Test 3 for Explainable Fuzzy AI Class
Solutions, Summer 2021

Question 1. Suppose that a 2-layer neural network has two inputs $x_1 = -2$ and $x_2 = 2$.

- In the first layer, we perform a linear transformation and compute the value $y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$.
- In the second layer, we apply, to the result of the first layer, the rectified linear activation function and get $z = f(y)$.

What will be the result z of this data processing in the following two situations:

- when $w_0 = 0$, $w_1 = 1$, and $w_2 = 2$, and
- when $w_0 = 0$, $w_1 = -1$, and $w_2 = -2$.

Answer. For $w_0 = 0$, $w_1 = 1$, and $w_2 = 2$:

- after the first layer, we get

$$y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0 + 1 \cdot (-2) + 2 \cdot 2 = 0 - 2 + 4 = 2;$$

- then, in the second layer, we get

$$z = F(y) = \max(0, y) = \max(0, 2) = 2.$$

For $w_0 = 0$, $w_1 = -1$, and $w_2 = -2$:

- after the first layer, we get

$$y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0 + (-1) \cdot (-2) + (-2) \cdot 2 = 0 + 2 - 4 = -2;$$

- then, in the second layer, we get

$$z = F(y) = \max(0, y) = \max(0, -2) = 0.$$

Question 2a. What is an activation function and why do we need it?

Answer. 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.

Question 2b. What activation function was used in traditional neural networks and why?

Answer. In the traditional neural networks, mostly, the following activation function is used – $F(x) = \dfrac{1}{1+\exp(-x)}$. This function – known as *sigmoid* or *logistic* activation function – was selected because it adequately reflects how signals are processed in most biological neurons.

Question 2c. What activation function is used in deep learning?

Answer. In deep learning, a different type of neural networks turned out to be much more efficient: the function

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

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

Question 2d. What operations are hardware supported on a computer? How does a computer computes $\exp(x)$?

Solution. In modern computers, only a few operations are hardware supported: namely,

- minimum $\min(a, b)$ and maximum $\max(a, b)$; these two are the fastest;
- sum $a + b$ which takes somewhat longer to compute, and
- product $a \cdot b$ which takes the longest – since a usual multiplication algorithm includes several additions.

Everything else is implemented as a combination of these elementary operations.

For example:

- When you ask a computer to compute $\exp(x)$, it actually computes the sum of the first few terms of the Taylor series of the function $\exp(x)$:

$$\exp(x) \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^N}{N!} \quad \text{(for some } N\text{)}.$$

- Division a/b is computed as $a \cdot (1/b)$, and the inverse $1/b$ is computed by an iterative procedure that consists of several additions and multiplications.

Question 2e. Explain why rectified linear activation functions work the best.

Answer. 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 $\max(x, 0)$ or $\min(x, 0)$.

Question 3a. What is pooling and why do we need it?

Answer. Pooling is when we replace several values $a_1, \ldots, a_n$ with a single value.

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.

Question 3b. Which poolings are used in deep learning?

Answer. Deep learning uses:

- *max-pooling*, when we combine two values a and b into a single value $\max(a, b)$, and
- *averaging*, when we combine two values a and b into their arithmetic average $(a + b)/2$; this is almost equivalent to *sum-pooling*, when we combine two values a and b into their sum $a + b$.

Question 3c. If we pool together the values $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$, what will be the result of max-pooling? sum-pooling?

Answer. Max-pooling leads to $\max(1, 2, 3) = 3$, and sum-pooling to $1+2+3 = 6$.

Question 3d. Explain why max-pooling and sum-pooling work the best.

Answer. 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 $\min(a, b)$, $\max(a, b)$, and $a + b$.