December, 2022

IA703

Algorithmic Information & Artificial Intelligence

Micro-study

teaching.dessalles.fr/FCI

Name: Alban TCHIKLADZE

Adapting the K-complexity for neural networks

Abstract

I have come to the conclusion that Kolmogorov complexity is not well adapted to neural networks. You will find in this article an analysis of the different factors that led to this conclusion. Based on this conclusion, it is necessary to find an adaptation of the K-complexity for neural networks. To further our research, we will try to find a computable approximation. In practice, this approximation should be applied to our K-complexity adaptation for neural networks.

Problem

We will define how the Kolmogorov complexity can be adapted to neural networks. As a reminder, the K-complexity defines the most compact program, in terms of length, that leads to the expected result. It is an evolution of Occam's razor principle. In mathematical terms, it is defined as follows : $C_U(s) = min_p \ |p| \ : \ U(p) = s$, with U a Turing machine, p a program, and |p| the program length.

We assume that neural networks can be considered as a kind of program. So let us replace p by n, where n is a subset of p, and represent the space of all possible neural networks. We then obtain : $C_U(s) = min_n \ |n| \ : \ U(n) = s$ in which $n \subset p$

Generally, neural networks are used for classification, so let's replace s by cl, which is a subset of s and represents the classes of a data array. The K-complexity is then written as follows $C_U(cl) = min_n \ |n| \ : \ U(n) = cl$ in which $n \subset p$ and $d \subset s$. We now have a less general version of K-complexity, derived directly from the original definition of K-complexity, which applies only to the space of neural networks and the space of neural network solutions.

However, there are problems with this definition, which can be divided into two parts. First, the neural network provides probabilities instead of definitive answers. Second, neural networks make mistakes and never lead to the exact result expected.

It is for these reasons that the classical notion of K-complexity *stricto sensus* falters in the case of neural networks. Therefore, we have shown that the K-complexity must be adapted. This will be achieved by redefining the K-complexity to better fit the specificities of neural networks.

Method

Let's recap what we have achieved so far. We have found that a version of neural networks of complexity K can be written as follows $C_U(cl) \ = \ min_n \ \ |n| \ : \ U(n) \ = \ cl$.

However, this definition has two problems, so let try to adapt it :

First, the neural network provides probabilities instead of definitive answers, which means that there are multiple answers for a given input. Most of the time, the answer will be the most probable. This means that our equation must be transformed as follows :
$C_U(cl) \ = \ min_n \ \ |n| \ : \ max(U(n)) \ = \ cl$ as $U(n)$ becomes a list.

Second, there is always a percentage of error. This means that novel data networks are not deterministic and sometimes make mistakes. Therefore, they can never achieve exactly the expected result. This leads us to a further modification of the K-complexity :
$C_U(cl) \ = \ min_n \ \ |n| \ : \ max(U(n)) \ \rightarrow \ cl$ where the term $max(U(n)) \ \rightarrow \ s$ represents the extent to which the results of the neural network are close to the solution. The implicit assumption is that this measure can be considered as an approximation of the expected results. This allows us to extend the K-complexity and adapt this fuzzy notion to neural networks.

We now have a version of the K-complexity that is adapted to neural networks by taking into account the specificities of this type of program. Let's now move on to the computational part. Although it is a great achievement to have a version of K-complexity for neural networks, it is more practical to find a way to approximate it.

Experimental protocole

In essence, the K-complexity captures the efficiency of a program, as it combines the notions of compactness and efficiency. Thus, to get an idea of the efficiency of a neural network, we can compute the ratio between its proximity to the expected results, represented by $max(U(n)) \ \rightarrow \ cl$ and the size he is taking, which represented by $|n|$. Let's thus approximate the complexity by $C(cl) \approx \frac{max(U(n)) \ \rightarrow \ cl}{|n|}$. With this formula, we hope that the higher the result, the closer the neural network will be to the ideal K complexity.

All that remains is to find estimators to compute the two elements of our formulas, the proximity of the expected results and the size of the network.

The closeness of the expected results can be represented in two ways.

The most natural estimator would be the number of correct predictions obtained by the network, which corresponds to the accuracy. But it does not take into account all possible error elements.

Therefore, loss can be a less obvious, but more accurate estimator, as it indicates the difference between the actual and the desired state of the network. However, loss is inversely proportional to accuracy, and if both choices are possible, one of the two must be reversed to preserve the meaning of the results.

On the other hand, the size of the neural network can be represented in four ways.

The computational description length of the network can be considered the closest measure of K-complexity. However, it is almost impossible to compute, because much of the code is embedded in built-in functions and is very difficult to access. However, it is almost impossible to compute, because much of the code is nested in built-in functions and is very difficult to access. In addition, we have to break it down to the byte computation phase of the Turing machine, which is also a challenge in itself. For all these reasons, this part will not be implemented in the experiment.

The number of neurons in the network can give us a quick idea. While this representation can sometimes be useful, it does not take into account how the neurons are connected to each other, and leaves most of the density uncontrolled.

A more accurate measure would be to examine the number of parameters. This quantity is more useful, as it represents almost all of the information in the network.

Finally, the size of the memory occupied by the network can also be an interesting measure, as it takes into account some of the more conventional coded components that may be embedded in the network.

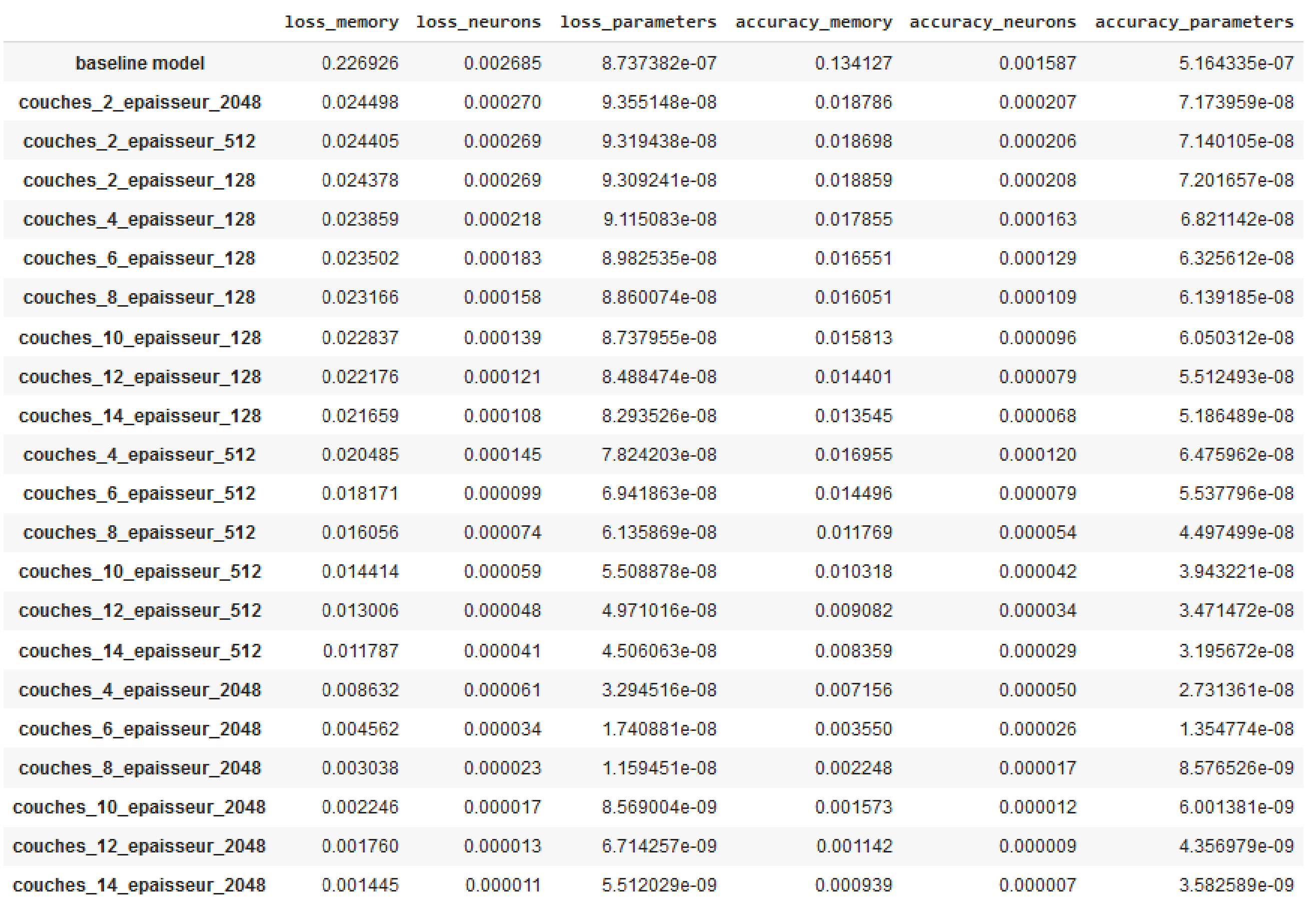
Now that we have many estimators for our quantities, let's create a code that can compute them based on the network information. To get comparable results, some assumptions must be made. As a dataset, we will take the CIFRA-100 small image classification, available on Keras. It has the advantage of being complex enough to allow a wide range of results. After some pre-processing, the data will be fed into the network. The whole network will be composed of a dense layer of 100 neurons with the softmax function as output, which will act as a classifier. Thus, the network composed of this single layer will be taken as our baseline, to evaluate the improvements of the results. All networks will be trained using a batch size of 64, for 20 epochs, with an early stop of one epoch to avoid overfitting.

The networks used in this experiment will differ in length, ranging from 128 to 4086, in steps of a power of 4, and in width, ranging from 0 to 14, in steps of 2. This evolution does not include the necessary layer, which will be in all networks. These are the dense input layer, size 1024, and the dense classifier, size 100.

After completing the training and evaluation, the complexity will be calculated using the special module developed to compute the neural network K-complexity approximation.

Results

Here are the raw results obtained after the experiment :

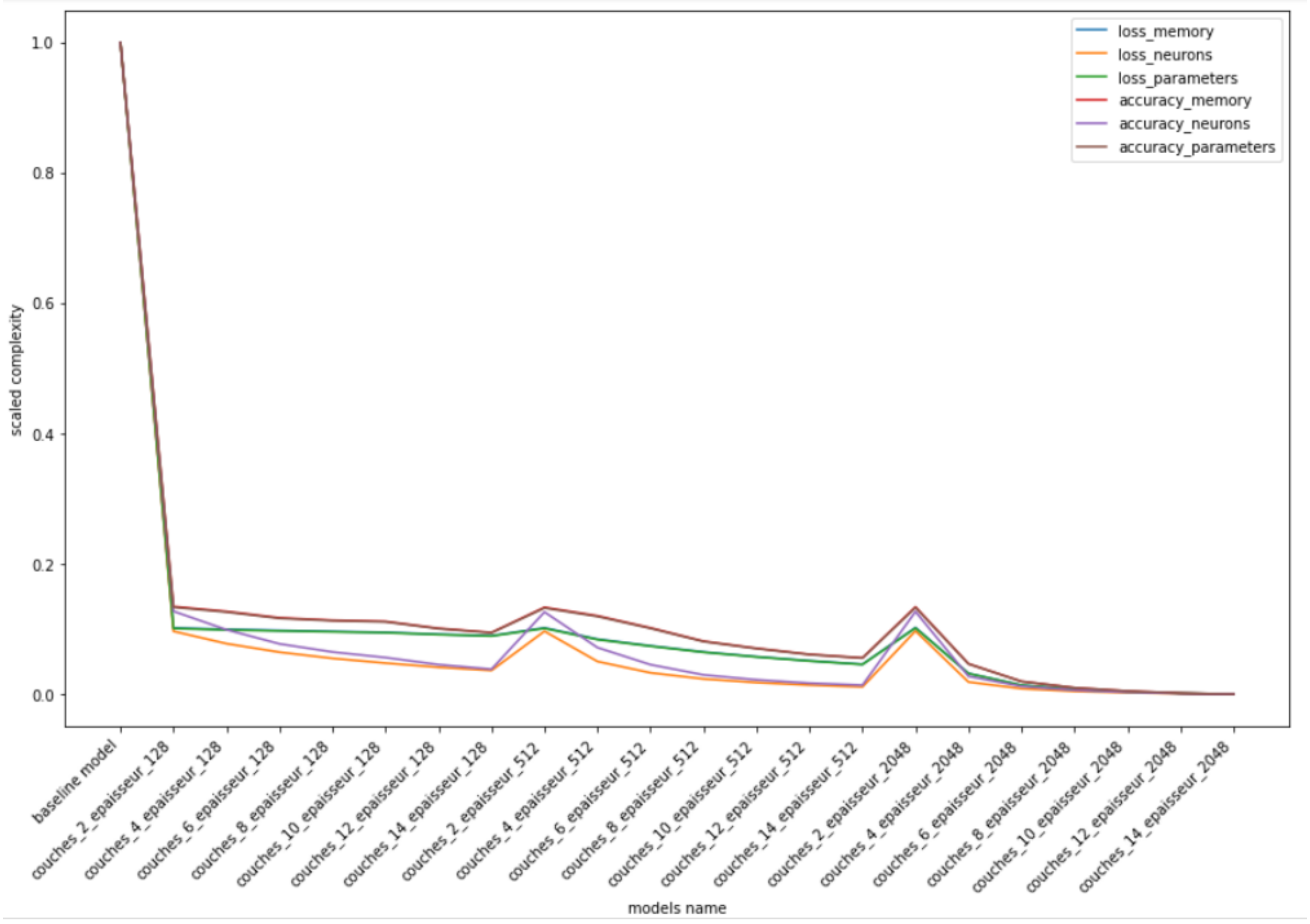


	loss_memory	loss_neurons	loss_parameters	accuracy_memory	accuracy_neurons	accuracy_parameters
baseline model	0.226926	0.002685	8.737382e-07	0.134127	0.001587	5.164335e-07
couches_2_epaisseur_2048	0.024498	0.000270	9.355148e-08	0.018786	0.000207	7.173959e-08
couches_2_epaisseur_512	0.024405	0.000269	9.319438e-08	0.018698	0.000206	7.140105e-08
couches_2_epaisseur_128	0.024378	0.000269	9.309241e-08	0.018859	0.000208	7.201657e-08
couches_4_epaisseur_128	0.023859	0.000218	9.115083e-08	0.017855	0.000163	6.821142e-08
couches_6_epaisseur_128	0.023502	0.000183	8.982535e-08	0.016551	0.000129	6.325612e-08
couches_8_epaisseur_128	0.023166	0.000158	8.860074e-08	0.016051	0.000109	6.139185e-08
couches_10_epaisseur_128	0.022837	0.000139	8.737955e-08	0.015813	0.000096	6.050312e-08
couches_12_epaisseur_128	0.022176	0.000121	8.488474e-08	0.014401	0.000079	5.512493e-08
couches_14_epaisseur_128	0.021659	0.000108	8.293526e-08	0.013545	0.000068	5.186489e-08
couches_4_epaisseur_512	0.020485	0.000145	7.824203e-08	0.016955	0.000120	6.475962e-08
couches_6_epaisseur_512	0.018171	0.000099	6.941863e-08	0.014496	0.000079	5.537796e-08
couches_8_epaisseur_512	0.016056	0.000074	6.135869e-08	0.011769	0.000054	4.497499e-08
couches_10_epaisseur_512	0.014414	0.000059	5.508878e-08	0.010318	0.000042	3.943221e-08
couches_12_epaisseur_512	0.013006	0.000048	4.971016e-08	0.009082	0.000034	3.471472e-08
couches_14_epaisseur_512	0.011787	0.000041	4.506063e-08	0.008359	0.000029	3.195672e-08
couches_4_epaisseur_2048	0.008632	0.000061	3.294516e-08	0.007156	0.000050	2.731361e-08
couches_6_epaisseur_2048	0.004562	0.000034	1.740881e-08	0.003550	0.000026	1.354774e-08
couches_8_epaisseur_2048	0.003038	0.000023	1.159451e-08	0.002248	0.000017	8.576526e-09
couches_10_epaisseur_2048	0.002246	0.000017	8.569004e-09	0.001573	0.000012	6.001381e-09
couches_12_epaisseur_2048	0.001760	0.000013	6.714257e-09	0.001142	0.000009	4.356979e-09
couches_14_epaisseur_2048	0.001445	0.000011	5.512029e-09	0.000939	0.000007	3.582589e-09

They were ranked in descending order, because the higher the accuracy or the inverse of the loss, the closer the model is to the expected results. And the higher the complexity, the closer we think we are to the Kolomogorov complexity.

We observe that all measurements give us the same order for the networks. This is a good indication of the agreement of the chosen estimators, as it shows that they discern the data along similar lines, which is what we wanted. Furthermore, each of our models far outperforms the baseline model. This proves that the baseline is perhaps the most efficient network, using all available metrics, because it has very few connections, very few parameters, tiny memory size, and pretty decent results. Thus, this model won the experiment by far for all complexity metrics.

In order to further analyze our metrics, a more interpretive version of our results is needed to gain more insight into the relevance of the measures. Therefore, we will standardize our results and plot them in the same graph. You can view the results below :



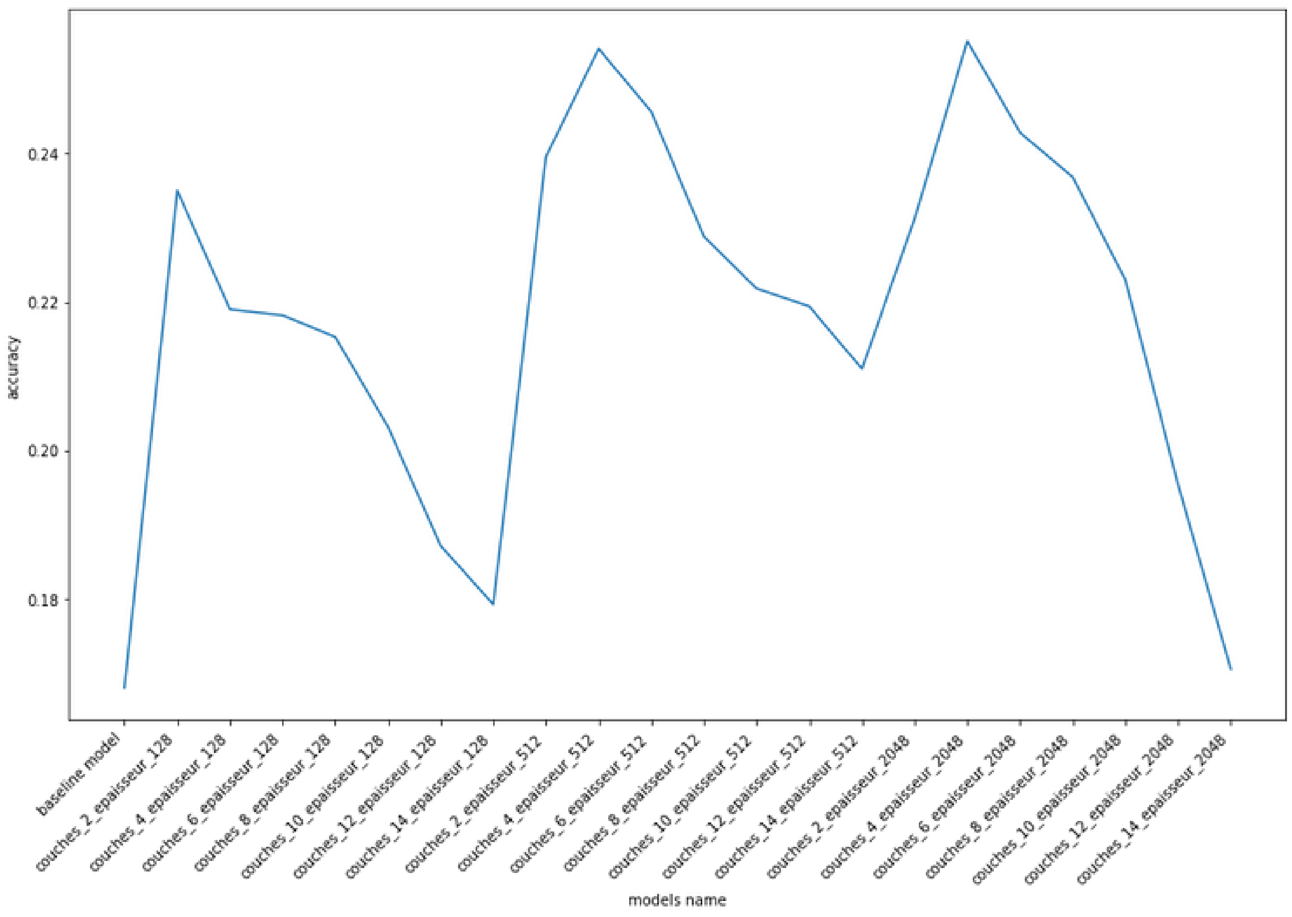
This graph allows for a better comparison of the metrics used. In this case, the best of all the metrics is the accuracy by neurons, because it highlights the differences by increasing distances better than the other metrics.

Discussion

As we have seen in these experiments, there is still a lot of work to be done to adapt K-complexity to neural networks. First, the adapted version we defined at the beginning only holds under certain assumptions. It remains to be seen whether K-complexity can be adapted to a subset of programs. Also, the task performed by the networks must be a classification task.

For the part where we looked for estimators of our quantities, the most relevant estimator of the program size was discarded. It is really annoying not to use the computational length when fitting the Kolmogorov complexity. Yet, approximations of this quantity have been found. On the other hand, the accuracy seems to be the best available estimator for the proximity of the expected results. This proves that we have achieved some improvement in the estimation of the K-complexity version of the neural network.

Regarding the results obtained, the model selected by our method has the least accuracy. As shown in the graph below :



This is a big enough problem to invalidate all of our theory. K-complexity first defines the programs that lead to the expected results, and then takes the shortest one. It now seems clear that our approximation does not work as expected and cannot be considered a K-complexity approximation for neural networks. Even if this article does not provide any meaningful estimation, it can be seen as a first draft in the path of getting closer to the desired K-complexity approximation for neural networks.

Also, significant work remains to be done to analyze the built-in functions and mapping subprocedures to allow the computation of program length in terms of bytes. Overall, although the ideal K-complexity cannot be calculated, this micro-study has shown that approximations can be found, even if the chosen one was not suitable. Ideally, a good approximation should rank programs according to the distance between themselves and the K-complexity. And thus this approximation will have an extremum, certainly infinite, corresponding to the ideal K-complexity, in which it would be possible to get closer.