Executive Summary

What is explainable AI? Why do we need explainable AI? Explainable AI is when an AI system not only provides *recommendations*, it also provides *explanations* for these recommendations.

Why do we need it? Many AI programs – in particular, the ones that use deep learning – just provide a recommendation, they do not come with any explanation. We know that these programs are not perfect, that sometimes their recommendations are wrong – but since there are no explanations, we do not know which recommendations are wrong. It is therefore desirable to have such explanations.

Why does it make sense to use fuzzy techniques in explainable AI? Desire for explanations means that we need to be able to transform numerical recommendations into natural-language explanations. In other words, we need to connect numerical recommendations with natural-language rules.

Such a connection has been explored before: this is exactly what fuzzy techniques are about. Fuzzy techniques were designed by Lotfi Zadeh who realized that a large part of expert experience – namely, the rules the experts formulate in terms of imprecise words from natural language – are not used in automatic control. So, he designed fuzzy techniques to translate experts' natural-language rules into precise control strategies.

What are the main steps of fuzzy techniques.

- First, for each natural-language term P like "small" used by experts, we ask the expert, for different inputs x, to provide his/her degree of confidence that this value x satisfies the corresponding property (e.g., the degree to which x is small). This way, we get the degrees $\mu_P\left(x^{(k)}\right)$ corresponding to finitely many values $x^{(1)} < \ldots < x^{(K)}$. Then, we use interpolation – usually, linear interpolation – to estimate the degrees $\mu_P(x)$ corresponding to other values x. For values x between $x^{(k)}$ and $x^{(k+1)}$, linear interpolation has the form

$$\mu_P(x) = \mu_P\left(x^{(k)}\right) + \frac{\mu_P\left(x^{(k+1)}\right) - \mu_P\left(x^{(k)}\right)}{x^{(k+1)} - x^{(k)}} \cdot \left(x - x^{(k)}\right).$$

- For each rule R_i of the type

"if $P_{i1}(x_1)$ and ... and $P_{in}(x_n)$, then $P_i(u)$",

we estimate the degree $r_i(u)$ to which this rule is applicable, as

$$r_i(u) = f_{\&}(\mu_{P_{i1}}(x_1), \ldots, \mu_{P_{in}}(x_n), \mu_{P_i}(u)).$$

After that, we compute the degree $\mu(u)$ to which the control u is reasonable as

$$\mu(u) = f_{\vee}(r_1(u), r_2(u), \ldots).$$

- Finally, for automatic control, we transform the fuzzy set $\mu(u)$ into a single control value $\overline{u}$. This defuzzification is usually performed by applying the centroid defuzzification formula

$$\overline{u} = \frac{\int u \cdot \mu(u)\, du}{\int \mu(u)\, du}.$$

Which interpolation algorithm should we use when generating a membership function and why? When we know the values $\mu(a)$ and $\mu(b)$, then to find values $\mu(x)$ for intermediate values x, we should use linear interpolation

$$\mu(x) = \mu(a) + \frac{\mu(b) - \mu(a)}{b - a} \cdot (x - a).$$

It is selected to minimize the effect of measurement uncertainty – due to which for the same actual value of the quantity, we may have somewhat different measurement results – on the result. In precise terms, for the values

$$x_1 = a,\ x_2 = x_1 + \Delta x,\ \ldots,\ x_n = x_{n-1} + \Delta x = b,$$

we want to make sure that $\mu(x_{i+1}) \approx \mu(x_i)$, i.e., that the squared distance

$$D^2 = (\mu(x_2) - \mu(x_1))^2 + (\mu(x_3) - \mu(x_2))^2 + \ldots + (\mu(x_n) - \mu(x_{n-1}))^2$$

between the tuples formed by the left-hand and right-hand sides of these approximate equalities is as small as possible.

Which "and"- and "or"-operations should we use and why. Expert's degrees are also approximate, they depend on the scale: a degree 0.8 corresponding to 5 on a 0-to-5 scale is not equal to any value coming from the 0-to-4 scale, on that scale, the closest value is $3/4 = 0.75$. Since close degree may correspond to the exact same expert opinion, we want the difference in degrees to minimally affect our results: if $a \approx a'$, then $f_{\&}(a, b) \approx f_{\&}(a', b)$ and $f_{\vee}(a, b) \approx f_{\vee}(a', b)$.

When we control a *group* of objects, and we want to achieve the best overall result, we need to make sure that the distance between the corresponding tuples is the smallest possible, which leads to

$$f_{\&}(a, b) = a \cdot b \text{ and } f_{\vee}(a, b) = a + b - a \cdot b.$$

If we are controlling an individual object, and we want to achieve the best result for this object, we need to make sure that *all* the differences between the resulting values of "and"- and "or"-operations are small, i.e., that

$$|f_{\&}(a,b) - f_{\&}(a',b')| \le K \cdot \max(|a-a'|, |b-b'|)$$

and

$$|f_{\vee}(a,b) - f_{\vee}(a',b')| \le K \cdot \max(|a-a'|, |b-b'|)$$

for the smallest possible value K. This leads to

$$f_{\&}(a,b) = \min(a,b) \text{ and } f_{\vee}(a,b) = \max(a,b).$$

What defuzzification procedure should we use and why? Let

$$\mu(u_1), \mu(u_2), \ldots$$

be the degrees to which the values $u_1, u_2, \ldots$ are reasonable. In a polling scheme, this means that out of N experts, $N \cdot \mu(u_i)$ consider the value u_i to be reasonable. We want the value $\overline{u}$ which is the closest to the opinions of all experts, i.e., for which:

- $\overline{u} \approx u_1$ for $N \cdot \mu(u_1)$ experts,
- $\overline{u} \approx u_2$ for $N \cdot \mu(u_2)$ experts, etc.

It is natural to interpret it as saying that the squared distance between the tuple $(\overline{u}, \overline{u}, \ldots)$ formed by the left-hand sides of all these approximate equalities and the tuple $(u_1, \ldots, u_1, u_2, \ldots, u_2, \ldots)$ formed by its right-hand sides is the smallest possible. This leads to the following formula – known as centroid defuzzification:

$$\overline{u} = \frac{u_1 \cdot \mu(u_1) + u_2 \cdot \mu(u_2) + \ldots}{\mu(u_1) + \mu(u_2) + \ldots} \approx \frac{\int u \cdot \mu(u)\, du}{\int \mu(u)\, du}.$$

How fuzzy techniques can be used in explainable AI? We start with expert rules – this what makes this approach explainable. We then use general fuzzy methodology – explained in the previous lectures – to find the first-approximation dependence $y = f(x_1, \ldots, x_n)$.

When applying the fuzzy methodology, we used some parameters – e.g., for negligible, we selected 5 as the borderline value starting with which the difference is absolutely not negligible. The choice of these parameters is rather arbitrary. For example, to describe what is negligible, we could use 4 or 6 instead of 5.

So, instead of picking a single such value:

- we make this value a parameter, and then
- we find the values of all these parameters for which, for each k, the predictions of the resulting fuzzy system are the closest to the desired value $y^{(k)}$.

How can we make machine learning itself more explainable: by providing theoretical explanations for deep learning's empirical choices.

First such explanation: why rectified linear activation functions. In a neural network, signals interchangingly undergo:

- linear transformations and
- non-linear transformations.

If we only had linear transformations, we would be able to only compute linear functions, and many real-life dependencies are nonlinear. So, we need nonlinear transformations. The corresponding nonlinear transformations are known as activation functions.

In deep learning, the following activation function is used:

$$F(x) = \max(0, x),$$

known as *rectified linear* function (ReLU, for short).

A deep neural networks has many layers which work one after another. Some of these layers perform linear combination, some apply the activation function. Thus, the time needed for the deep neural network to produce the result is much larger than for the traditional neural network. How can we save time?

- There is not much that can do to speed up the computation of a linear combination: we already apply the fastest possible algorithms for this.
- However, the time needed to compute an activation function differs: some nonlinear functions are faster to compute, for other, computations require a much longer time.

So, to save time, a reasonable idea is to select an activation function which is the fastest to compute. Whatever we compute consists of the hardware supported operations.

- The more operations we perform, the longer it takes.
- So, to make computations faster, it is desirable to use as few operations as possible.

It is therefore desirable to use just one such operation – and the fastest, which leads to min and max. The input can be x or a constant.

- it makes no sense to compute $\min(x, x)$ or $\max(x, x)$ – since both expressions are equal to x;
- so, we end up with $\min(x, c)$ or $\max(x, c)$.

The fastest-to-generate constant is 0 – since it is the default contents of the cells. So, we end with

$$\max(x, 0) \text{ or } \min(x, 0).$$

Second such explanations: why max- and sum-poolings. One of the main applications of neural networks is to process pictures. In a computer, a picture is represented by storing intensity values – or, for color pictures, intensity values corresponding to three basic colors – for each pixel, and there are millions of pixels. Processing all these millions of values would take a lot of time.

To save this time, we can use the fact that for most images:

- once we know what is in a given pixel,
- we can expect approximately the same information in the neighboring pixels.

Thus, to save time, instead of processing each pixel one by one, we can combine ("pool") values from several neighboring pixels into a single value.

Empirically, the following poling work the best: max-pooling $\max(a, b)$ and sum-pooling $a + b$ – or, equivalently, averaging $\dfrac{a+b}{2}$. Why?

The whole objective of pooling is to speed up data processing. From this viewpoint, we need to select a pooling operation which the fastest to perform.

This means that we need to select a pooling operation which is performed by using the smallest possible number of hardware supported computer operations, and these operations should be the fastest. If we use only one hardware supported operation, we get

$$\min(a, b),\ \max(a, b),\ \text{and}\ a + b.$$