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).

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 :

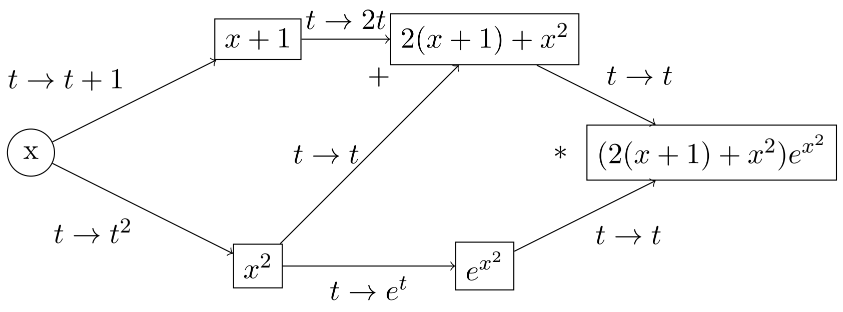

Figure 1: One possible computation graph of f.

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 :

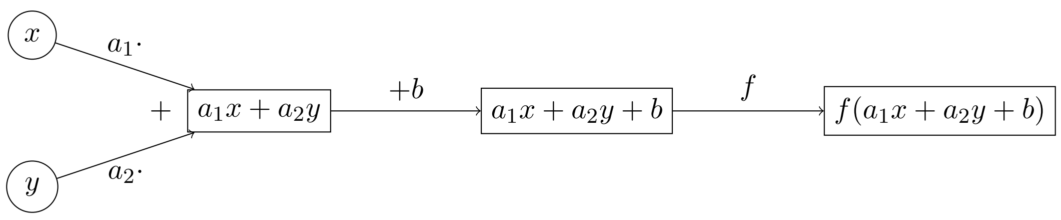

Figure 2: Computation graph of a perceptron.

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.

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 F
 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
```

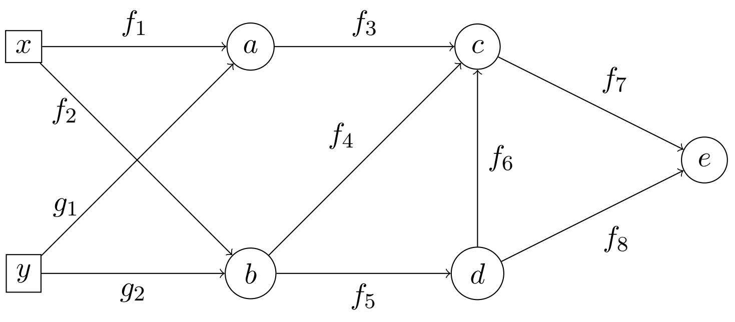

Figure 3: An abstract computation graph.

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:

Property on the coputation graph
The edges that enter a node are not ordered The concatenation of two computation graphs is a computation graph that represent the composition of the
•
•
•

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;*
- *$\forall (f,g,h) \in F^3$, $(f+g)\circ h = f\circ h + g\circ h$;*
- *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 $\phi(f \circ g) = \phi(f) \circ' \phi(g)$.

2.2 Examples

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

Near-semiring 1 (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.*

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.*

2.3 Universal object

The near-semiring generated by a finite number of elements.

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 monoid* $(M, +, 0)$ *together with a set of functions* $M \to M$ *containing* 0 *and id and stable by* $\circ$ *and* $+$.

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 means that we are given a sub $\square$

3 Training a model

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

4 Experiments ?