiment	1 Heteroscedastic Interval Network	2 Heteroscedastic MLP
Minibatch Size, M	200	200
Initial Learning Rate	0.001	0.01
Learning Rate Decay Rate	0.995	0.995
Number of Training Epochs	2000	2000
Decay Steps	200	200
ℓ_2 regularisation scale	N/A	N/A
Dropout Rate	0.01	N/A

Table 10: Hyper-parameters used in the numerical experiments for the Outaouais dataset.

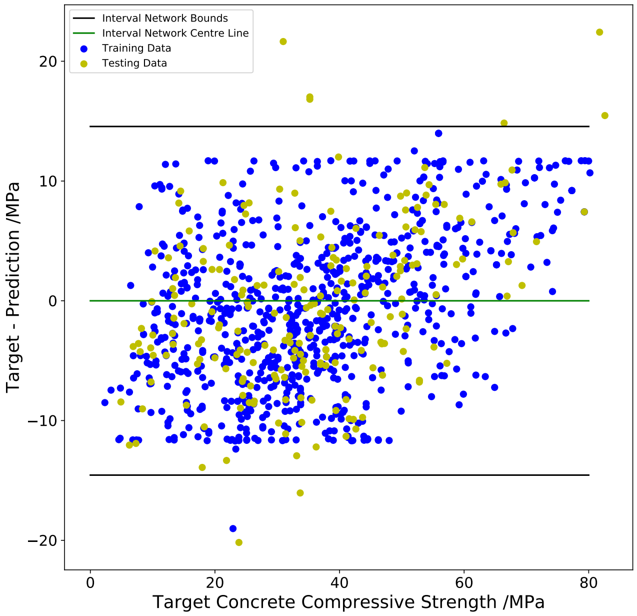

Figure 18: Plot of residuals (difference of predictions and targets) for Interval Neural Network to predict Concrete Compressive Strength. Model central line shown in green, and bounds shown in black. Training set shown in blue, test set shown in yellow.

Experiment	1 Heteroscedastic Interval Network	2 Heteroscedastic MLP
Test Points, N_t	9000	9000
Bound Violating Test Points, N_v	671	N/A
$\bar{v}$	6.6×10^{-2}	N/A
$\underline{v}$	8.3×10^{-2}	N/A
Model Half-width, h	0.29	N/A
Normalised Mean Squared Error, nMSE	N/A	0.038
Runtime (seconds)	691	705

Table 11: Results from the numerical experiments for the Outaouais dataset. The data variance used to compute the nMSE metric was 0.55.

6 Conclusions

In this paper, we have demonstrated how to create Neural Networks which quantify their uncertainty with interval predictions. In order to achieve scalability the proposed technique relies upon techniques developed for modern deep learning applications, such as minibatch gradient descent. The proposed approach converges reliably and is not restricted to a specific architecture. Crucially we avoid using explicit set inclusion relationships in the training process, which usually cause computational difficulties for practitioners of interval methods.

Since the model is not Bayesian it is unnecessary to specify prior distributions, or to use complex variational inference implementations. Instead the uncertainty is modelled using a interval which contains at least a specific proportion of the true output with near certainty.

The main contribution of this paper is to provide a computationally feasible alternative to Bayesian models of uncertainty in Neural Networks, by allowing the Neural Network to be trained from data specified by ℓ_2 or ℓ_∞ balls, which the network is forced to include in its prediction interval. The theoretical contributions of this paper could be applied to many convex models of uncertainty, and hence useful domain specific models could be derived from the work presented in this paper. In future, the work presented in this paper could be extended to consider more general uncertainty models, such as fuzzy neural networks or imprecise probabilistic network models.

Acknowledgements

This research has been generously supported by the *EPSRC Centre for Doctoral Training in Nuclear Fission - Next Generation Nuclear* (Grant reference: EP/L015390/1) which is gratefully acknowledged by the authors. This research is also partly funded by the UK Engineering & Physical Sciences Research Council with grant no. EP/R006768/1.

A Proof of statements in Section 3.2

We will solve the Optimisation Problem in Eqn. 3 approximately using Algorithm 1. Explicitly we wish to minimise the loss function

$$\mathcal{L} = \max_{j \in [1,\ldots,N]} |y^{(j)} - \hat{y}(x^{(j)})|. \tag{30}$$

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

$$P(i) = \frac{\binom{N-i}{M-1}}{\binom{N}{M}}. \tag{31}$$

Then to find the expectation of i we calculate

$$\mathbb{E}(i) = \sum_{i=1}^{i=N-M+1} i \frac{\binom{N-i}{M-1}}{\binom{N}{M}} = \frac{N+1}{M+1}. \tag{32}$$

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

$$\frac{\mathbb{E}(i)}{N} \approx \frac{1}{M} \tag{33}$$

as promised. The variance of the percentile is

$$\mathrm{Var}\,(\frac{i}{N}) = \frac{M(N-M)(1+N)}{N^2(1+M)^2(2+M)}, \tag{34}$$

which becomes

$$\mathrm{Var}\,(\frac{i}{N}) \approx \frac{1}{M^2} \tag{35}$$

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.

Now, let us consider the case when the $k-1$ data points in the minibatch with the largest error are ignored, i.e. we minimise the k-th largest error in the minibatch. The probability that the k-th largest error point selected in the minibatch is the i-th largest error in the training set is

$$P(i) = \frac{\binom{N-i}{M-k}\binom{i-1}{k-1}}{\binom{N}{M}}. \tag{36}$$

In [34] the order statistics are given for uniform distributions sampled without replacement. This allows us to find the expectation of i, which is given by

$$\mathbb{E}(i) = \frac{k(1+N)-1-M}{(1+M)}, \tag{37}$$

which reduces to Eqn. 32, when $k=1$ and $\frac{N}{M} >> 1$ as expected. The variance of i is

$$\mathrm{Var}\,(i) = \frac{k(M-k+1)(N+1)(N-M)}{(M+1)^2(M+2)}, \tag{38}$$

which reduces to Eqn. 35 when the appropriate limits are taken. This provides valuable insight - when $k > 1$ is minimised it is necessary to increase M slightly to maintain constant variance in the gradient.

All the above results assume the minibatch is constructed by sampling without replacement. If the minibatch is constructed by sampling with replacement then the order statistics for sampling with replacement should be used instead.

References

[1] T. Alamo, R. Tempo, A. Luque, and D. R. Ramirez. Randomized methods for design of uncertain systems: Sample complexity and sequential algorithms. *Automatica*, 52:160–172, 2015.

[2] A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[3] M. S. Balch, R. Martin, and S. Ferson. Satellite conjunction analysis and the false confidence theorem. *arXiv preprint arXiv:1706.08565*, 2017.

[4] 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(1):153–189, 2016.

[5] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[6] G. Calafiore and M. C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102(1):25–46, 2005.

[7] G. C. Calafiore. Random convex programs. *SIAM Journal on Optimization*, 20(6):3427–3464, 2010.

[8] M. C. Campi and A. Caré. Random convex programs with l_1-regularization: sparsity and generalization. *SIAM Journal on Control and Optimization*, 51(5):3532–3557, 2013.

[9] M. C. Campi and S. Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM Journal on Optimization*, 19(3):1211–1230, 2008.

[10] 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(2):257–280, 2011.

[11] M. C. Campi and S. Garatti. Wait-and-judge scenario optimization. *Mathematical Programming*, 167(1):155–189, 2018.

[12] M. C. Campi, G. Calafiore, and S. Garatti. Interval predictor models: Identification and reliability. *Automatica*, 45(2):382–392, 2009.

[13] M. C. Campi, S. Garatti, and F. A. Ramponi. Non-convex scenario optimization with application to system identification. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 4023–4028. IEEE, 2015.

[14] M. C. Campi, S. Garatti, and F. A. Ramponi. A general scenario theory for nonconvex optimization and decision making. *IEEE Transactions on Automatic Control*, 63(12):4067–4078, 2018.

[15] A. Carè, F. A. Ramponi, and M. C. Campi. A new classification algorithm with guaranteed sensitivity and specificity for medical applications. *IEEE Control Systems Letters*, 2(3):393–398, 2018.

[16] A. Care, S. Garatti, and M. C. Campi. Scenario min-max optimization and the risk of empirical costs. *SIAM Journal on Optimization*, 25(4): 2061–2080, 2015.

[17] G. C. Cawley, N. L. Talbot, and O. Chapelle. Estimating predictive variances with kernel ridge regression. In *Machine Learning Challenges Workshop*, pages 56–77. Springer, 2005.

[18] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2017.

[19] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 713–720, 2011.

[20] S. Ferson, C. A. Joslyn, J. C. Helton, W. L. Oberkampf, and K. Sentz. Summary from the epistemic uncertainty workshop: consensus amid diversity. *Reliability Engineering & System Safety*, 85(1-3):355–369, 2004.

[21] S. Freitag, W. Graf, and M. Kaliske. Recurrent neural networks for fuzzy data. *Integrated Computer-Aided Engineering*, 18(3):265–280, 2011.

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

[23] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

[24] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *ICLR 2015*, 2014.

[25] 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(2):334–345, 2016.

[26] M. Hsu, M. Bhatt, R. Adolphs, D. Tranel, and C. F. Camerer. Neural systems responding to degrees of uncertainty in human decision-making. *Science*, 310(5754):1680–1683, 2005.

[27] L. Huang, B.-L. Zhang, and Q. Huang. Robust interval regression analysis using neural networks. *Fuzzy sets and systems*, 97(3):337–347, 1998.

[28] 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(1):27–39, 1993.

[29] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *ICLR 2017*.

[32] M. J. Lacerda and L. G. Crespo. Interval predictor models for data with measurement uncertainty. In *American Control Conference (ACC), 2017*, pages 1487–1492. IEEE, 2017.

[33] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *ICML 2017 Workshop on Principled Approaches to Deep Learning*, 2017.

[34] H. Nagaraja. Order statistics from discrete distributions. *Statistics: a journal of theoretical and applied statistics*, 23(3):189–216, 1992.

[35] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[36] 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.

[37] I. Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *Proceedings of the NIPS* 2016 Workshop on Bayesian Deep Learning*, 2016.

[38] E. Patelli, M. Broggi, S. Tolo, and J. Sadeghi. Cossan software: A multidisciplinary and collaborative software for uncertainty quantification. In *Proceedings of the 2nd ECCOMAS thematic conference on uncertainty quantification in computational sciences and engineering, UNCECOMP*, 2017.

[39] J. Quinonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges Workshop*, pages 1–27. Springer, 2005.

[40] J. Sadeghi, M. de Angelis, and E. Patelli. Frequentist history matching with interval predictor models. *Applied Mathematical Modelling*, 61:29–48, 2018.

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

[42] samcoveney. samcoveney gp_emu_uqsa: First official release, Dec. 2016. URL https://doi.org/10.5281/zenodo.215521.

[43] S. Tolo, T. V. Santhosh, G. Vinod, U. Oparaji, and E. Patelli. Uncertainty quantification methods for neural networks pattern recognition. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017.

[44] S. Wenkel. Revisiting machine learning datasets - concrete compressive strength // simonwenkel.com, 2019. URL https://www.simonwenkel.com/2018/08/08/revisiting_ml_datasets_concrete_compressive_strength.html.

[45] I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.

[46] C. Zhang and Z. Zhang. Improving multiview face detection with multi-task deep convolutional neural networks. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1036–1041. IEEE, 2014.