How to Propagate Uncertainty via AI Algorithms

Olga Kosheleva\textsuperscript{1} and Vladik Kreinovich\textsuperscript{2}

Departments of\textsuperscript{1} Teacher Education and\textsuperscript{2} Computer Science
University of Texas at El Paso
El Paso, Texas 79968, USA, \{olgak, vladik\}@utep.edu

Abstract. Any data processing starts with measurement results. Measurement results are never absolutely accurate. Because of this measurement uncertainty, the results of processing measurement results are, in general, somewhat different from what we would have obtained if we knew the exact values of the measured quantities. To make a decision based on the result of data processing, we need to know how accurate is this result, i.e., we need to propagate the measurement uncertainty through the data processing algorithm. There are many techniques for uncertainty propagation. Usually, they involve applying the same data processing algorithm several times to appropriately modified data. As a result, the computation time for uncertainty propagation is several times larger than data processing itself. This is a very critical issue for data processing algorithms that take a lot of computational steps – such as modern deep learning-based AI techniques, for which a several-times increase in computation time is not feasible. At first glance, the situation may seem hopeless. Good news is that there is another problem with modern AI algorithms: usually, once they learn, their weights are frozen, and they stop learning – as a result, the quality of their answers decreases with time. This is good news because, as we show, solving the second problem – by allowing at least one learning step for each new use of the model – helps to also come up with an efficient uncertainty propagation algorithm.

Keywords: uncertainty propagation, machine learning, frozen weight, AI algorithms

1. Introduction

Need for data processing. In many application areas, we need to process data, i.e., we need to transform the available information $x_1, \ldots, x_n$ into an estimate $y$ for some quantity describing the current or the future state of the world, or describing an action or design that is recommended based on this information. In the following text, we will denote the data processing algorithm by

$$y = f(x_1, \ldots, x_n).$$

(1)

Data processing is what computers were invented for, data processing is what computers are mostly used now.

AI-based data processing has become ubiquitous. In the last decades, more and more data processing is done by AI-based algorithms, mostly by deep neural networks; see, e.g., (Goodfellow et al., 2016). To come up with these algorithms, we first train a multi-layer neural network on thousands and millions of example – and, as a result, come with the weights for which the neural
network best fits these examples. This training usually takes a lot of time. Once the training is done, the weights are fixed ("frozen"), and the neural network is ready to be used for data processing.

**Need for uncertainty propagation.** The data $x_1, \ldots, x_n$ that we process comes either directly from measurements, or from some previous processing of measurement results. Measurements are never absolutely accurate (see, e.g., (Rabinovich, 2005)): the result $\tilde{x}$ of measuring a quantity is, in general, somewhat different from the actual (unknown) value $x$ of this quantity. Because of this, the value $\tilde{y} = f(\tilde{x}_1, \ldots, \tilde{x}_n)$ that we get by processing measurement results is, in general, somewhat different from the value $y = f(x_1, \ldots, x_n)$ that we would get if we knew the exact values of the corresponding quantities.

To make an appropriate decision, it is important to know how accurate is our estimate. For example, if we estimate the amount of oil in an oilfield and our estimate is 150 million tons, there is a big difference between:

- a situation in which it is $150 \pm 50$ – so we should start exploiting this field, and

- a situation in which it is $150 \pm 200$, so that maybe there is no oil at all, and we should perform additional tests before investing serious money into this project.

**Current uncertainty propagation techniques are not always applicable for AI-based algorithms.** There are many techniques for uncertainty propagation. Usually, they involve applying the same data processing algorithm several times to appropriately modified data. As a result, the computation time for uncertainty propagation is several times larger than data processing itself.

This is a very critical issue for data processing algorithms that take a lot of computational steps – such as modern deep learning-based AI techniques. For these techniques, for which a several-times increase in computations time is not feasible.

**An additional problem related to AI-based data processing.** Uncertainty propagation is one of the problems of the modern AI-based data processing techniques, there are other important problems. One of them is related to the fact that the more data we use for training and the more up-to-date is this data, the better the training results. Once an algorithm has been trained, its weights are frozen, and learning stops – otherwise, if we continue training, the processing time will drastically increase. As a result, it misses the opportunity to learn from the new inputs – and, with the passage of time, the original training data becomes less and less up-to-date, and the quality of this algorithm decreases.

**What we do in this paper.** In this paper, we show that there is a feasible way to solve both problems – of uncertainty propagation and of continuing learning, a way that would not be possible if we simply tried to solve the uncertainty propagation problem by itself.

**Structure of this paper.** We start in Section 2 with a brief reminder of deep learning algorithms. In Section 3, we briefly recall the current uncertainty propagation techniques and explain why they are not always applicable to AI-based algorithms. Finally, in Section 4, we describe the new uncertainty propagation method for AI-based algorithms.
2. Deep learning algorithm: a brief reminder

How a deep neural network processes data. A deep neural network consists of neurons, i.e., devices that transform inputs \( s_1, \ldots, s_m \) into the outputs \( s = a(w_0 + w_1 \cdot s_1 + \ldots + w_m \cdot s_m) \), for some coefficients \( w_i \) known as weights and for some function \( a(z) \) known as activation function. Usually, \( a(z) = \max(0, z) \); this activation function is known as Rectified Linear Unit (ReLU, for short).

Some neurons directly process the data \( x_1, \ldots, x_n \). Other neurons use the outputs of other neurons as their inputs. The output of one of the neurons is then returned as the result \( y \) of data processing.

How a deep neural network is trained. To train a neural network, we use it to process the values \( x_1^{(k)}, \ldots, x_n^{(k)} \) for which we know the value \( y^{(k)} \) of the desired quantity \( y \). When we perform this data processing, we not only compute the value \( y \), we also store all intermediate results. We set up an objective function \( J(y, y^{(k)}) \) whose value is the smallest in the ideal case when the result \( y \) of data processing coincides with \( y^{(k)} \). For example, we can set \( J(y, y^{(k)}) = (y - y^{(k)})^2 \).

Then, we use a special backpropagation algorithm to compute, for each weight \( w \) of each neuron, the corresponding partial derivative \( \frac{\partial J}{\partial w} \).

Once the derivatives have been computed, we update all the weights by using the gradient descent formula

\[
 w \mapsto w - \lambda \cdot \frac{\partial J}{\partial w},
\]

for some appropriately selected value \( \lambda \).

3. Uncertainty propagation: a brief reminder and why this is a problem for AI-based algorithms

What we need to estimate. We need to estimate the accuracy of the results of data processing: how the result \( \tilde{y} = \tilde{f}(\tilde{x}_1, \ldots, \tilde{x}_n) \) of processing measurement results \( \tilde{x}_i \) is different from the ideal value \( y = f(x_1, \ldots, x_n) \) that we would have gotten if we knew the actual values \( x_i \). In other words, we need to estimate the difference

\[
 \Delta y = \tilde{y} - y = \tilde{f}(\tilde{x}_1, \ldots, \tilde{x}_n) - f(x_1, \ldots, x_n).
\]

By definition of \( \Delta x_i \) as the difference \( \Delta x_i = \tilde{x}_i - x_i \), we have \( x_i = \tilde{x}_i - \Delta x_i \). Substituting this expression for \( x_i \) into the formula (4), we conclude that

\[
 \Delta y = \tilde{y} - y = f(\tilde{x}_1, \ldots, \tilde{x}_n) - f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n).
\]

Possibility of linearization. Measurement errors are usually relatively small. As a result, terms which are quadratic – or higher order – in terms of measurement errors are much smaller than
linear terms and can, therefore, be safely ignored. For example, even if we have a not very accurate measurement – with accuracy 10% – the square of 10% is 1% which is an order of magnitude smaller than 10%. Thus, we can do what physicists usually do in such situations (see, e.g., (Feynman et al., 2005; Thorne and Blanford, 2021)): expand the expression (5) in Taylor series in terms of $\Delta x_i$ and keep only linear terms in this expansion. Here,

$$f(\bar{x}_1 - \Delta x_1, \ldots, \bar{x}_n - \Delta x_n) = \bar{y} - \sum_{i=1}^{n} y_i \cdot \Delta x_i,$$

where we denoted

$$y_i \equiv \frac{\partial f}{\partial x_i}|_{x_1=\bar{x}_1, \ldots, x_n=\bar{x}_n}.$$

Substituting this expression into the formula (5), we conclude that

$$\Delta y = \sum_{i=1}^{n} y_i \cdot \Delta x_i.$$

How do we estimate $\Delta y$. Depending on what we know about the measurement uncertainty $\Delta x_i$, we can get similar information about $\Delta y$.

Case of probabilistic uncertainty. In many cases, we know the probability distributions of all measurement errors (and we also know that measurement errors corresponding to different measurements are statistically independent). This means, in particular, that we know the mean $m_i$ (also known as bias) and the standard deviation $\sigma_i$ of each measurement error. Since we know the bias, we can simply subtract this bias from all measurement results and thus get this bias equal to 0. In this case, due to the formula (8), the mean value of $\Delta y$ is also 0, and the standard deviation $\sigma$ of $\Delta y$ is described by the following formula:

$$\sigma^2 = \sum_{i=1}^{n} y_i^2 \cdot \sigma_i^2,$$

Case of partial information about probabilities. In some cases, we only have partial information about the probabilities. In such cases, instead of the exact values of $m_i$ and $\sigma_i$, we only know intervals $[m_i, \bar{m}_i]$ and $[\sigma_i, \bar{\sigma}_i]$ of possible values of these quantities. Similarly to the previous case, we subtract the average value of the bias

$$\frac{m_i + \bar{m}_i}{2}$$

from all the measurement results and thus conclude that the possible values of the remaining bias $m_i$ form the interval $[-b_i, b_i]$, where we denoted

$$b_i \equiv \frac{\bar{m}_i - m_i}{2}.$$
From the formula (5), we conclude that

\[ m = \sum_{i=1}^{n} y_i \cdot m_i. \]  

(12)

One can check that when \( m_i \in [-b_i, b_i] \), the possible values of the mean form an interval \([\overline{m}, \underline{m}]\), where

\[ \overline{m} = \sum_{i=1}^{n} |y_i| \cdot b_i. \]  

(13)

As for the bounds on standard deviation of \( \Delta y \): since the expression (9) is increasing with respect to each \( \sigma_i \):

- the smallest value \( \sigma^2 \) of the expression (9) is attained when all the values \( \sigma_i \) are the smallest, i.e., when for each \( i \), we have \( \sigma_i = \sigma_i \), and
- the largest value \( (\sigma)^2 \) of the expression (9) is attained when all the values \( \sigma_i \) are the largest, i.e., when for each \( i \), we have \( \sigma_i = \sigma_i \).

Thus, we have:

\[ (\sigma)^2 = \sum_{i=1}^{n} y_i^2 \cdot (\sigma_i)^2; \]  

(14)

\[ (\sigma)^2 = \sum_{i=1}^{n} y_i^2 \cdot (\sigma_i)^2. \]  

(15)

**Interval case.** In many other cases, we do not know the probabilities, all we know are bounds \( \Delta_i \) on the absolute values of the measurement errors \( \Delta x_i \): \( |\Delta_i| \leq \Delta_i \). In this case, after we know the measurement result \( \tilde{x}_i \), the only information that we gain about the actual value \( x_i \) is that this value is somewhere in the interval \([\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i] \). Because of this fact, such cases are known as cases of *interval uncertainty*; see, e.g., see, (Jaulin et al., 2012; Kubica, 2019; Mayer, 2017; Moore at al., 2009). In this case, all we can do is find the set of possible values of \( \Delta y \). One can check that this set is an interval \([-\Delta, \Delta] \), where

\[ \Delta = \sum_{i=1}^{n} |y_i| \cdot \Delta_i. \]  

(16)

To use all these formulas, we need to know the derivatives \( y_i \), and this is not easy for AI-based algorithms. All the above formulas use the derivatives \( y_i \). Once we know these derivatives, the remaining computations are straightforward – just add \( n \) easy-to-compute terms. The question is how to compute the desired derivatives \( y_i \).
When the data processing algorithm is complex – as in the case of AI-based algorithms – computing the derivatives is not easy. In such situations, we can compute $y_i$ by using numerical differentiation, i.e., as

$$y_i \approx f(x_1, \ldots, x_{i-1}, x_i + h_i, x_{i+1}, \ldots, x_n) - \tilde{y}$$

for some small $h_i$.

The problem is this approach requires applying the same time-consuming computation of the function $f$ several times:

- first to compute the value $\tilde{y} = f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$, and then
- to compute auxiliary values $f(x_1, \ldots, x_{i-1}, x_i + h_i, x_{i+1}, \ldots, x_n)$.

For AI-based algorithms, the computation time is already high, and it is often not feasible to repeat this procedure several times.

The straightforward formula (12) requires that we repeat the computations $n + 1$ times: one time to compute $\tilde{y}$, and $n$ times to estimate all $n$ derivatives $y_i$. There are techniques – such as Monte-Carlo simulations – that need fewer times, but they still need several applications of the data processing algorithm. It is therefore desirable to come up with an uncertainty propagation method that would not require such repeated applications at all – and this is exactly what we propose.

### 4. What we propose

**Observation.** Usually, the data processing algorithm is applied only when we do not know the actual value $y$. However, in cases when the data processing algorithm is semi-empirical – as is the case of AI-based algorithms – it makes sense to also apply it to situations in which we know $y$. This way, we can check whether this algorithm is correct – and how accurate it is.

**Main idea.** What we propose is, for each set of inputs $x_1, \ldots, x_n$, after computing $\tilde{y}$, to use backpropagation to compute the partial derivatives (2).

For the inputs for which we know the actual value $y^{(k)}$, we can actually apply the gradient descent step (3) and thus, use this set to continue training the algorithm.

For the inputs for which we do not know the actual value $y^{(k)}$, we can simply take, as $y^{(k)}$, some value which is close to – but different from – the computation result $\tilde{y}$. We will show that, based on the derivatives (2), we can then feasible compute the desired derivatives $y_i$.

**Analysis of the problem.** Let us consider neurons in the first layer, i.e., neurons that directly process the inputs $x_1, \ldots, x_n$. For these neurons, we get the output signals

$$s_k = a(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n), \quad k = 1, \ldots, K.$$  

According to the formula for the derivative of the composition, the derivative of the objective function $J$ with respect to each $x_i$ takes the following form:

$$\frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial s_k} \cdot \frac{\partial s_k}{\partial x_i}. \quad (19)$$

REC 2024
Here, due to formula (18), we conclude that
\[ \frac{\partial s_k}{\partial x_i} = a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot w_{ki}, \]  
(20)
where \(a'(z)\), as usual, denotes the derivative of the activation function \(a(z)\). So,
\[ \frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{ki}} \cdot \frac{\partial J}{\partial w_{ki}}. \]  
(21)
What we know from the backpropagation step is the derivatives
\[ \frac{\partial J}{\partial w_{ki}}. \]  
(22)
Due to the same formula for the derivative of the composition, each such derivative has the form
\[ \frac{\partial J}{\partial w_{ki}} = \frac{\partial J}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{ki}}. \]  
(23)
Here, due to formula (18), we conclude that
\[ \frac{\partial s_k}{\partial w_{ki}} = a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot x_i, \]  
(24)
so,
\[ \frac{\partial J}{\partial w_{ki}} = \frac{\partial J}{\partial s_k} \cdot a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot x_i. \]  
(25)
By comparing the formulas (21) and (25), we conclude that
\[ \frac{\partial J}{\partial s_k} \cdot a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot w_{ki} = \frac{\partial J}{\partial w_{ki}} \cdot \frac{w_{ki}}{x_i}. \]  
(26)
Substituting this expression into the formula (21), we conclude that
\[ \frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial w_{ki}} \cdot \frac{w_{ki}}{x_i}, \]  
(27)
i.e.,
\[ \frac{\partial J}{\partial x_i} = \frac{1}{x_i} \cdot \sum_{k=1}^{K} \frac{\partial J}{\partial w_{ki}} \cdot w_{ki}. \]  
(28)
This is the derivative of \(J(y, y^{(k)})\) with respect to \(x_i\). What we want is the derivative \(y_i\) of \(y\) with respect to \(x_i\). Again, due to the same formula for the derivative of the composition, we conclude that
\[ \frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial x_i} = \frac{\partial J}{\partial y} \cdot y_i. \]  
(29)
Thus, we arrive at the following formula for computing $y_i$.

Resulting formula for computing $y_i$.

$$y_i = \frac{1}{x_i \cdot d} \sum_{k=1}^{K} \frac{\partial J}{\partial w_{ki}} \cdot w_{ki},$$

(30)

where we denoted

$$d \overset{\text{def}}{=} \frac{\partial J(y, y^{(k)})}{\partial y}.$$  

(31)

In particular, for $J(y, y^{(k)}) = (y - y^{(k)})^2$, we have $d = 2 \cdot (y - y^{(k)})$.

Once we know the values $y_i$, we can use the feasible formulas from Section 3 to find out how uncertainty is propagated via the AI-based algorithm.

Acknowledgements

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by a grant from the Hungarian National Research, Development and Innovation Office (NRDI), and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Focus Program SPP 100+ 2388, Grant Nr. 501624329.

The authors are greatly thankful to the anonymous referees for valuable suggestions.

References


