Synthesis of Energy-based Out-of-distribution Detection

Martin BENITO-RODRIGUEZ

1 Introduction

AIs are more and more present in our lives and sometimes make mistakes that can be costly, but there are solutions to prevent these mistakes. The article [2] proposes a model based on energy for a better detection of out-of-distribution in neural networks. We will see together what is a neural network, the softmax function, the out-of-distribution, the cost functions and the use of energy.

2 What is deep learning ?

Deep learning is a technology belonging to the field of machine learning which has the particularity of using a neural network to make predictions.

A neural network is composed of several successive layers, a layer being a set of neurons.

The first layer corresponds to the input of our network, the last layer corresponds to the output, between the two, there are several layers that we call "hidden layers". In a classification problem, each output corresponds to a class, the neural network must associate a probability to each class.

The value of a neuron is determined by the value of the neurons in the previous layer. In the most classical neural network, the "feedforward neural network", each neuron in layer i is connected to all neurons in layer $i-1$ via connectors called synapses. Thus the successive layers from input to output will have their neurons activated.

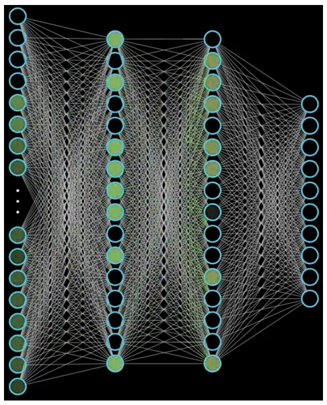

Figure 1: A neural network working.
But what is a Neural Network? Deep learning, chapter 1 — 3Blue1Brown

For each synapse, we assign a *weight* which acts as a coefficient on the value of the neuron of the previous layer. To know the value of a neuron, it is necessary to calculate the sum of each neuron of the preceding layer weighted with the weight of its synapse. We can also add a *weight* to each layer which manifests itself as an addition that allows us to correct the value.

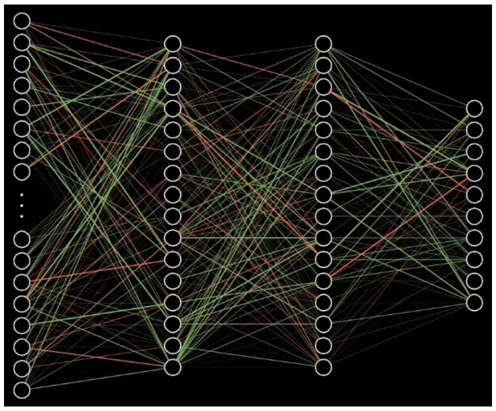

Figure 2: A weighted neural network, the more visible the synapse, the higher the weight.
But what is a Neural Network? Deep learning, chapter 1 — 3Blue1Brown

Since each neuron contains a number, and each layer contains neurons, we can represent a layer as a vector of numbers. Also, since each neuron depends on the previous layer, we can represent each neuron as a function that takes as parameter a vector and makes calculations with the vector, the weights and the bias to get a value.

In the end, we can represent the neural network as a function which takes as input a vector and the parameters of weights and bias, and gives as output a vector (the size of the vector corresponds to the number of classes).

The training consists in finding the most optimal weights and biases to make accurate predictions. At the beginning of the training, the weights and biases have random values, the neural network will make first predictions that are probably wrong. To make it understand this, we need a cost function that measures the difference between the prediction and reality.

With each prediction, the neural network will then modify its weights and biases in order to minimise the cost function and thus make increasingly accurate predictions. [4]

To make the last layer easily understandable to a human, the value of each of its neurons can represent a probability of class membership. One of the tools for establishing these probabilities is the softmax function.

3 Softmax

The softmax function (called Boltzman distribution in physics or Gibbs distribution in mathematics) is used in particular in the last layer of a neural network. It takes as parameter a vector $z = (z_1, ..., z_K) \in \mathbb{R}^k$ and normalizes it into a probability distribution with K probabilities proportional to the exponential of the input number. The K probabilities represent the probabilities of the K classes of being associated with the input.

$$\sigma(z)_i = \frac{e^{z_i/T}}{\sum_{j=1}^{K} e^{z_j/T}} \tag{1}$$

Normalisation allows the sum of the output vector to be equal to 1. This allows us to see what proportion each value of e^{z_i} has in the sum.

The temperature parameter $T \in \mathbb{R}^+$ allows to flatten the curve, the higher T is, the flatter the distribution is. From a physical point of view, by comparing the softmax function with the Boltzmann distribution, we notice that the temperature T of softmax does not correspond to a temperature but to an energy. We can see the flattening of the curve as a function of T by taking the extreme values.

$$\lim_{T \to 0} \frac{e^{z_i/T}}{\sum_{j=1}^{K} e^{z_j/T}} = argmax(z)$$

$$\lim_{T \to +\infty} \frac{e^{z_i/T}}{\sum_{j=1}^{K} e^{z_j/T}} = \frac{1}{\sum_{j=1}^{K} 1} = \frac{1}{K}$$

When T tends to 0, only one class has a probability of 1 and all the others have a zero probability, whereas when T tends to $+\infty$, the distribution is uniform.

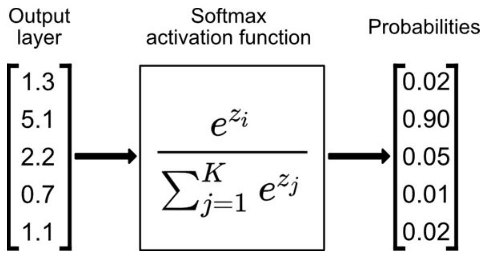
Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Figure 3: Softmax example with T=1.
From: Dario Radečić, Softmax Activation Function Explained, 2020, Towards Data Science

An exponential is used to exaggerate the differences. The function will return a value close to 0 when z_i is much smaller than the other values, and it will return a value close to 1 if it is much larger than the other values (see Figure 3). This allows the construction of a weighted average that acts as a smooth function.

The softmax function is used to normalise the output of a neural network, its input will be a vector corresponding to the last layer of the neural network. Since we can represent the neural network as a function we can write the softmax function like this:

$$p(y|x) = \frac{e^{f_y(x)/T}}{\sum_{j=1}^{K} e^{f_j(x)/T}} \tag{2}$$

With the neural network $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}^K$, and $f_y(x)$ the y^{th} value of the output vector of $f(x)$.

$p(y|x)$ is therefore the probability of obtaining class y with input x from the neural network f.

4 Out-of-distribution

Programs using machine learning are trained with sample data, we expect that when our machine is called upon, the data it will have to process will be similar to the training data. But this is not always the case, sometimes the machines face such rare and exceptional cases that they have not been trained to react to this situation. If I give my animal detector an image of an aeroplane, maybe it will say that it is a bird with a high probability. We need to find a way to detect whether the input is what the machine can handle or whether it is an unclassifiable case.

To test our model's ability to know if an input is an out-of-distribution input, we need to give it data that it has been trained to handle and data that it has not been trained to handle, called out-of-distribution. It will then try to find out which data belong to which distribution. We can observe the detection capacity with a mapping of our model of the 2 distributions, the smaller the area common to the 2 distributions, the less confusions there will be and the better our model will be at detecting out-of-distribution.

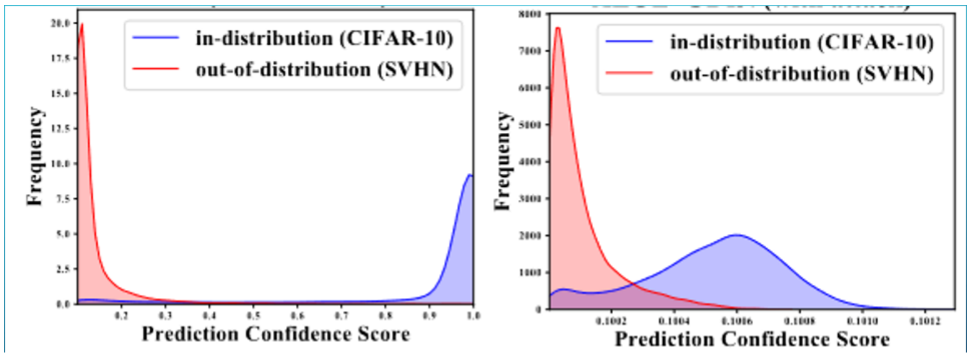

Figure 4: Confidence score distributions. It can be seen that the model on the left-hand curve separates the two distributions much better than the model on the right-hand curve and is therefore more efficient.
[3]

5 Costs functions

A cost function is a function that seeks to estimate the reliability of a model in relation to reality. During training, the cost function signals to the neural network its reliability, which modifies its weights and biases to minimise its cost.

To evaluate our model, we seek to minimise the difference between our estimated distribution $\hat{Y}$ and the actual distribution Y.

A common cost function is the square of the mean error:

$$MeanSquaredError(MSE) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3}$$

The MSE is between 0 and 1, the closer it is to 0 the more accurate the model.

Another cost function is the maximum likelihood [1]:

$$L = \prod_i \hat{y_i}^{y_i} \tag{4}$$

The maximum likelihood is between 0 and 1, the closer it is to 1, the more accurate the model.

Here is an example with the data: $\hat{y} = (0.1, 0.2, 0.7)$ et $y = (0, 0, 1)$.

$MSE = \frac{1}{3}(0.01 + 0.04 + 0.09) = 0.046$
$L = 0.1^0 \times 0.2^0 \times 0.7^1 = 0.7$

Another cost function can be derived from the maximum likelihood: the cross-entropy which is the negative of the logarithm of the maximum likelihood:

$$EC = -\log L = -\log \prod_i \hat{y_i}^{y_i} = -\sum_i y_i \log \hat{y_i} \tag{5}$$

6 Energy

Energy-based models are machine learning models. They are based on the encoding of dependencies between variables and according to these known variables, one seeks the unknown variables which minimises a measure which one calls energy. article explains what energy-based learning is and the different cost functions that can be used.

The challenge of [2] is to try to exploit the energy to identify out-of-distribution input. As a first step, we can compare the out-of-distribution detection of our model by testing the softmax function and a function based on the energy in the last layer role of the neural network. We can also train models to detect out-of-distribution.

The energy can be expressed as the negative of the logarithm of the denominator of the softmax function (Eq.2):
[1]

$$E(x; f) = -T. \ln \sum_i^K e^{f_i(x)/T} \tag{6}$$

This function can be analysed.

Let $v(x) : \mathbb{R} \rightarrow \mathbb{R}^K$, a vector of size k with the same value x on all its components.

$$\lim_{f(x) \rightarrow v(-\infty)} E(x; f) = +\infty$$
$$f(x) = v(0) => E(x; f) = -T \ln k$$
$$\lim_{f(x) \rightarrow v(+\infty)} E(x; f) = -\infty$$

We can see that the higher the output values of the neural network, the lower the energy, which means that the lower the energy, the less likely the input is an out-of-distribution. But the out-of-distribution detectors assign the out-of-distribution to a low value of their function, that's why to compare the detectors between them, we will use the negative of the energy: $-E(x; f)$.

In reality, $-E(x; f)/T$ is linearly aligned with the logarithm of the maximum likelihood, which allows efficient out-of-distribution detection.

Energy is a unitless measure. To make energy comparisons, the most consistent way is to make a normalised distribution. The most common and simple way to do this is to use a Gibbs distribution:

[1]The article did not specify which logarithm, but it seems logical to me that it is the neperian logarithm.

$$p(x) = \frac{e^{-E(x;f)/T}}{\int_x e^{-E(x;f)/T}} \tag{7}$$

We can also train a model with energy. To do this, we need to give the neural network the objective of creating an energy gap between expected and unexpected data. This can be done by assigning low energy to expected distribution data and high energy to out-of-distribution data.

7 Results

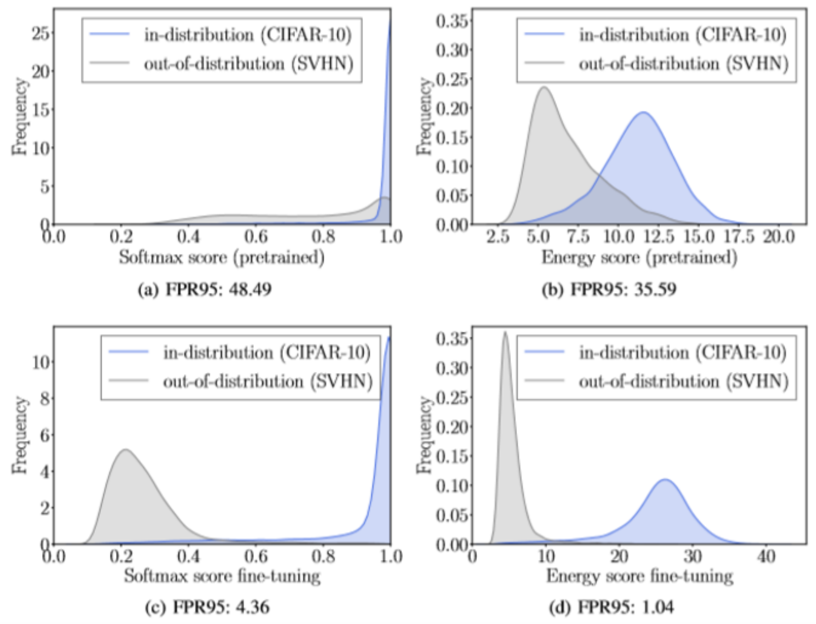

Figure 5: Distribution of softmax (left) vs energy with T=1 (right). Top: results for pre-trained neural networks, bottom: results for fine-tuned neural networks with the classifier.
[2]

It can be seen graphically that the distributions obtained with energy are much better disjointed than the softmax distributions. This is confirmed by the FPR95 scores (false positive rate when there are 95

8 Conclusion

The paper shows a promising solution that seems to be quite effective in detecting out-of-distribution in neural networks, whether with a pre-trained model or an already trained model. The proposed solution could allow us to have more reliable machines at a time when they have more and more important roles in society.

9 Acknowledgment

I would like to thank Violaine ANTOINE who sponsored me during this EU. Her time and advice were invaluable to me.

References

[1] L. Le Cam. *Maximum Likelihood: An Introduction.* 1990.

[2] Weitang Liu et al. *Energy-based Out-of-distribution Detection.* 2020.

[3] Xi Wu Yingyu Liang Somesh Jha Jiefeng Chen, Yixuan Li. *Robust Out-of-distribution Detection for Neural Networks.* 2020.

[4] Jürgen Schmidhuber. *Deep learning in neural networks: An overview.* 2014.