Chromatic Analysis of Numerical Programs

David Defour, Franck Vedrine
LAMPS, Univ. of Perpignan Via Domitia, France
Université Paris-Saclay, CEA, List, Saclay, France

***Abstract*—This paper introduces the concept of chromatic numbers, which allows to tint a scalar or a set of scalars to estimate the relations between input and output variables under additive property. This consists in proposing a decomposition of a resulting value as a sum of tinted values. We illustrate how this concept can be used on a deep neural networks example by tinting group of value at once allowing to track a large number of values together.**

***Index Terms*—IEEE-754, sensitivity analysis, automatic differentiation, rounding error analysis**

I. Introduction

A numerical program is a sequence of instructions executed by a computer that process a set of values as input and produces a numerical output. The sequence of instructions can itself be broken down into floating-point operations and numerical values on which we can perform error analysis to track the propagation of errors and approximations. However, such solutions do not help to understand and estimate the contribution of input values, parameters and constants in a given result.

This phenomenon is accentuated by the recent trend toward smaller representation formats to reduce memory and data transfer costs. This is especially true for applications such as neural networks. This trend reflects the fact that we are more interested in the few leading digits of the final results and less interested in error propagation, event thought both are related.

We propose to estimate the weight of input values, parameters and constants in the output produced by a program, and interpret them graphically using colors, which has inspired the name of the method: *Chromatic Analysis* (CA). In other words, it is a way when running a program to say "these output values have been influenced by this set of input values by this amount".

The rest of this article is organized as follows. Section II introduces the concept of chromatic numbers. Section III illustrates how chromatic numbers can be used to produce numerical and graphical interpretations.

II. Chromatic Analysis

A *chromatic number* (or CN in the sequel of this article) corresponds to a set composed of: a floating-point number x, a constant k_x and a vector V_x of n floating-point numbers representing the weight of the n chromaticity (or tint) within x, as defined in Definition 2.1.

Definition 2.1 (Additive property): A chromatic number corresponds to a triplet $\langle x, k_x, V_x \rangle$ such that x is a scalar and V_x is a vector of scalar value of size n representing the decomposition of x as the sum of tracked values such that $x = \frac{1}{k_x}\sum_{i=0}^{n} V_x[i]$.

Thanks to this definition, each floating-point number is replaced by the set $\langle x, k_x, V_x \rangle$ corresponding to a chromatic number. This requires to substitute regular arithmetic on floating-point numbers by a new arithmetic on triplet. This can be done at the language level using operator overloading, similar to the dual number concept (section IV-B). Regular arithmetic is used on the first component x, and an arithmetic that preserves the property $x = \frac{1}{k_x}\sum_{i=0}^{n} V_x[i]$ on the third term:

- Addition:
  $\langle x, k_x, V_x \rangle + \langle y, k_y, V_y \rangle = \langle x + y, 1, V_x/k_x + V_y/k_y \rangle$
- Subtraction:
  $\langle x, K_x, V_x \rangle - \langle y, K_y, V_y \rangle = \langle x - y, 1, V_x/k_x - V_y/k_y \rangle$
- Multiplication:
  $\langle x, k_x, V_x \rangle \, . \, \langle y, k_y, V_y \rangle = \langle x.y, k_x + k_y, y.V_x + x.V_y \rangle$
- Division:
  $\langle x, k_x, V_x \rangle \, / \, \langle y, k_y, V_y \rangle = \langle x/y, k_x + k_y, x/V_y + V_x/y \rangle$
  $= \langle x/y, k_x + k_y, (x/y^2).V_y + V_x/y \rangle$ if $(y \neq 0)$
- Sqrt:
  $sqrt(\langle x, k_x, V_x \rangle) = \langle sqrt(x), 1, V_x/(k_x.sqrt(x)) \rangle$
- tanh:
  $tanh(\langle x, k_x, V_x \rangle) = \langle tanh(x), 1, V_x/(k_x.Q) \rangle$ with
  $Q = 1 + \prod_{i=0}^{n} tanh(V_x[i])$

and in general for the function f,

$$f(\langle x, k_x, V_x \rangle, \langle y, k_y, V_y \rangle) = \langle f(x+y), k_x + k_y, f(x, V_y) + f(V_x, y) \rangle$$

This arithmetic gives a component-wise decomposition of a numerical value with additive property. At initialization, a tint corresponding to an index $j, 0 < j \leq n$ is set for each scalar x to be followed, so that the given scalar is promoted to a CN $\langle x, k_x = 1, V_x \rangle$, with $V_x[i] = 0$ for $i \neq j, 0 \leq i \leq n$ and $V_x[j] = x$. The goal of the parameter k_x is to ponderate the relative weight of tracked values during multiplication and division (see section III for an example).

One can notice that a CN consisting of associating a tint with a scalar value can be extended by associating the same tint to a set of scalar without violating the 2.1 property. This helps track multiple values at the same time, while reducing the curse of dimensionality of the problem by allowing aggregation (or tinting with the same color) of sets of numbers. Indeed, to ease the interpretation, the aggregation should make sense in the additive sense.

In order to compare the weight of tracked values with others which are non-tracked, we require to dedicate a specific tint for

This research was supported by the InterFLOP (ANR-20-CE46-0009) project of the French National Agency for Research (ANR).

non-tracked element in V_x which we named *garbage element*. The purpose of this element is to collect contributions of non-chromatic numbers encountered during the execution of a program such that the property 2.1 is preserved.

One can notice that this element would be optional if there were no rounding errors in any of the operations, since $x = \frac{1}{k_x}\sum_{i=0}^{n} V_x[i]$.

A. *Impact of floating-point arithmetic*

Except for the element k which is an integer, every elements composing a CN are stored as floating-point numbers and are subject to rounding error and cancellation.

The first implication is with the additive property which may not stand exactly as the operations done on the first element x of a CN are different from the operations done on the vector of element V_x.

The second implications is that chromatic analysis will fails delivering information when dramatic cancellation is encountered. Let consider the *(u,v)=FastTwoSum(a,b)* algorithm with $b < a$ which computes $u = a + b$ and $v = b - ((a + b) - a)$ to illustrate this issue. Chromatic analysis will informs us how u is influenced by a and b, but will return a zero vector for v. A non-detailed workaround is to track and propagate rounding-error performed on the first element of a CN along V_x. One can notice that this issue is encountered with automatic differentiation as well.

B. *Implementations*

We implemented the previous arithmetic using operator overloading in both Python and C++. For each of them, we designed two versions, depending on the usage. A first implementation stores the vector V_x as an array of floating-point values. This requires knowing and setting the number of track values at the beginning of the section to be observed. This version is efficient as it is fully vectorized. However, if the vector V_x is mostly sparse, this solution may be inappropriate.

To address this case where the interaction between the tracked values is limited, we designed a second version that represents the vector V_x as a dictionary (hash tables). This is particularly useful to represent sparse contributions and/or to discard contributions that become too small compared to others. However, this solution works element-wise requiring control-flow therefore hindering executions optimisations.

III. EXAMPLES

We illustrate how chromatic information propagate thanks to CN on examples and what kind of informations it can delivers.

A. *Cancellation*

As a first example, lets take 2 CN $a = \langle 2, 1, [0, 2, 0]\rangle$ and $b = \langle 3, 1, [0, 0, 3]\rangle$, the element of index 0 in the vector corresponding to the garbage element. Then the sequence of operations $r = (a.a).b - 12$ will produce the resulting CN $r = \langle 0, 1, [-12, 8, 4]\rangle$ meaning that we have encountered a cancellation involving non-tracked element leading to a value -12, with a sequence of operations involving a and b with the weight of a double the weight of b (respectively 8 and 4).

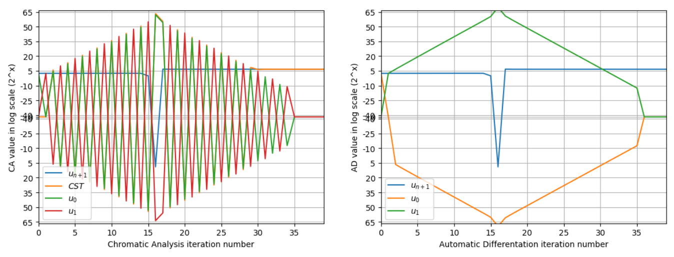

Fig. 1. Chromatic Analysis (left) and Differentiation Analysis (right) for Muller's series, where values are represented using symmetric logarithmic scale.

B. *Muller's series*

For this example, we consider the Muller's series:

$$\begin{cases} u_0 & = & 5.5 \\ u_1 & = & 61.0/11.0 \\ u_{n+1} & = & 111. - 1130./u_n + 3000./(u_n.u_{n-1}) \end{cases}$$

which is known to converge mathematically toward 6 and numerically toward 100 when executed using either binary32 or binary64. When using chromatic analysis on the first 30th iterations we get figure 1(left) with u_0, u_1 tinted.

We can observe on this figure that the relative weight of those two tinted values is oscillating with a huge contribution up to iteration 13, and is vanishing after iteration 35 to the profit of non tracked values (in this case the constants $111, 1130, 3000$). At iteration 13, the sum of the contributions $10^{14} \approx 2^{48}$ is close to the value 5.9 of the series. The different order of magnitude between the contributions and the result suggests an unstable result. After the 35^{th} iteration, u_0 and u_1 does not "influence" the final result any longer and the output value (in blue) is fully linked with non-tracked scalars (in orange).

Muller's series has been widely used as a case study to evaluate how numerical tools behave. On this example, stochastic analysis, Automatic Differentiation or sensitivity analysis provides information about an issues such as instability due to the propagation of rounding errors. Figure 1 (right) represents an Automatic Differentiation analysis according to u_0 and u_1.

C. *DNN*

We have successfully used the Python implementation on the MNIST classification problem, which consists of images of digits from 0 to 9, of shape 28x28x1. We have designed 2 types of tests by reimplementing a 3 fully connected layers deep neural network (28x28, 100, 50, 10) with sigmoid activation function in Python.

The first one is designed to track the relative weight of pixels during the inference phase for a given network. The second one consists in tracking the relative weight of images according to the category to which they belong during the training phase.

1) Inference & adversarial attack: The first test is to track the relative weight of pixels for a given image during the inference phase, so that the resulting classification probability

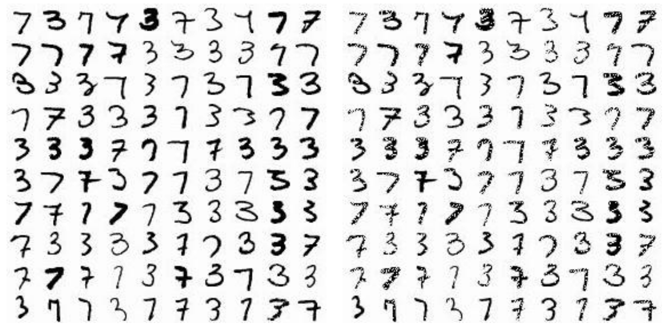

Fig. 2. Adversarial construction on MNIST dataset of 3s and 7s such that each example has a minimal number of pixels alterted to mislead the discrimination between the two sets among the ten classification bins.

can be decomposed according to pixel weight. This is done by assigning a different tint to each pixel of an image.

For the MNIST classification, the input images consist of an array of 28×28 pixels, with the resulting 10 output class corresponding to the probability of an image belonging to a given class. The array of input pixels is first converted to a floating-point number in the range $[0, 1]$, and then to a chromatic number such that pixels of index i are represented as $\langle x_i, V_{xi} \rangle$ where x_i is the float value of the pixel and $V_{xi}[k] = x_i$ for $k = i + 2$ and 0 for all other components.

At the end, we get 10 chromatic numbers, one for each output class. Each number $C_k = \langle c_k, V_{ck} \rangle$ corresponds to the output probability c_k for the given class $k, 0 \leq k \leq 9$, and the weight of each input pixel represented by the vector V_{ck}, such that $\sum_{i=0}^{28*28} V_{ck}[i] = c_k$. With this information, it is possible to perform an adversarial attack, or to determine the transformation to be applied to each input pixel in order to change the output probability classification. This can be done by solving an optimization problem, similar to the fast gradient sign method in [1].

Figure 2 corresponds to a dataset from the MNIST 3s and 7s using a 3 fully connected layers deep neural network (28x28, 100, 50, 10) with sigmoid activation function in Python, altered to mislead all ”7” images toward ”3” and the contrary. If Θ is the parameter of the model, x the input image and y the associated category and $J(\Theta, x, y)$ the cost function, we generated an optimal max-norm perturbation by replacing each pixel of x with a chromatic number to generate a component wise cost function. The number of components corresponds to the number of pixel in the original image.

Let

$$F(i, r) = \sum_{k=0}^{9} J(\Theta, x, y[k])[i] - 2.J(\Theta, x, y[r])[i]$$

with $r = 3$ if the image is a 7, which we want to mislead to a 3, or $r = 7$ on the contrary. Then we iteratively select the pixel i so that

$$i = max\left(|F(i, r)| : i = 0, ..., 28 \times 28\right)$$

and change $x[i]$ to maximize $F(i, r)$ until enough changes are needed. In this way, we minimize the number of modified pixels and the execution overhead induced by using the chromatic number is less than 100.

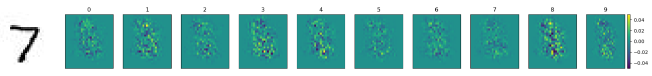

Fig. 3. Example of absolute pixel weight generated to classify image ”7” with a given network trained with chromatic numbers, where image pixels are indexed according to the class to which they belong (index between 0 and 9). On each image, the color of the pixel corresponds to the contribution weight of the pixel.

2) Training & categorized classification: The second test on DNN consists in tracking the relative weight of image classes during the learning phase. The goal is to obtain a resulting network of chromatic numbers tinted according to the image class to which they belong. In this version, each pixel value of a given image is tinted with the same values corresponding to the class between 0 and 9 to which it belongs.

The resulting network is then used during the inference phase to understand the relative weight of each pixel according to the resulting classification vectors. Each chromatic number could be interpreted as the weight of the input class images given during the learning phase that led to the resulting output probability. This corresponds to a simple graphical interpretation of the relationship between the images provided during the training phase and the image provided during the inference phase. An example of such a correlation is shown in figure 3 for the image ”test_0.jpg” from MNIST, which corresponds to number 7. This test was performed by training the network on the first 100 images of the MNIST training set, which takes 5400 seconds using chromatic numbers versus 3 seconds using standard IEEE-754 arithmetic.

IV. Comparison with other arithmetic and tools

There exists other numerical methods and tools delivering related information to CA which we will describe next.

A. *Sensitivity analysis*

Sensitivity analysis is a method used to study how variations in the input parameters of a model affect the output [2]. It is used to determine the effect of small changes in the input parameters on the output of the model and to identify which input parameters have the greatest effect on the output. It allows one to determine the range of input values that result in acceptable output values. It is commonly used in engineering, finance, and economics to evaluate the robustness of a model and to identify potential sources of uncertainty.

Sensitivity analysis can be either local or global. Locally, it tracks how a (small) perturbation of the input perturbs the output. It can be done by computing either the numerical or analytical derivative using for example Automatic Differentiation (section IV-B). It considers one parameter at a time, keeping the rest of the process identical, in order to study its effect of the entire process.

Globally, from multiple runs, it recreates models of output with respect to the influential variables, and evaluates the magnitude of the output with respect to these influential variables using Monte-Carlo technique along with factorial analysis and differential sensitivity analysis.

Among the problems these analyses face, we can cite the curse of dimensionality, their inability to handle correlated input, or to interpret variation on multiple output. The curse of dimensionality is due to the need for multiple runs, which leads to important execution time depending on the input exploration space, that is exponential w.r.t the influential variables. The space exploration relies on measuring the impact of small perturbations on the input, which are explored one at a time, missing the correlation between inputs. In addition, for problems with multiple correlated outputs, the sensitivity measure can be difficult to interpret.

B. Automatic differentiation

Automatic differentiation (AD) is a method for numerically computing the derivative of a function using the basic arithmetic operations of the function [3]. It is an efficient and accurate alternative to numerical methods, such as finite differences, for finding derivatives.

In automatic differentiation, backward and forward accumulation refer to the two main methods for calculating the gradients of a function with respect to its inputs. Both methods are based on the chain rule and the choice of which method to use depends on the specific problem at hand.

It is possible to perform forward automatic differentiation with minor code modifications thanks to *dual number* arithmetic and operator overloading. Each number is given a new component to represent the derivative of a function at the number requiring to define a new algebra. Each number x is replaced by a new number $x = x + x'\epsilon$, where x' is the derivative and ϵ is an abstract number such that $\epsilon^2 = 0$. This arithmetic is implemented by introducing an arithmetic on ordered pairs $\langle x, x' \rangle$ [4], [5].

AD has become the pervasive operation behind all of the machine learning library such as PyTorch or TensorFlow. It has been successfully used to perform adversarial attack on neural network.

We have seen that, AD on Muller's series will allows us to produce the second part of figure 1. This example illustrate the fact that the information produced by AD is very similar to CA as we are able to notice along the iterations when the derivative for (u_0, u_1) vanishes meaning that small modifications on them does not influences the results. On this example, AD lacks information about the set of parameters that influence the results, whereas CA does. The coefficients computed for Muller's series with automatic differentiation are 25 times more important in magnitude at iteration 13 than the coefficients obtained with CA at the same iteration. This conforms to the fact that automatic differentiation computes a local gradient providing information about the neighborhood for a given point according to specific variables whereas CA provides information about how a given point is decomposed according to variables. Moreover, with CA, we have access to the sum of the contributions giving a stability criteria of the result by comparing the order of magnitude of the norm of the contribution vector with the absolute value of the result. This criteria is absent in automatic differentiation. If the contributions alternate in CA whereas the derivative contributions do no change sign along the iterations, this is due to the interpretation of the inverse operation: this operation generates a positive contribution in CA and a negative one in AD for positive values. AD lacks information about why this series is converging toward 100, whereas CA gives this information straightaway with ponderation.

C. Automated sparsity detection

To accelerate AD of complex multivariate programs, it is possible to rely on the computation of the sparsity pattern of the Jacobian or Hessian matrix of term $\frac{dy_i}{dx_j}$ with x and y respectively input and output vectors, such that $y = f(x)$ with f representing an arbitrary program execution [6]. As a 0 in the Jacobian implies that x_j has no influence on y_i, computing the sparsity pattern helps identifying the set of parameter that has an influence over a set of output. The sparsity corresponding to a binary matrix, it does not embed any information about the relative weight but only indicates if a given parameters can influence an output but not by how much. This is why this solution is usually linked with AD.

V. Conclusion

We have presented a solution to answer to the problem of identifying the set of values that influence outputs and how there are related together. The proposed solution rely on an additive rule offering the possibility to merge contribution of several scalars together. Thanks to this possibility, we are able to better deal with the dimensionality problem by either discarding values that become too small or aggregating input values under a given tint.

This work is a preliminary version and as future work, we will look at how this analysis could be linked with other analyses. For example, global sensitivity analysis is very useful, but does not work well when the number of inputs to track is large. In this case, prior knowledge is needed to select the variables of interest. Therefore, chromatic analysis could automatically identify such variables.

References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[2] A. Saltelli, "Sensitivity analysis for importance assessment," *Risk Analysis*, vol. 22, no. 3, pp. 579–590, 2002.

[3] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, 2008.

[4] W. Yu and M. Blair, "DNAD, a simple tool for automatic differentiation of Fortran codes using dual numbers," *Computer Physics Communications*, vol. 184, no. 5, pp. 1446–1452, 2013.

[5] J. A. Fike and J. J. Alonso, "Automatic differentiation through the use of hyper-dual numbers for second derivatives," in *Recent Advances in Algorithmic Differentiation*, ser. Lecture Notes in Computational Science and Engineering, S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds. Berlin: Springer, 2012, vol. 87, pp. 163–173.

[6] S. Gowda, Y. Ma, V. Churavy, A. Edelman, and C. Rackauckas, "Sparsity programming: Automated sparsity-aware optimizations in differentiable programming," 2019.