June, 2023

Algorithmic Information
&
Artificial Intelligence

Micro-study

teaching.dessalles.fr/

Ce document contient les contributions des étudiants rédigées lors du cours "Information Algorithmique et IA" enseigné en mai-juin 2023.
Je suis très reconnaissant aux étudiants d'avoir été des participants actifs, et aussi pour la qualité de leur travail.

This document contains students' contributions written during the course "Algorithmic Information and AI" taught in May-June 2023.
I am very thankful to students for having be active participants, and also for the quality of their work.

Jean-Louis Dessalles

Content

June, 2023

IA225

Algorithmic Information & Artificial Intelligence

Micro-study

teaching.dessalles.fr/FCI

Name: Lais Isabelle ALVES DOS SANTOS

Identifying coincidences in Cross-verified database of notable people

Abstract

This micro-study aims to apply and develop the knowledge gained in the Algorithmic Information Theory classes in a practical way. More specifically, the main goal is to develop a proof of concept related to the generation of *unexpectedness* and *coincidence*.

Problem

How to tell a story? What makes a story interesting? What is so fascinating about the coincidences between Abraham Lincoln and John F. Kennedy? These are some of the main questions that gave rise to the desire of studying and carrying out this work. One can investigate whether some facts are very interesting and unexpected while others are not so much, and how Kolmogorov Complexity and Algorithmic Information are related to these events.

Method

In order to explore the phenomenon of coincidence and unexpectedness related to complexity and algorithmic information, the project uses a data set of notable people, *A cross-verified database of notable people* [1], which contains some interesting information about these people, such as the name; years of birth and death; along with the latitude and longitude of these places; domain and main area; the logarithm of the sum of 5 criteria (number of hits, number of Wikipedia editions, non-missing biographical information, length of pages and total number of external links); and many other data. For this project in particular, the most important features are the ones

mentioned above, since the goal is to create unexpected according to the place of birth and death of two people, along with their popularity.

The project repository [2] consists of two main files, a Jupyter notebook with the main computations, such as the data pre-processing and the loop to generate the unexpectedness, and a Python file with all functions used in the project. To compute the unexpectedness and coincidence for given an event, the fact that two famous people were born and died in close proximity, the *causal complexity* must be large and the *description complexity* or *Kolmogorov complexity* must be small. The difference between these two complexities and the drop in complexity generates unexpectedness, which makes the event surprising. The equations below summarize this explanation:

$$C_w(s) = C_w(p_b) + C_w(p_d) + C_w(a_b) + C_w(a_b)$$
$$C(s) = C(l_1 - l_2) + C(l_3 - l_4) + C(P_1) + C(P_2)$$
$$U(s) = C_w(s) - C(s)$$

where s is the event, p_b and p_d are total population of the place of birth and death, respectively, of the two considered people, a_b and a_d are the area of birth and death of these same places, when considering the causal complexity. As for the description complexity, $C(s)$, the difference between the two places of birth, l_1, l_2, and death, l_3, l_4, take into account the geodesic distance. The simplicity of person P_1 and person P_2 are calculated considering the popularity of famous people from the same area, who are also dead. The formula is as follows:

$$S(p) = \frac{log_2(p_f)}{log_2(p)}$$

where S is the simplicity, p_f is the famous person and p is the person in question. The generation complexity of each person is not included in the equations, because it is already in the context, so it is available for free.

In this sense, the people present in the preprocessed data set are iterated to find the unexpectedness among them.

Results

As for the results, after sorting people by popularity, some interesting coincidences were found. To filter the unexpectedness, a threshold of 70 was considered to include only the most surprising coincidences. Some interesting results:

- Marilyn Monroe and Frank Sinatra
- Both musicians
- Born in Los Angeles, California - United States and Hoboken, New Jersey - United States
- Both died in Los Angeles, California - United States

In this instance, the unexpected was due to the fact the proximity within the places of the death and the simplicity of both musicians, due to their popularity.

Discussion

In conclusion, the micro-study allowed to prove the fact that, when there is a complexity drop between generation complexity and description complexity, some unexpectedness and coincidence might be generated. Considering the implementation for the study, before sorting the people according to their popularity, the results were not very interesting, since there the people were not very known, even if they were born or died in the vicinity. Even after sorting, computing each pair in a large dataset takes a lot of time, and a more powerful GPU might be needed to get all the results.

If one wants to see some pertinent results considering only the most popular people, an alternative is to take only the first five or ten people in the data set and compare their unexpectedness results. In addition, another improvement to obtain more accurate results would be to compute the generation complexity for the village where the person was born or died instead of the complexity of the whole country.

Bibliography

[1] **A cross-verified database of notable people** `https://doi.org/10.1038/s41597-022-01369-4`
[2] **Github Micro-study** `https://github.com/liadsantos/ai225-mini-project-coincidences`

June, 2023

IA225

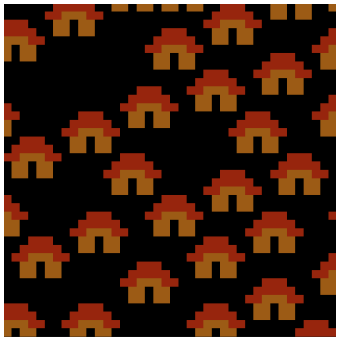
Algorithmic Information & Artificial Intelligence

Micro-study

teaching.dessalles.fr/FCI

Name: Louis CAUBET

Using Kolmogorov complexity in procedural generation



Abstract

Procedural generation is a computer method that create data using a generator and an input. This is most often used in games where there is a need for complexity in pattern, something that can be measured with kolmogorov complexity. In this paper we will analyze the utility of using kolmogorov complexity to classifies resulting output

Problem

Procedural generation generates random images with certain constraints. This sometime results in image that does not appear "random" for the human eye. This randomness can be computed using an approximation of kolmogorov complexity and images can then be ranked using this same complexity. We will analyze an algorithm, the WFC and try to apply constraints on the algorithm and resulting output using kolmogorov complexity.

Method

For this problem we will utilize an algorithm developed for procedural generation and inspired by quantum physics, the Wave Function Collapse (WFC). The WFC algorithm goes as follow:

1. Read the input bitmap and count NxN patterns.
2. Create an array with the dimensions of the output (called "wave" in the source). Each element of this array represents a state of an NxN region in the output. A state of an NxN region is a superposition of NxN patterns of the input with boolean coefficients (so a state of a pixel in the output is a superposition of input colors with real coefficients). False coefficient means that the corresponding pattern is forbidden, true coefficient means that the corresponding pattern is not yet forbidden.
3. Initialize the wave in the completely unobserved state, i.e. with all the boolean coefficients being true.
4. Repeat the following steps:
   (a) Observation:
      i. Find a wave element with the minimal nonzero entropy. If there is no such elements (if all elements have zero or undefined entropy) then break the cycle (4) and go to step (5).
      ii. Collapse this element into a definite state according to its coefficients and the distribution of NxN patterns in the input.
   (b) Propagation: propagate information gained on the previous observation step.
5. By now all the wave elements are either in a completely observed state (all the coefficients except one being zero) or in the contradictory state (all the coefficients being zero). In the first case return the output. In the second case finish the work without returning anything.

In this paper we will use patterns that are 2x2 pixels as to speed the algorithm and study on larger set of output. One thing to note is that the WFC algorithm is assured to create a valid image and error can happen because this problem is NP-hard and creating a somewhat fast solution requires us to accept a chance of error in output.

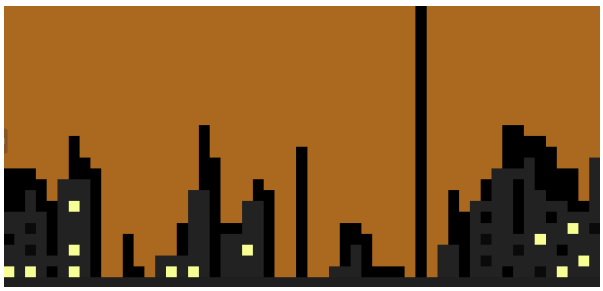
The output generated through the algorithm will be random but will follow two properties given:

1. The output should contain only those NxN patterns of pixels that are present in the input.
2. Distribution of NxN patterns in the input should be similar to the distribution of NxN patterns over a sufficiently large number of outputs. In other words, probability to meet a particular pattern in the output should be close to the density of such patterns in the input.

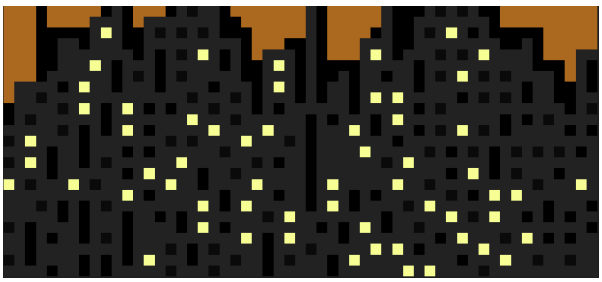
This means we can generate multiple image and compare their respective complexity to chose which one is better in the context of use.

For this endeavor we will need a way to compare each image complexity. To approximate kolmogorov complexity on an image we can compress it using a compressor. The compressor c needs to be normal:

$$c(x, x) = c(x)$$



Least complex image



Most complex image

Figure 1: Two images generated using the same input with different Kolmogorov complexity

the kolmogorov complexity of the concatenation of two identical image should be the kolmogorov complexity of the single image.

The LZMA compressor respect this property and we will use it for the rest of this study. We then define the NCD distance which represent the similarity of two image:

$$NCD(x,y) = \frac{C(x,y) - min(C(x), C(y)}{max(C(x), C(y))}$$

We can then compare two image and computes how different they are using their kolmogorov complexity but we still needs a way to compare their complexity and rank it. For this we will use ICR:

$$ICR(x,y) = \frac{NCD(x,y) \times C(y)}{C(x)}$$

We then define a relation

$$R(x,y) = xRy \iff ICR(y,x) \leq ICR(x,y)$$

This relation is a partial order that will allows us to rank image based on their perceived complexity. ICR has been tested on large dataset and is a good way to rank image based on perceived complexity.

We can use this ranking to determine which image to display to end user based on complexity. Most of the time we will display the most complex that appears as the most random for the human eye.

We then test another heuristic for pattern choosing in the algorithm using kolmogorov complexity ranking established earlier. We want to know if this heuristic can helps to change the complexity of the output image.

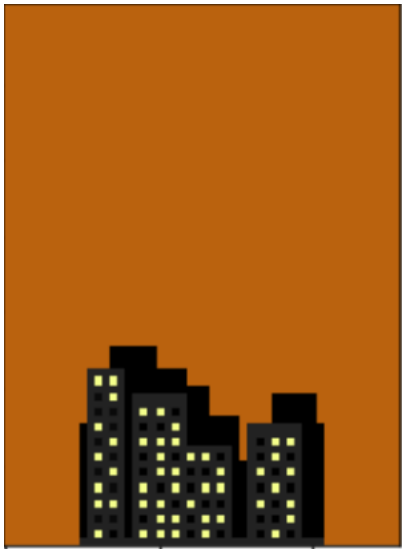
Results

Table 1: mean rank in ordered list

input	**Normal heuristic**	**Kolmogorov heuristic**
normal image	99.03	99.97
augmented image	102.63	96.37

200 images were produced using the same input. 100 with the basic heuristic and 100 with the kolmogorov heuristic described in the method section. they were then ranked based on complexity. Figure 1 is an example of the efficiency of such ranking. The experience was then repeated on an augmented image that used the same image but with a "blank" pattern added.

normal input



augmented input

Figure 2: different input

The lower the rank, the higher the complexity. The kolmogorov heuristic does not seems to be efficient in computing more complex output in either case but this could be because of the pattern size of 2x2 chosen for simplicity in computing. Choosing a 3x3 pattern size could result in a significative improvement if studied. The image might also need to be different to improve the impact of the augmented image. The augmented image was tested to prove that the kolmogorov heuristic could improve image if there was an added "blank" pattern that could reduce the frequency of more complex pattern.

Discussion

The ranking method of complexity works well with different type of input and a human can clearly see the difference between the first and last image of each ranking. The kolmogorov heuristic does not work as intended on augmented data and this might be because of choice discussed in the result section but this might also not work as an heuristic. The kolmogorov complexity of an image appears important in procedural generation where the perception of complexity in an image is crucial for the end user. A complex image appears random for a player or a painting enthusiast and researcher are becoming more and more interested in kolmogorov complexity applied to procedural generation.

Bibliography

Why Oatmeal is Cheap: Kolmogorov Complexity and Procedural Generation, Younès Rabii, Michael Cook `https://arxiv.org/abs/2305.02131`

May, 2023

IA225

Algorithmic Information & Artificial Intelligence

Micro-study

teaching.dessalles.fr/FCI

Name: Fadi KHATTAR

Applications and Limits of Zipf's Law

Abstract

This report tackles the applications of Zipf's Law on different corpuses and datasets. Its purpose is testing the validity of Zipf's Law and using this Law to make predictions on word ranks and frequencies. We will be comparing complexities and results to see if we can draw conclusions.

Problem

The distribution of word frequencies in natural language has been a topic of great interest in algorithmic and AI studies. One particular law that has gained my attention is Zipf's Law, which describes a consistent relationship between the rank of a word and its frequency in a given corpus. While Zipf's Law has been observed to hold in various linguistic datasets, I am particularly interested in exploring the extent to which this law can be applied and its predictive power across different corpuses and datasets. It remains a captivating area of investigation to determine the reliability and generalizability of Zipf's Law in various contexts.

This research paper aims to address the problem of determining the validity of Zipf's Law and exploring its potential applications in predicting word ranks and frequencies. Understanding the underlying patterns and complexities of word distributions can have significant implications in various fields, such as natural language processing, information retrieval, recommender systems and text mining. If Zipf's Law is found to be consistently valid across different corpuses, it could provide a valuable tool for predicting and analyzing word frequencies.

Method

To address the problem I presented above, I used the code Zipf.py provided in the labs to test Zipf's Law on random corpuses (books in my case) and I created a Python program which has the purpose of analyzing texts and corpuses. The main idea behind the program that I created is to tokenize the given corpus after cleaning the text and create a dictionary that stores words with their respective frequencies. I then compare the complexities of some words across different corpuses and the Web. Finally, using Zipf's Law, I aim to predict word frequencies and ranks and compare them to their actual rank or frequency in each corpus.

For the purpose of this research, I used three different books downloaded directly from https://www.gutenberg.org/. The three books are: The book of The Cheese, by T. W. Reid, Escape from east Tennessee to the federal lines, by R. A. Ragan, and The human species, by Armand De Quatrefages. I also used the Google Search Engine API to retrieve word frequencies from the web to later compare the complexities.

The first step in my Python Program involves creating the function get_word_freq that returns the dictionary of the words with their respective frequencies. To achieve that, I first created a function clean_and_tokenize that cleans the text from all symbols and unnecessary characters and splits the text on spaces. Therefore, this function will output a list containing all the words of the corpus. I then use this list as an argument for the function get_word_freq that iterates over the list while counting the occurrences of each word and storing it in a dictionary. These two functions also have an optional argument remove_stopwords which can be set to True so we can see the most relevant words for example after removing stopwords. This is useful to capture context since in most english corpuses for example the most frequent words are "the", "of" and "and". I also created the function get_web_frequencies which uses the Google API to retrieve word frequencies from the web. I used my own generated API key for the purpose of this project.

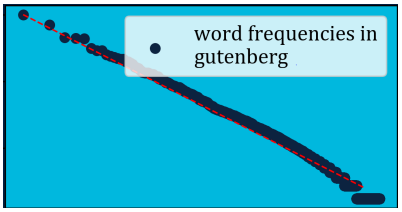
The second step is to calculate the complexities of the words using the formula $C=log2(1/f)$. I created the function convert_frequencies_to_complexities to do this task.

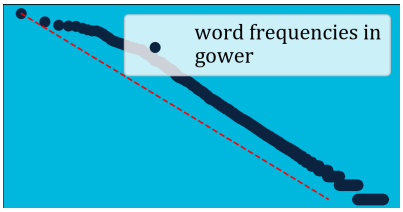
The last step in my method is to predict the word ranks and frequencies using Zipf's Law. The formula of the Law is: $r(w)=k/f(w)$ where w refers to a certain word and $r(w)$ refers to its rank and $f(w)$ refers to its frequency. Given this Law, if we have the word frequencies for example we will be able to predict their rank in the corpus by just applying the formula. The same applies if we want to predict the frequency of a word. Let's say we know that a word is the second most frequent word in a certain corpus, we can therefore calculate its frequency by only applying Zipf's Law without having to count its occurrences in the corpus. In this case, having the value of k and for $r(w)=2$, we get the value of $f(w)$.

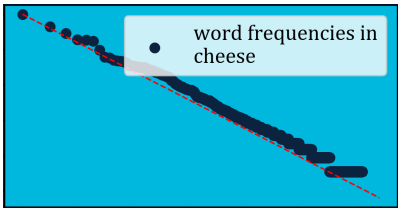
Therefore, we have to find the value of k for each corpus. My method consists of taking the most frequent word in a certain corpus which has the rank of 1 and using its rank and frequency to get an estimate of k for this particular corpus. Then, I use this value of k and apply it on all the different words of the corpus.

Results

Using the program Zipf.py provided in the Lab, I tested Zipf's Law on the three different books that I mentioned above. I got the following results:





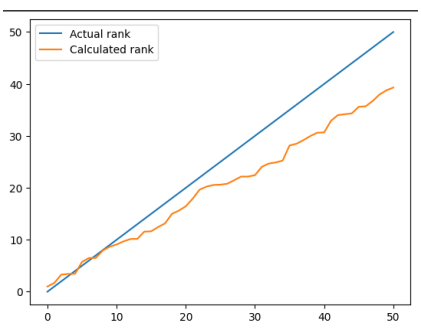


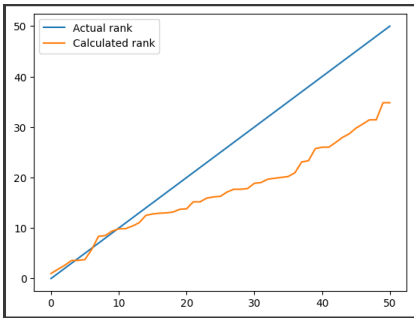
The figures show the constant relation between word frequencies and word ranks on a logarithmic scale.

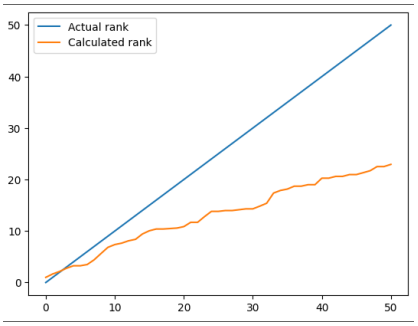
Using the program that I created, I tokenized and extracted the dictionary of word frequencies from each corpus. I then calculated the complexities for the most 5 frequent words in each corpus and compared the complexities of these words with their complexities after retrieving their frequencies from the web. I found that the most frequent words had lesser values in terms of complexity than the most frequent ones which was also consistent from the results from the web. However, the complexities of the words on the web are always relatively higher than those in a smaller corpus.

I then used my program to predict word ranks and word frequencies of the three different corpuses and compare them with their actual values. I got the following results:

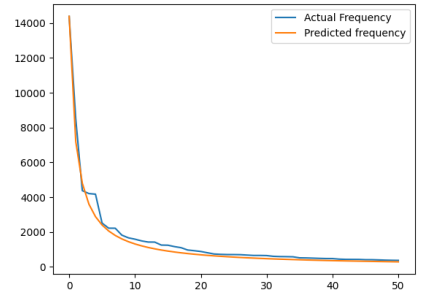
- Predicting word ranks for the most 50 frequent words:

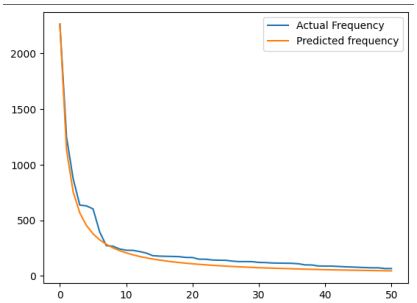


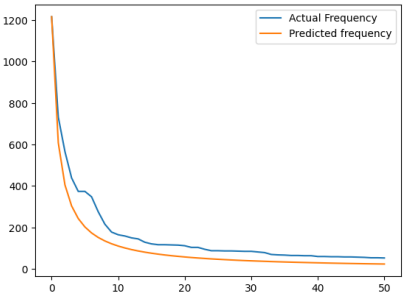




- Predicting word frequencies for the most 50 frequent words:







Discussion

As expected, after running Zipf.py on the three different books, I found that Zipf's Law was satisfied. This proves that Zipf's Law remains consistent on normal language corpuses.

Regarding word complexities, it is expected that the complexity for more frequent words would be relatively small due to the inverse relation between complexity and frequency.

However, on larger corpuses such as the Web and Google, the complexities of words can be higher because of the larger nature of the corpus.

In terms of predicting word ranks and frequencies, I got consistent results on each corpus which implies the importance of Zipf's Law. Using Zipf's Law, we can easily capture the context of a document or corpus without having to do any machine learning. By simply knowing the relation between word ranks and frequencies, we can directly know if a word is frequent in a certain corpus which can be useful in applications such as recommender systems. An important thing to note is that when estimating the value of k for each corpus, I only relied on the most frequent word in the corpus. This can be problematic especially considering the fact that after examining Zipf's Law on the three different books, we can see that the Law is most consistent with the most frequent and less frequent words. To get a better estimate of k and ultimately better predictions on a certain corpus, we should try different values of k by using words in the middle of the corpus which most probably yields a better representation of the corpus.

Bibliography

[1] Measuring Information through compression
https://aicourse.r2.enst.fr/FCI/Chapitre2.html

[2] Project Gutenberg https://www.gutenberg.org/

June, 2023

IA225

Algorithmic Information & Artificial Intelligence

Micro-study

teaching.dessalles.fr/FCI

Name: Thalis ROCHA PESTANA

NGD-BASED WORD PREDICTION ALGORITHM FOR CONTEXTUALLY RELEVANT SUGGESTIONS

Abstract

In this research, we leverage concepts of Normalized Information Distance (NID), Normalized Compression Distance (NCD), and Normalized Google Distance (NGD) based on Kolmogorov complexity, to build a word predictor using the GPT-2 language model. The goal is to predict suitable words to complete given incomplete phrases. Challenges arise in dealing with the computation of NGD for vast vocabularies, restrictions in utilizing Google API, and handling high-frequency closed-class words. The model generally aligns well with expected predictions, however, it struggles with high-frequency, closed-class words. Also, the response time of the model is not suitable for real-time applications. Despite these limitations, our study reveals the impressive capacity for contextual understanding and suggests future research for better model evaluation.

Problem

Researchers like Bennett, Vitanyi, Cilibrasi, and others have made important contributions to using Kolmogorov complexity for classification and measuring the shared information between objects. Their work has yielded promising results in different fields, including natural language.

The idea behind their approach is to compare pairs of objects and determine how much information they have in common. This is done by assigning a distance value that reflects the similarity between binary representations of the objects. If two objects share a lot of common information, their distance is small, indicating they are close. Conversely, if two objects have less shared information, their distance is larger, suggesting they are more independent [2].

For instance, when comparing two identical words, their distance is zero because they have all the information in common. On the other hand, two completely unrelated words would have a distance close to 1 in a normalized scale of distances between 0 and 1. The researchers' goal is to measure the similarity or dissimilarity between objects by quantifying the amount of shared information. This provides a way to classify objects based on their common characteristics, regardless of the specific domain they belong to.

In this context, the researchers introduced three metrics: NID (Normalized Information Distance), NCD (Normalized Compression Distance), and NGD (Normalized Google Distance). These approaches are aligned with Kolmogorov's concept of defining a numerical measure of information content of words, i.e. a measure of their randomness.

The objective of this research is to explore the effectiveness of utilizing specific distance measures in the prediction of suitable words to complete given incomplete phrases. To achieve this, an interface has been developed in Python. This interface accepts an incomplete sentence as input and feeds it into a pre-trained language model, GPT-2. The model is then asked to produce a certain number of word predictions, as specified by the user, to complete the sentence. Subsequent calculations are performed using Normalized Google Distance

(NGD) and Normalized Compression Distance (NCD) measures to determine the "distance" between the input phrase and each of the generated predictions. Following these calculations, the system presents the top 5 predictions.

Method

This study is based on the notions of Normalized Information Distance (NID) which measures the normalized difference in information content between two objects. NID is based on conditional complexity K(x|y) and measures the amount of information contained in x that is not present in y.

Bennet and al. defined the information distance between two words x and y as the size of the shortest program which maps x to y and y to x. An alternative definition can be given as follows:

$$ID'(x, y) = max\{K(x|y),\ K(y|x)\} \quad (1)$$

The equation 1 states that the shortest program which computes x from y takes into account all similarities between x and y. By using NID as a distance measure, it is possible to define a metric space where the distance between objects is determined by their information content. Therefore, NID satisfies the metric axioms:

$(M1)\ for\ any\ x\ \in\ X,\ d(x,x)\ =\ 0$ $(identity)$
$(M2)\ for\ any\ x,\ y\ \in X,\ if\ x \neq y,\ then\ d(x,y) \neq 0$
$(M3)\ for\ any\ x,\ y\ \in X,\ d(x,y)\ = d(y,x)$ $(symmetry)$
$(M4)\ for\ any\ x,\ y,\ z\ \in X,\ d(x,y)\ \leq d(x,z)\ +\ d(z,y)$ $(triangle\ inequality)$

From the axioms it is possible to develop the Normalized Information Distance as formulated in the equation 2.

$$NID(x, y) = \frac{K(x,y) - min[K(x), K(y)]}{max[K(x),K(y)]} \quad (2)$$

The NID remains an abstract notion, since Kolmogorov complexity is not computable [1]. Consequently, NID was proposed as an ideal distance which can be approximated by replacing the Kolmogorov function K by computable compression algorithms. Vitanyi proposed two other metrics : Normalized Compression Distance (NCD) and Normalized Google Distance (NGD).

NCD is obtained by replacing K by a compressor such as “zip” as shown in equation 3.

$$NCD(x, y) = \frac{Z(x,y) - min[Z(x), Z(y)]}{max[Z(x), Z(y)]} \quad (3)$$

NGD is obtained by replacing K by log2(1/f(x)), where f(x) is the observed frequency of x on the Web, or by log2(N/g(x)), where N is the corresponding total number of indexed pages and g(x) denotes the number of pages containing x. The equation 4 shows the NGD.

$$NGD(x,y) = \frac{max[log(f(x)), log(f(y))] - log(f(x,y))}{log\ N - min[log(f(x)), log(f(y))]} \quad (4)$$

Pre-processing:

The initial idea of applying NGD directly to make predictions had limitations due to the computational feasibility of calculating NGD for each word in a vocabulary with several words. To overcome this challenge, the GPT-2 model was utilized to make predictions and the NGD was applied to rank the predictions, returning the "best" 5 words to complete a phrase given by the user.

The GPT-2 model is a powerful language model that has been trained on a large corpus of text data from the internet. It has the ability to generate coherent and contextually relevant text based on the patterns it has learned from the training data. Despite being a powerful tool, the GPT-2 was doing identical predictions for simple sentences with contextual information. For instance, the phrase: "I'm tired, I need…" received the word "help" several times. As the main objective was to provide a word predictor with different word suggestions, a pre-treatment of the information needed to be done before the computation of the distances in order to avoid multiple identical predictions.

An additional challenge we encountered involved conducting searches on Google. Ideally, using Google's API is the most appropriate method to extract search results from Google. However, this API has very limited quotas. To avoid problems with these limits, two other libraries were experimented: BeautifulSoup and Selenium. However, Google's built-in measures to prevent automated scraping resulted in inconsistent search result numbers when executing the script. Therefore, the decision was made to utilize the Google API, even with its limitations.

Another issue arose in connection to how the input phrase was queried on the search engine. Without enclosing the search term in quotation marks, the engine returned results containing all words from the phrase, but not in the specified order. By incorporating quotation marks, phrases like "I'm tired" only returned results with all words in the exact stated order.

Method evaluation

After the pre-processing, two approaches were applied. The first approach consists in calculating the NGD between the entire input phrase and each word predicted (without any tokenization) and the second approach consists in calculating the mean NCD in an analogous way.

To test the different methods, some phrases were used as inputs for the system, and the predictions it made were collected for review. These phrases covered a wide variety of common situations and topics to check how flexible the system is. The phrases used and the expected predictions were:

“I am tired, I need…” (sleep)
“Last week I went to fast foods everyday, I love…” (eat/hamburguer)
“I play football every week. Football is my…” (passion/hobby/favorite)
“There is a movie showing, want to go with…” (me?/ us?)
“Spiderman is a…” (superhero/ hero)

Results

The table 1 shows for each phrase tested the predictions from GPT-2, the rank of words using NGD and the rank using NCD. For a posterior comparison, the word predictions from an iPhone 11 keyboard were noted. In the third topic of the References section is provided a link with some images from the prompt interface with the words predicted by GPT-2 and the suggestions based in NGD and NCD.

Table 1 - Input phrases, predictions and suggestions.

	GPT-2 Predictions	**NGD ranking**	**NCD ranking**	**iPhone keyboard**
“I’m tired, I need…”	['another', 'rest', 'to', 'more', 'a', 'you', 'your', 'some', 'sleep', 'help']	1 - 'rest'; 2 - 'sleep'; 3 - 'another'; 4 - 'some'; 5 - 'more'	1 - 'another'; 2 - 'rest'; 3 - 'to'; 4 - 'more'; 5 - 'a'	1 - to; 2 - a; 3 - some
“Last week I went to fast foods everyday, I love…”	['it', 'my', 'to', 'their', 'burgers', 'fast', 'being', 'food', 'the', 'them']	1 - 'burgers'; 2 - 'fast'; 3 - 'food'; 4 - 'being'; 5 - 'them'	1 - 'fast'; 2 - 'food'; 3 - 'it'; 4 - 'my'; 5 - 'to'	1 - it; 2 - the; 3 - ❤️
“I play football every week. Football is my…”	['life', 'team', 'highlight', 'hobby', 'friends', 'body', 'number', 'passion', 'go', 'big']	1 - 'hobby'; 2 - 'passion'; 3 - 'highlight'; 4 - 'friends'; 5 - 'team'	1 - 'life'; 2 - 'team'; 3 - 'highlight'; 4 - 'hobby'; 5 - 'friends'	1 - first; 2 - favorite; 3 - best
“There is a movie showing, want to go with…”	['with,', 'TR', 'me', 'it', 'them', 'with?"', 'a', 'the', 'that', 'some']	1 - 'with,'; 2 - 'them'; 3 - 'some'; 4 - 'that'; 5 - 'it'	1 - 'with,'; 2 - 'with?"'; 3 - 'that'; 4 - 'TR'; 5 - 'me'	1 - the; 2 - us; 3 - a

“Spiderman is a…”	['big', 'member', 'superhero', 'real', 'master', 'monster', 'good', 'super', 'fantastic', 'former']	1 - 'superhero'; 2 - 'fantastic'; 3 - 'former'; 4 - 'monster'; 5 - 'master'	1 - 'big'; 2 - 'member'; 3 - 'superhero'; 4 - 'real'; 5 - 'master'	1 - cute; 2 - nice; 3 - cool

Source: Produced by the author.

Discussion

We can observe that, with the exception of the fourth phrase, the NGD ranking aligns with our expected predictions. NCD also performs well but omits some significant words, such as 'passion' in the third phrase, 'burgers' in the second, and 'sleep' in the first. The iPhone keyboard prediction is less successful in these instances.

However, when examining the fourth input phrase, NCD and the iPhone keyboard provided superior predictions. NCD included "us" among its top predictions, while the iPhone keyboard suggested "me". In this case, the words predicted by the GPT-2 model are largely functional or "closed-class" words. This class includes pronouns, determiners, prepositions, and conjunctions, which are infrequently updated with new words—hence, they are "closed." These words typically serve grammatical or structural functions and are common in a variety of contexts, thereby reducing their distinctiveness. This high frequency and diversity block the specificity of the NGD measure, making it challenging to effectively rank these words.

Another issue lies not with the NGD function, but with the predictions generated by the GPT-2 model. The GPT-2 model occasionally produced predictions that did not align with the expected outcomes. For instance, the phrase "Spiderman is a…" might result in predictions such as: powerful; unique; new; small; serious; good; Super; mutant; prominent; p—without any mention of the word "hero". However, it does demonstrate the model's capability to understand the context of the words since GPT-2 consistently returns "hero" or "superhero" for the input phrase: "Dr. Octopus is a villain. Spiderman is a…".

It's important to note that we need more sophisticated methods for evaluating the model's performance. Testing a larger number of input phrases with a more extensive vocabulary of expected predictions of similar meanings would allow the computation of more comprehensive metrics, such as precision, recall, or F1-score. But as the Google API, the only method found to provide consistent search results, imposes a daily quota, such extensive testing is not feasible.

Another significant point is the unsuitability of this model for real-time applications. In real-world scenarios where people need to compose messages quickly, the model's average response time of 3 minutes is impractical for regular conversation in a messaging application.

In conclusion, while the Normalized Google Distance (NGD) ranking generally aligns well with the expected predictions, it falters when dealing with high-frequency, closed-class words. In addition, the model is limited by the GPT-2 predictions that do not always align with the ground truth. More sophisticated evaluation methods are required for better model assessment and the response time renders it unsuitable for real-time applications such as instant messaging. Despite its shortcomings, the model still demonstrates an impressive capacity for contextual understanding.

Bibliography

[1] - Ferbus-Zanda, Marie, and Serge Grigorieff. **"Kolmogorov Complexity in perspective."** arXiv preprint arXiv:0801.0354 (2008).

[2] - **"FCI - Chapitre 2: Codage et Compression."** 2023. AI225. Télécom Paris. Accessed June 13, 2023. https://aicourse.r2.enst.fr/FCI/Chapitre2.html.

[3] - **"Results Folder"** Google Drive, accessed June 13, 2023, https://drive.google.com/drive/folders/1JEq2gva_yyAUsWOjuag-i692rwmFn9g9?usp=sharing.

June, 2023

IA225

Algorithmic Information & Artificial Intelligence

Micro-study

teaching.dessalles.fr/FCI

Name: Guillermo Daniel TOYOS MARFURT

Kolmogorov Complexity in CNN's kernels

Abstract

This experiment aimed to investigate if Convolutional Neural Networks, after being trained, are compressible. To this we applied compression and pruning techniques and verified that, without a significant loss in accuracy, a CNN can be significantly compressed and their kernel matrices have visually appealing patterns.

Problem

"Machine Learning is compression". The objective of this work is to put in practice this phrase by studying the compressibility of a Convolutional Neural Network. As seen in the course, we can obtain a higher bound on Kolmogorov complexity by compressing the model parameters [2]. In the context of neural networks, complexity can provide insights into the minimum description length required to represent the model's parameters. This experiment aims to explore the relationship between model compression, achieved through statistical learning, weight pruning, and the resulting bounds on the Kolmogorov complexity. Particularly, we focus on the kernel weights of a 2-layer convolutional neural network. We would like to know if by adjusting a model to the data, the kernel weights become compressible, which means they can have a simpler representation.

This can be considered useful, as deep learning models, particularly CNNs, often exhibit high memory requirements due to their large number of parameters. This can limit their deployment on resource constrained devices, such as mobile phones and embedded systems. representation of the network.

Method

1. Dataset Selection: A suitable dataset was chosen to train a CNN model. In this experiment, the MNIST dataset consisting of 60,000 small square 28×28 pixel gray-scale images of handwritten single digits from 0 to 9[3], was used. We choose this dataset as it can be considered one of the simplest machine learning vision tasks.

2. Model: The architecture used was a standard convolutional network with multiple layers, such as convolutional, pooling, and fully connected layers. The convolutional layers consisted of convolutions of size 5x5. We used PyTorch [4] framework for implementing the model and the training algorithm.

3. Compression: Once the CNN model was trained, the compression was divided in two phases: pruning and lossless compression. First, the pruning technique was applied to compress the model. The pruning process involved identifying and removing unimportant weights according to threshold hyper-parameter value. Then, the pruned weights were losslessly compressed using Zlib [1].

4. Compressed Model Evaluation: The accuracy of the compressed model was evaluated on a validation set from the MNIST dataset. The evaluation metrics used included accuracy, and negative log likelihood loss. These metrics were compared with those of the original, uncompressed model to assess the impact of compression on performance. In addition, the kernel weights are also evaluated in the search of any patterns and to assess the state of the convolutional layers

5. Comparison with Original Model: The results obtained from the pruned model were compared with the original model to measure the compression achieved and determine any potential loss in accuracy.

Results

The experiment yielded results that showcased the successful compression of the CNN model using the pruning technique and its relationship to Kolmogorov complexity:

1. Original Model Accuracy: 97.6%

2. Compressed Model Accuracy: 97.1%

3. Compression Ratio for first layer: 35%

4. Compression Ratio for second layer: 220%

The values in bytes for the convolutional layers before training and pruning are the following

1. Layer 1 compressed size: 989 bytes.

2. Layer 1 uncompressed size: 1011 bytes.

3. Layer 2 Compressed size: 18578 bytes.

4. Layer 2 Uncompressed size: 20011 bytes.

After training the model and applying weight pruning:

1. Layer 1 compressed size: 749 bytes.

2. Layer 2 compressed size: 9095 byes.

The second layer reduced its size by more than two compared to the original model, indicating a significant reduction in the number of parameters. Despite this reduction, the pruned model's accuracy decreased only slightly by 0.5%.

It's important to note that if we try to compress an untrained model - that is convolutional kernels that are randomly initialised from a normal distribution - we obtain virtually zero compression. Which is what's expected from random data.

These results demonstrate the effectiveness of the pruning technique in compressing CNN models while maintaining accuracy. The compressed models enable a more succinct representation of the network's parameters, highlighting their potential for reducing complexity in deep learning models.

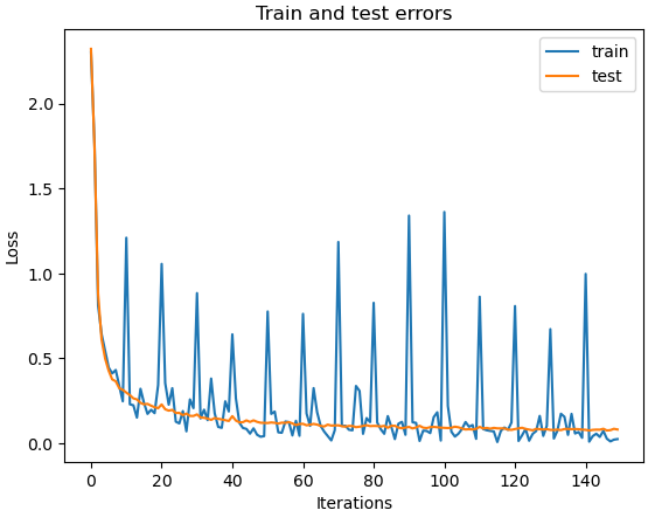


Figure 1: Evolution of the negative log likelihood on the training (per batch) and test datasets

Image 1 shows the evolution of the errors on both datasets. We observe that the model succeeds in classifying the numbers as the loss converges to a small value.

We observe the results of the pruning technique on image 2. The percentage of zero entries is 40%.

The first layer in the convolutional kernels (figure 3) can be less compressed and this is verified when comparing with the kernels of the second layer. However, there is still a locality principle which enables the reported compression level.

Image 5 is an interesting kernel. As we can observe, most of its entries are zero. In addition, values in the left are negative and on the right are positive. All of this indicates that a simple representation of the convolutional kernel exists and indeed it is verified in the compression level of the second layer as observed in the results section .

In both figure 4 and 3 we can visually spot many clear patterns in the kernel configurations. For example we observe that proximate entries have similar values. This fact is exploited by the compression algorithms of Zlib [1].

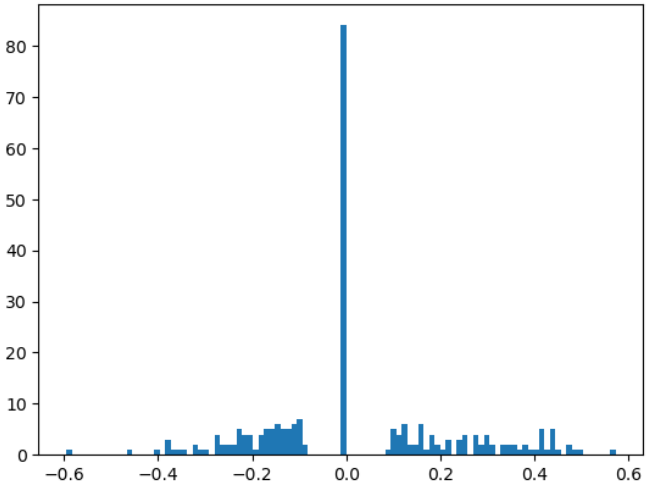


Figure 2: Histogram of values in the convolutional kernels on the compressed model.

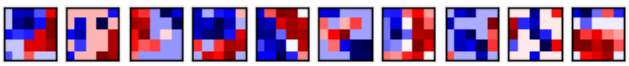


Figure 3: First layer convolutional kernels. Redder pixels means more positive values, white zero entries and bluer pixels more negative values.

Discussion

The experiment successfully explored the compression of CNN kernel weights using a pruning technique and its relationship to Kolmogorov complexity. By compressing the model, we found a higher bound on complexity. And by showing that the model after training is compressible, we indicated a more concise representation of the network. Moreover, the results demonstrated that pruning can significantly reduce the model's complexity while incurring only a marginal loss in accuracy. This has implications for understanding the minimum description length required to represent neural network parameters. Future work could involve further analysis of Kolmogorov complexity in relation to different pruning techniques and evaluating the resulting model in more complex datasets such as ImageNet.

Indeed, knowledge is compression. Through gradient descent, the machine learning algorithm has found a structure that succeeds in extracting relevant features from the handwritten images by using convolutional kernels. We expect this kernels to be rather simple, to represent a simple feature, not to appear as random data. As we have observed in the coloured representation of the kernels. We intuitively observe certain patters, be it sparse or with a clustered distribution: very positive values and very negative values are each grouped together in a certain way.

Its important to note that what we show here is that there are CNN models that successfully solves a task, with an arbitrary degree of accuracy, which are compressible. Thus, the learning algorithm, starting with a random incompressible model, perfroms gradient descent and reaches a model that, with the aid of weight pruning, is compressible and recognises numbers.

References

[1] Mark Adler. Zlib: A massively spiffy yet delicately unobtrusive compression library.

Figure 4: Second layer convolutional kernels.

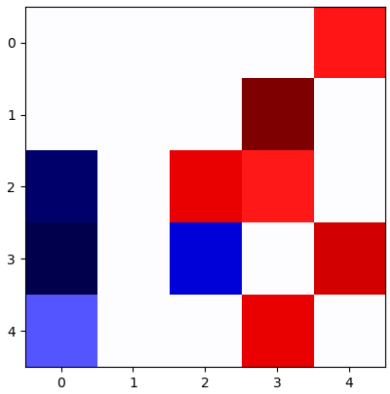


Figure 5: Selected kernel of the second layer.

[2] Jean-Louis Desalles. Algorithmic information and a.i.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.