Abstraction of computation graphs

Paul LANDRIER

1 Introduction

We try to abstract the notion of *computation graphs*, having in mind the particular example of neural networks.

Computation graphs are a type of graphs that represent a computation. Formally, in this work, we define a computation graph (V, E, Fun, Op) as a directed acyclic graph (V, E) together with a function $Fun : E \to F$ where F is a set of functions $S \to S$ (S is a fixed set, the same for all functions) and a function $Op : V \to Operator$ where $Operator$ is a set containing operators to aggregate several inputs in S. (Remark : usually in a computation graph, the functions are stored in the nodes and the edges represent a data dependency. We choose a different convention here to remain closer to provenance in graphs as defined in [TODO:Ramusat].)

For instance, if we want to compute the function $f(x) = e^{x^2}(x^2 + 2x + 2)$ using only the linear, "plus constant" and exponential functions, with $+$ and $\times$ operators, then we can model a possible computation with the computation graph :

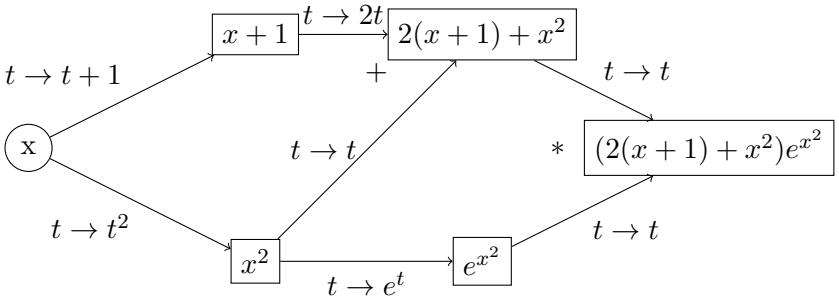


Figure 1: One possible computation graph of f.

Let's take the example of a (non-discretized) perceptron with two inputs. If we want to model its execution using only elementary operations and the activation function f, we get the representation in Figure 2.

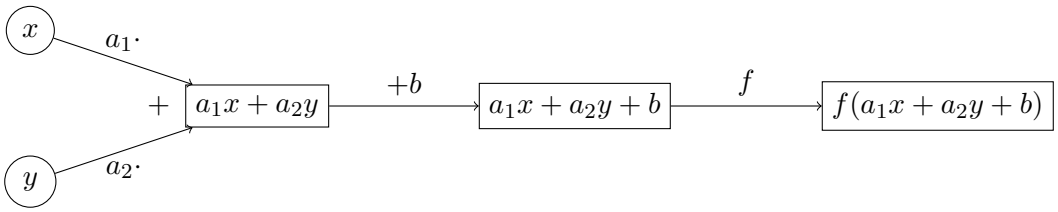


Figure 2: Computation graph of a perceptron.

We could discuss another representation of perceptron with a node storing the value 1, so the $+b$ function is actually another multiplication $(\cdot b)$ from this "1-node" that enters the $(a_1x + a_2y)$-node. This could be convenient for the instanciation of the model we are going to developp (since it simplifies the base set of functions from "activation-multiplication-plus constant" to "activation-multiplication") but to model the special "1-node" we would need to reduce the generality of the structure.

2 Executing a computation

2.1 Model

We attempt to characterize the minimal algebraic structure required to model a computation graph. We fix S the domain of the functions and, keeping in mind our targeted use case, we assume that there is only one operator : $Operator = \{+\}$.

The full computation represented by the graph cannot be expressed in a similar fashion as for provenance in graphs ($\sum_{\pi \in P_{x,y}} w(\pi)$) because composition does not distribute over $+$. In a general computation graph, the algorithm to execute the computation consists of executing the nodes following a topological ordering (by executing a node we mean executing all the functions that enters the node, sum their result and store the value).

Another way to see it is to define the input as the nodes in the graph with in-degree 0 and the output as the node in the graph with out-degree 0. The algorithm to retrieve the function represented by the computation graph from the definition is then a recursive algorithm that builds the function starting at an output node with function identity. The "LabelFun" attribute of input nodes is the a function that takes in argument all the inputs (let's say, as a vector) and returns the value corresponding to the particular input we are considering (typically one entry in the vector).

Algorithm 1 Retrieve Function

```
1: Function RetrieveFunction(Graph)
2:     Output ← Graph.Output
3:     Result ← Retrieve(Output, IdentityFunction)
4:     Return Result
5: End Function
```

Algorithm 2 Retrieve

```
1: Function Retrieve(Node, F)
2:     If Node.IsInput Then
3:         Return Compose(F,Node.LabelFun)
4:     Else
5:         Result ← ZeroFunction
6:         For Each (Ancestor, G, Node) In Node.IncomingEdges Do
7:             Result ← Result + Retrieve(Ancestor, Compose(F, G))
8:         End For
9:         Return Result
10:    End If
11: End Function
```

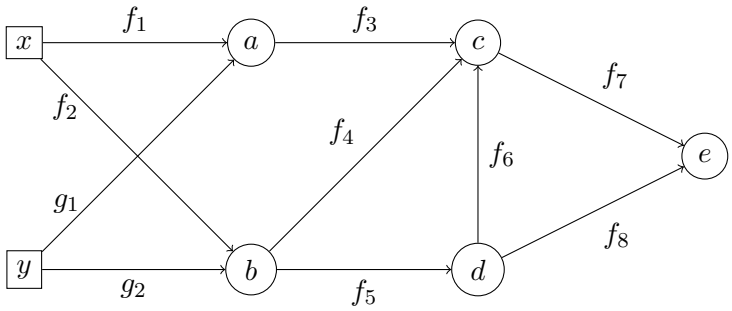


Figure 3: An abstract computation graph.

Property on the computation graph	Algebraic traduction of the property
The edges that enter a node are not ordered	$+$ is associative and commutative
The concatenation of two computation graphs is a computation graph that represent the composition of the functions	$\circ$ is associative
There is a notion of zero-weight that model the fact that there is no interaction between two nodes.	There is a 0_F element that is a neutral element for $+$ and that is left absorbing for $\circ$.
We can reuse computation previously done in a natural way.	The operator $\circ$ is right distributive over $+$.
We should be able to model the identity function.	There is a neutral element id for $\circ$.

Figure 4: Motivation of the algebraic structure.

Structure needed We consider a structure $(F, \circ, +, 0, id)$. The $+$ and $\circ$ are operators $F^2 \to F$ respectively used to model aggregation and composition in the computation graph.

Following what has been made in *provenance semirings* by Green, Karvounarakis and Tannen and by Ramusat in its thesis, in order to identify the structure that best fits our application we try to identify some remarkable properties of computation graph to deduce some properties on the structure used to generalize them. We sum up the process in the table:

Some additional remarks on the structure: [TODO: suppress the useless ones (redundant with what precedes]

[TODO: the distributivity axiom and the composition axiom are both made so we can use some subpart of a graph as black-boxes. Can we prove that if we have a black-box graph representing a function f, we can indeed plot other graphs at the begining/ at the end and the concatenation represents $g \circ f \circ h$?]

Since there is no natural order on incoming edges for a node, $+$ must be associative and commutative so the sum on incoming edges is well-defined. (For instance consider the node c in Figure 3).

We require the operator $\circ$ to be associative so that computations in the graph can proceed either from left to right or from right to left to retrieve the function it represents, and to ensure modularity, which allows us to define the function associated with a subgraph.

To abstract the concept of zero-weight, we require the existence of a neutral element 0 which is a left absorbing element for the $\circ$ operator and which is a neutral element for $+$.

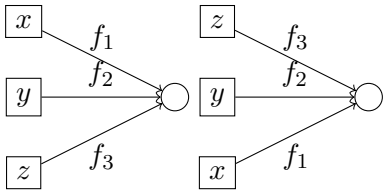
We require the left distributivity $(f + g) \circ h = f \circ h + g \circ h$ beacuse it is of theoretical interest (it is necessary for the interpretation of near-semirings as sub-near-semirings of the the complete near-semiring over a monoid), because it is a natural identity to interpret the elements as functions and because it allows for a simplification of the computation in some cases.

Finally, we require the existence of a neutral element id for $\circ$. This might not be crucial and could change, but it has interexting theoretical properties (it allows to model the one node graph for instance, which can be useful to initialize some algorithms such as Retrieve earlier).

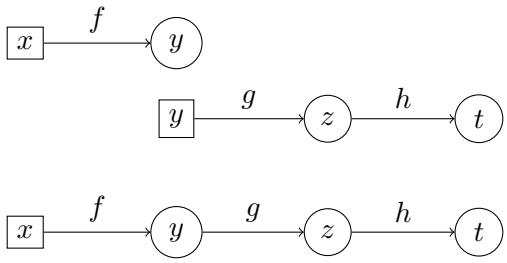
The axiom identified are those of a near-semiring, with two differences ($+$ is not assumed to be commutative in general and the "id" does not necessarily exist). In our case, we obtain the definition:

Definition 1 (Near-semiring)**.** *A near-semiring $(F, +, \circ, 0_F, id)$ is a set F equiped with two binary operations $+, \circ$ and two distinguished elements $0_F, id$ such that :*

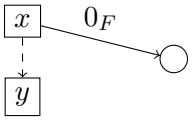
- *$(F, +, 0_F)$ is a commutative monoid;*
- *$(F, \circ, id)$ is a monoid;*
- *$\forall (f, g, h) \in F^3$, $(f + g) \circ h = f \circ h + g \circ h$;*



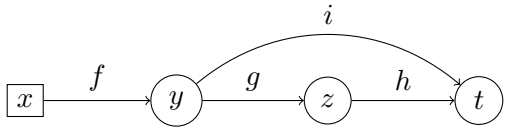
(a) A case that necessits the "+" operator to be commutative and associative. Indeed, these two graphs are the same represented in two different ways, and the left one represent the element $(f_1 + f_2) + f_3$ whereas the right one represents $(f_3 + f_2) + f_1$. Replacing f_3 by 0 then shows that $+$ must be comutative, and using the commutativity of $+$ shows that the right expression is equal to $f_1 + (f_2 + f_3)$, so the associativity is required too.



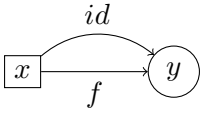
(b) Necessity of the associativity for the modularity. The fact that the graph below, which associated element is $h \circ (g \circ f)$, represents the same element as the composition of the two graphs above, which associated element is $(h \circ g) \circ f$, is equivalent to the associativity of $\circ$. Then we can say without ambiguity that they both represent $h \circ g \circ f$.



(c) Use-cases of the 0 weight. The plain arrow represent an existing edge with weight 0 (in the context of neural network for instance) and the dashed arrow represent an edge that does not exist and could be represented by the 0 element.



(d) Illustration of the modularity allowed by the right distributivity. We can factor the expression $i \circ f + h \circ g \circ f$ in the smaller one $(i + g \circ h) \circ f$.



(e) A use case of the identity: representing $f(x) + x$.

Figure 5: The situations that justify the different properties discussed in Figure 4.

- 0_F *is left absorbing for* $\circ$*, i.e. for all* $f \in F$*,* $0_F \circ f = 0_F$*.*

Together with this definition comes the notion of morphism of near-semiring and of sub-near-semiring:

A sub-near-semiring of a near-semiring $(F, +, \circ, 0, id)$ is a subset $G \subset F$ such that $0 \in G, id \in G$ and G is stable by $+$ and $\circ$. If G is a sub-near-semiring of $(F, +, \circ, 0, id)$, then $(G, +, \circ, 0, id)$ (for the restriction of operations $+$ and $\circ$ to G) forms a near-semiring.

If $(F, +, \circ, 0, id)$ and $(G, +', \circ', 0', id')$ are two near-semirings, then $\phi : F \to G$ is said to be a near-semiring morphism if it satisfies $\phi(0) = 0', \phi(id) = id', \forall (f, g) \in F^2, \phi(f + g) = \phi(f) +' \phi(g)$ and $\phi(f \circ g) = \phi(f) \circ' \phi(g)$. The image $\phi(F)$ is a sub-near-semiring of G.

2.2 Examples

We should now study a few examples of near-semiring.

Near-semiring 1 (Complete near-semiring over a monoid)**.** *Let* $(M, +, 0_M)$ *be a monoid. We define* $\mathcal{F}$ *the set of all functions* $M \to M$*. Then the tuple* $(\mathcal{F}, +, \circ, 0_F, id)$ *where* $0_F : x \mapsto 0_M$ *is the zero function and* $id : x \mapsto x$ *is the identity function is a near-semiring, that we will call complete near-semiring over* M*. A particular sub-near-semiring of* $\mathcal{F}$ *is the set of monoid endomorphisms of* M *(it is even a semiring).*

Near-semiring 2 (Semirings)**.** *Let* $(S, +, \cdot, 0, 1)$ *be a semiring. Then* $(S, +, \cdot, 0, 1)$ *is also a near-semiring. Indeed, the axioms of near-semiring are stricly more general than those of a semiring. The interpretation of semirings as sets of function together with the interpretation of* $\cdot$ *as a composition can be made through the structure of semimodules, which is itself a sub-near-semiring of the complete near-semiring over a monoid.*

Near-semiring 3. *Matrices over a semiring*

2.3 Universal object

Given a near-semiring F, we want to characterize the sub-near-semiring generated by a subset $G \subset F$. This notion makes sense for the usual reason: an intersection of sub-near-semiring is a sub-near-semiring itself, and we note $< G >$ this sub-near-semiring .

For this purpose we define a sequence of sets (G_n) by $G_0 = \{0, id\} \cup G$ and

$$G_{n+1} = \left\{ g \circ \left(\sum_{i=1}^{k} g_i \right) \middle| \ k \in \mathbb{N}, g \in G, (g_i)_{1 \leqslant i \leqslant n} \in (G_n)^k \right\}.$$

The claim is now that $< G >= \underset{n \in \mathbb{N}}{\cup} G_n$. Indeed, $G \subset G_0 \subset \underset{n \in \mathbb{N}}{\cup} G_n$, then we show by induction that $\forall n \in \mathbb{N}, G_n \subset < G >$ and finally we show that $\underset{n \in \mathbb{N}}{\cup} G_n$ is indeed a sub-near-semiring of F.

2.4 Generality of the complete semiring over a monoid.

Theorem 1. *Universal Form of Near-semirings*

It is equivalent to be given:

1. *a near-semiring* $(F, +, \circ, 0, id)$
2. *a commutative monoid* $(M, +, 0)$ *together with a set of functions* $M \to M$ *containing* 0 *and* id *and stable by* $\circ$ *and* $+$*. (I.e. a sub-near-semiring of the complete near-semiring over* M*.)*

Proof. Firstly, we assume that we are given a monoid $(M, +, 0)$ and a set of functions $M \to M$ containing 0, id and stable by $+$ and $\circ$ like in two. This precisely means that we are given a sub-near-semiring of the complete near-semiring over $(M, +, 0)$, and hence a near-semiring.

Secondly, if we are given a near-semiring $(F, +, \circ, 0, id)$, we define for an element f the function

$$\bar{f} : \left\{ \begin{array}{ccc} F & \to & F \\ g & \mapsto & f \circ g \end{array} \right. ,$$

then we define the function $\phi : \left\{ \begin{array}{ccc} F & \to & ((F,+,0) \to (F,+,0)) \\ f & \mapsto & \bar{f} \end{array} \right.$.

The function ϕ is a morphism of near-semirings. Indeed:

- $\phi(0)$ is the zero function since zero is left absorbing,
- $\phi(id)$ is the identity function because $\forall g \in F, id \circ g = g$,
- $\forall f_1, f_2 \in F^2,\ \forall g \in F,\ \phi(f_1 + f_2)(g) = (f_1 + f_2) \circ g = f_1 \circ g + f_2 \circ g = \phi(f_1)(g) + \phi(f_2)(g)$, hence $\phi(f_1 + f_2) = \phi(f_1) + \phi(f_2)$
- $\forall f_1, f_2 \in F^2,\ \forall g \in F,\ \phi(f_1 \circ f_2)(g) = (f_1 \circ f_2) \circ g = f_1 \circ (f_2 \circ g) = \phi(f_1)(f_2 \circ g) = (\phi(f_1) \circ \phi(f_2))(g)$, hence $\phi(f_1 \circ f_2) = \phi(f_1) \circ \phi(f_2)$

Therefore, $\phi(F)$ is a sub-near-semiring of the complete near-semiring over the monoid $(F, +, 0)$.

□

This theorem is interesting because it clearly indicates a direction for further investigation -if we want to instantiate the theoretical framework, we should define computation graphs over some monoids- and a limit of the expresiveness of the theory. However, it does not provide any information on the structure of near-semirings. For instance, we have an analoguous theorem on groups that states that all groups are subgroups of a bijection group, but this is of little interest when we want to study the structure of groups (like classification of finite abelian groups).

It might be interesting to note that for the example of computation graphs we restrict ourselves to finitely generated monoids, because the computation only involves a finite number of elements.

3 Instantiation of the theory

3.1 Layer-Wise Relevance Propagation

[Founded on LRP overview, https://iphome.hhi.de/samek/pdf/MonXAI19.pdf]

[Dependance or not on an execution, want to do the computation once or several times]

Layer-Wise Relevance Propagation (LRP) is a technique used to explain the prediction made by a neural network on one of its execution. The idea of the technique is to propagate backwards the contribution (or the *relevance*) of neurons in the network to the final prediction of the model. The general formula is

$$R_j = \sum_k \frac{z_{j,k}}{\sum_j z_{j,k}} R_k.$$

The value R_j is the score of the neuron j, and it is computed by summing over all neurons k in the next layer the term $\frac{z_{j,k}}{\sum_j z_{j,k}} R_k$, where R_k is the relevance of neuron k and $\frac{z_{j,k}}{\sum_j z_{j,k}}$ represents the extent to which neuron j contributed to the relevance of the neuron k.

The normalization term ensures that vanishing phenomenoms cannot occur: the sum of the relevances of neurons on a given layer is preserved.

We should now discuss the interpretation of LRP as an instantiation of our model.

On the one hand, we can model LRP with generalized computation graphs. Indeed, the input nodes of the computation graphs are the output neurons of the network (or even the only neuron corresponding to the class predicted, since the value of other neurons is set to zero), the edges of the computation graph are the reversed edges of the neuron and the associated function is the multiplication by the scalar $\frac{z_{j,k}}{\sum_j z_{j,k}}$ (between neurons k and j).

However, on the other hand, this model might have a limited interest.

3.2 Shapley Values

Any chance it helps for Shapley values?

4 Training a model

Want to add a derivative d. Forces to add a $\cdot$. What identities ? $\rightarrow d(f \circ g)$?

5 Experiments ?

6 Further Investigations

Other types of NN ?