How to Propagate Uncertainty via AI Algorithms

Olga Kosheleva and Vladik Kreinovich

University of Texas at El Paso
El Paso, Texas 79968, USA
olgak@utep.edu, vladik@utep.edu

1. Need for data processing

- In many application areas, we need to process data.
- 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 talk, we will denote the data processing algorithm by

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

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

2. 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.
- To come up with these algorithms, we first train a multi-layer neural network on thousands and millions of examples.
- As a result, we 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.

3. 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.
- The result $\widetilde{x}$ of measuring a quantity is, in general, somewhat different from the actual (unknown) value x of this quantity.
- Because of this:
  - the value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ 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.

4. Need for uncertainty propagation (cont-d)

- 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 ± 50 – so we should start exploiting this field, and
  - a situation in which it is 150 ± 200, so that maybe there is no oil at all, and we should perform additional tests.

5. 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.
- For these techniques, for which a several-times increase in computations time is not feasible.

6. 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.
- So, with the passage of time, the original training data becomes less and less up-to-date, and the quality of this algorithm decreases.

7. An additional problem related to AI-based data processing (cont-d)

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

8. How a deep neural network processes data: a brief reminder

- A deep neural network consists of *neurons*, i.e., devices that transform inputs $s_1, \ldots, s_m$ into the outputs $s = a(w_0 + w_1 \cdot s_1 + \ldots + w_m \cdot s_m)$.
- The coefficients w_i are known as *weights*.
- The function $a(z)$ is 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.

9. 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 $J(y, y^{(k)})$ whole value is the smallest 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 partial derivative
$$\frac{\partial J}{\partial w}.$$
- Then, we update all the weights by using, for some appropriately selected value λ, the gradient descent formula
$$w \mapsto w - \lambda \cdot \frac{\partial J}{\partial w}.$$

10. Uncertainty propagation: what we need to estimate

- We need to estimate the accuracy of the results of data processing:
  - how the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of processing measurement results $\widetilde{x}_i$ 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

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(x_1, \ldots, x_n).$$

- By definition of Δx_i as the difference $\Delta x_i = \widetilde{x}_i - x_i$, we have $x_i = \widetilde{x}_i - \Delta x_i$.
- Substituting this expression for x_i into the formula for Δy, we conclude that

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n).$$

11. 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:
  - expand the expression for Δy in Taylor series in terms of Δx_i and
  - keep only linear terms in this expansion.

12. Possibility of linearization (cont-d)

- Here,

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

$$\text{where we denoted } y_i \stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i}_{|x_1=\widetilde{x}_1,\ldots,x_n=\widetilde{x}_n}.$$

- Substituting this expression into the linearized formula for Δy, we conclude that

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

- Depending on what we know about the measurement uncertainty Δx_i, we can get similar information about Δy.

13. 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 σ_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, the mean value of Δy is also 0, and the standard deviation σ of Δy is described by the following formula:

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

14. 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 σ_i, we only know intervals $[\underline{m}_i, \overline{m}_i]$ and $[\underline{\sigma}_i, \overline{\sigma}_i]$ of possible values of these quantities.
- Similarly to the previous case, we can subtract the average value of the bias $\dfrac{\underline{m}_i + \overline{m}_i}{2}$ from all the measurement results.
- Thus, we conclude that the possible values of the remaining bias m_i form the interval $[-b_i, b_i]$, where we denoted

$$b_i \stackrel{\text{def}}{=} \frac{\overline{m}_i - \underline{m}_i}{2}.$$

- From the linearlized formula for Δy, we conclude that

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

15. Case of partial information about probabilities (cont-d)

- One can check that when $m_i \in [-b_i, b_i]$, the possible values of the mean form an interval $[-\overline{m}, \overline{m}]$, where $\overline{m} = \sum\limits_{i=1}^{n} |y_i| \cdot b_i$.
- As for the bounds on standard deviation of Δy: since the above expression is increasing with respect to each σ_i:
    - the smallest value $\underline{\sigma}^2$ of this expression is attained when all the values σ_i are the smallest, i.e., when for each i, we have $\sigma_i = \underline{\sigma}_i$,
    - the largest value $(\overline{\sigma})^2$ of this expression is attained when all the values σ_i are the largest, i.e., when for each i, we have $\sigma_i = \overline{\sigma}_i$.
- Thus, we have:

$$\underline{\sigma}^2 = \sum_{i=1}^{n} y_i^2 \cdot \underline{\sigma}_i^2; \quad (\overline{\sigma})^2 = \sum_{i=1}^{n} y_i^2 \cdot (\overline{\sigma}_i)^2.$$

16. Interval case

- In many other cases, we do not know the probabilities, all we know are bounds Δ_i on the absolute values of the measurement errors Δx_i:

$$|\Delta x_i| \le \Delta_i.$$

- In this case:
  - after we know the measurement result $\widetilde{x}_i$,
  - the only information that we gain about the actual value x_i is that this value is somewhere in the interval $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.
- Because of this fact, such cases are known as cases of *interval uncertainty*.
- In this case, all we can do is find the set of possible values of Δy.
- One can check that this set is an interval $[-\Delta, \Delta]$, where

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

17. Fuzzy case: reminder

- In many practical situations, the information about x_i comes from expert estimates.

- Experts often use imprecise ("fuzzy") natural language words to describe their opinion.

- For example, they may say that x_1 is *close* to 1.0, with accuracy *about* 0.1.

- Computers can easily handle numbers, but they are not that good in processing imprecise words like "close" and "about".

- Lotfi Zadeh proposed technique – that he called *fuzzy* – to describe such knowledge in computer-understandable terms.

- He proposed that for each possible value x_i, we ask the expert to estimate the degree $\mu_i(x_i) \in [0, 1]$ to which this value x_i is possible.

18. How to propagate fuzzy uncertainty: problem

- The function $\mu_i(x_i)$ is known as the *membership function*, or, alternatively as a *fuzzy set*.
- Usually, the function $\mu_i(x_i)$ first increases, then decreases.
- Such fuzzy sets are called *fuzzy numbers*.
- He came up with a natural formula – called *Zadeh's extension principle* – for propagating fuzzy uncertainty via an algorithm

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

- It is known that this formula can be reduced to interval computations.
- Namely, each fuzzy set $\mu(x)$ is uniquely determined by its α-*cuts*

$$\mathbf{x}(\alpha) \stackrel{\text{def}}{=} \{x : \mu(x) \geq \alpha\}.$$

- For fuzzy numbers, all α-cuts are intervals.

19. Fuzzy case reduces to interval case

- It is known that for each α, the α-cut of y can be obtained from α-cuts of x_i by interval computations: $\mathbf{y}(\alpha)=f(\mathbf{x}_1(\alpha),\ldots,\mathbf{x}_n(\alpha))$.
- So how can we propagate fuzzy uncertainty via an algorithm?
- It is sufficient, e.g., for $\alpha=0.1,0.2,\ldots,1$, to perform the corresponding interval computations.

20. What are intuitionistic fuzzy sets?

- The original fuzzy approach does not distinguish two different cases:
  - when we know nothing, and
  - when we have any arguments in favor and equally many against.
- In both cases, we get $\mu(x)=0.5$.
- To get a more adequate description of expert knowledge, Krassimir Atanassov proposed a notion of *intuitionistic fuzzy sets* (IFS).
- Here, for each x_i, we also ask for a degree $\mu_i^-(x_i)$ to which x_i is *not* possible; then:
  - in the first case, $\mu(x)=\mu^-(x)=0$; while
  - in the second case, $\mu(x)=\mu^-(x)=0.5$.

21. How can we propagate intuitionistic fuzzy uncertainty via a data processing algorithm $y = f(x_1, \ldots, x_n)$?

- From the mathematical viewpoint, an IFS is equivalent to an interval $[\mu(x), \overline{\mu}(x)]$ for $\overline{\mu}(x) = 1 - \mu^-(x)$.
- Vice versa, an interval $[\mu(x), \overline{\mu}(x)]$ is equivalent to IFS with
$$\mu^-(x) = 1 - \overline{\mu}(y).$$
- An interval means that value of the actual membership function $\nu(x)$ can be anywhere in this interval.
- For different $\nu_i(x_i)$ from the corresponding intervals, we have, in general, general $\nu(y)$.
- One can show that the smallest value $\nu(y)$ is attained when $\nu_i(x_i)$ are the smallest: $\nu_i(x_i) = \mu_i(x_i)$;
- Similarly, the largest value $\nu(y)$ is attained when $\nu_i(x_i)$ are the largest:
$$\nu_i(x_i) = \overline{\mu}_i(x_i).$$

22. How can we propagate IFS uncertainty via data processing (cont-d)

- So, to propagate IFS via an algorithm, we:
  - apply Zadeh's extension principle to $\mu_i(x_i)$ and get $\mu(y)$; and
  - apply Zadeh's extension principle to $\overline{\mu}_i(x_i) = 1 - \mu^-(x_i)$ and get $\overline{\mu}(y)$;
  - then, we compute $\mu^-(y) = 1 - \overline{\mu}(y)$.
- The first two steps can be reduced to interval computations.
- So, the IFS case can also be reduced to the interval case.

23. 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

$$y_i \approx \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - \widetilde{y}}{h_i} \text{ for some small } h_i.$$

24. To use all these formulas, we need to know the derivatives y_i, and this is not easy for AI-based algorithms (cont-d)

- The problem is this approach requires applying the same time-consuming computation of the function f several times:
  - first to compute the value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$, and then
  - to compute auxiliary values $f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$.
- For AI-based algorithms, the computation time is already high, and it is often not feasible to repeat this procedure several times.
- The above straightforward formula requires that we repeat the computations $n + 1$ times:
  - one time to compute $\widetilde{y}$,
  - and n times to estimate all n derivatives y_i.
- There are techniques – such as Monte-Carlo simulations – that need fewer times.

25. To use all these formulas, we need to know the derivatives y_i, and this is not easy for AI-based algorithms (cont-d)

- However, these techniques 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.
- This is exactly what we propose.

26. 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.

27. Main idea

- What we propose is as follows:
  - for each set of inputs $x_1, \ldots, x_n$, after computing $\widetilde{y}$,
  - to use backpropagation to compute the partial derivatives $\partial\frac{\partial J}{\partial w_i}$.
- For the inputs for which we know the actual value $y^{(k)}$:
  - we can actually apply the gradient descent step and thus,
  - use this step 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 $\widetilde{y}$.
- We will show that, based on the derivatives $\partial\frac{\partial J}{\partial w_i}$, we can then feasible compute the desired derivatives y_i.

28. 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:
$$\frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial s_k} \cdot \frac{\partial s_k}{\partial x_i}.$$
- Here, due to the previous formula, we conclude that
$$\frac{\partial s_k}{\partial x_i} = a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot w_{ki}.$$

29. Analysis of the problem (cont-d)

- Here, $a'(z)$, as usual, denotes the derivative of the activation function $a(z)$; so:

$$\frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial s_k} \cdot a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot w_{ki}.$$

- What we know from the backpropagation step is the derivatives

$$\frac{\partial J}{\partial w_{ki}}.$$

- Due to the same formula for the derivative of the composition, each such derivative has the form

$$\frac{\partial J}{\partial w_{ki}} = \frac{\partial J}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{ki}}.$$

- Substituting the known expression for $\dfrac{\partial s_k}{\partial x_i}$, we conclude that

$$\frac{\partial s_k}{\partial w_{ki}} = a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot x_i.$$

30. Analysis of the problem (cont-d)

- So:

$$\frac{\partial J}{\partial w_{ki}} = \frac{\partial J}{\partial s_k} \cdot a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot x_i.$$

- By comparing this formula with the formula for $\dfrac{\partial J}{\partial x_i}$, we conclude that

$$\frac{\partial J}{\partial s_k} \cdot a'(w_{k0} + w_{k1} \cdot x_1 + \ldots + w_{kn} \cdot x_n) \cdot w_{ki} = \frac{\partial J}{\partial w_{ki}} \cdot \frac{w_{ki}}{x_i}.$$

- Substituting this expression into the formula for $\dfrac{\partial J}{\partial x_i}$, we conclude that

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

31. Analysis of the problem (cont-d)

- So:
$$\frac{\partial J}{\partial x_i} = \frac{1}{x_i} \cdot \sum_{k=1}^{K} \frac{\partial J}{\partial w_{ki}} \cdot w_{ki}.$$
- 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
$$\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial x_i} = \frac{\partial J}{\partial y} \cdot y_i.$$
- Thus, we arrive at the following formula for computing y_i.

32. Resulting formula for computing y_i

- The final formula is:

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

- Here we denoted

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

- 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 above feasible formulas to find out how uncertainty is propagated via the AI-based algorithm.

33. Acknowledgments

This work was supported in part by:

- Deutsche Forschungsgemeinschaft Focus Program SPP 100+ 2388, Grant Nr. 501624329;
- National Science Foundation grants 1623190, HRD-1834620, HRD-2034030, and EAR-2225395;
- AT&T Fellowship in Information Technology;
- program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and
- a grant from the Hungarian National Research, Development and Innovation Office (NRDI).