S2 Text

Conceptually, the learning of associations can be regarded as a (second-order) function that takes a list of pairs and returns a network (graph) of associations, where each node of the graph represents an associated entity and each (directed) edge is labeled with the strength of association from the source associate to the target associate. Note that the returned graph also represents a (first-order) function, which justifies regarding the learning of associations as a second-order function (second-order cognitive capacity). Lists can be built up inductively—a list is either the empty list, or an item prepended to a list. This inductive construction provides the basis for a recursive model of systematic learning of paired associates, i.e., if there are no more paired associates, then return the constructed network, else add the current paired associate to the network obtained from the learning of the remaining list of associates. The following two examples provide a conceptual guide to this (dual) algebraic approach, and also motivate our coalgebraic model.

Example 1 (Summing). The summing of a list of numbers can be performed by a type of function called a fold over lists, denoted fold(0, +), which is an instance of a catamorphism [1]. The first argument, 0, assigns the result of zero to the empty list, and the second argument, +, adds the head (first element) of the list to the fold of the tail (remaining elements) of the list. Thus, for example,

\[
\text{fold}(0, +)[1, 2, 3] = 1 + \text{fold}(0, +)[2, 3] \\
= 1 + (2 + \text{fold}(0, +)[3]) \\
= 1 + (2 + (3 + \text{fold}(0, +)[3])) \\
= 1 + (2 + (3 + 0)) = 6.
\]

Example 2 (Counting). Counting list elements is computed by the catamorphism, fold(0, inc) : L → N, where the first argument, 0, assigns the result of zero to the empty list, and the second argument, inc, ignores the head (first element) of the list and increments the count of the tail (remaining elements) of the list (cf. Diagram 2). In other words, the count of a list is either 0, for the empty list, or one plus the
count of the rest (tail) of the list. For instance,

\[
\text{fold}(0, \text{inc})[a, b, c] = 1 + \text{fold}(0, \text{inc})[b, c] = 1 + (1 + \text{fold}(0, \text{inc})[c]) = 1 + (1 + (1 + 0)) = 3.
\]

A categorical treatment of recursion starts with a suitable endofunctor \( F : C \to C \) from which we construct the category \( \text{Alg}(F) \), whose objects are \( F \)-algebras, i.e., the pairs \((X, \phi)\) consisting of objects \( X \) and morphisms \( \phi : F(X) \to X \) in \( C \), and morphisms are \( F \)-algebra homomorphisms, i.e., the morphisms \( h : (X, \phi) \to (Y, \psi) \) such that \( h \circ \phi = \psi \circ F(h) \). An initial algebra in \( \text{Alg}(F) \) is an \( F \)-algebra \((A, \alpha)\) such that for each \( F \)-algebra \((X, \phi)\) in \( \text{Alg}(F) \) there exists a unique \( F \)-algebra homomorphism \( u : (A, \alpha) \to (X, \phi) \), called a catamorphism and denoted \( \langle \phi \rangle \), since it is uniquely determined by \( \phi \) [1]. Thus an initial algebra is an initial object (dual to terminal object) in \( \text{Alg}(F) \).

In greater detail, a catamorphism \( k : (A, \text{in}) \to (B, \beta) \) from initial \( F \)-algebra \((A, \text{in})\) to \( F \)-algebra \((B, \beta)\) as indicated by

\[
\begin{array}{c}
F(A) \xrightarrow{\text{in}} A \\
\downarrow \quad \downarrow \quad \downarrow k \\
F(B) \xrightarrow{\beta} B
\end{array}
\]

Catamorphism \( k \) is denoted \( \langle \beta \rangle \), by banana brackets [2], since \( k \) is determined by \( \beta \). Catamorphism is also called fold. For lists built from elements of \( A \), catamorphisms are given by

\[
\begin{array}{c}
1 + A \times L \xrightarrow{\text{empty, cons}} L \\
\downarrow 1 + \alpha \times \langle l_v, f \rangle \quad \downarrow \langle l_v, f \rangle \\
1 + A \times X \xrightarrow{\langle l_v, f \rangle} X
\end{array}
\]

where \( \text{empty} : * \to [\ ] \) returns the empty list, \( \text{cons} : (h, t) \to h \cdot t \) returns the list with (head) element \( h \) prepended to (tail) list \( t \), and \( 1_v \) assigns the value \( v \) to the empty list. The morphism component of
an initial algebra is an isomorphism. For \([\text{empty}, \text{cons}]\), the inverse is \((\text{empty}? \rightarrow I, (\text{head}, \text{tail}))\). Thus, traversing Diagram 2 from \(L\) to \(X\) counterclockwise yields the following definition:

\[
\langle l_v, f \rangle : \\
\begin{cases}
\; \; \; \emptyset \rightarrow v; \\
\; \; \; \langle h \cdot t \rangle \rightarrow f(h, \langle l_v, f \rangle(t)).
\end{cases}
\]

We also write \(\langle l_v, f \rangle\) as \(\text{fold}(l_v, f)\).

For our purposes we employ an endofunctor on the category \(\textbf{Set}\) for list-related constructions, i.e., \(F_A : X \mapsto 1 + A \times X\), where \(A\) is the set of possible list elements, 1 is the \textit{terminal object} (associated with the empty list), and operators \(\times\) and \(+\) are the \textit{Cartesian product} and \textit{disjoint union} of sets. Recall that the Cartesian product \(A \times B\) is the set of all pairwise combinations of the elements of \(A\) and \(B\). The disjoint union \(A + B\) is the set of all elements from either \(A\) or \(B\) labeled to identify their set of origin. The initial algebra is \((L, [\text{empty}, \text{cons}])\), where \(L\) is the set of lists constructed from the elements of \(A\), \(\text{empty}\) is the function that creates the empty list, and \(\text{cons}\) is the function that prepends an item to a list to create a new list. Every catamorphism \(\langle l_v, f \rangle\) from the initial algebra in this category has the form given by the following (commutative) diagram:

\[
\begin{array}{c}
1 + A \times L \\
\Downarrow \quad [\text{empty}, \text{cons}] \\
L
\end{array}
\begin{array}{r}
\Downarrow \quad [1_v, f] \\
X
\end{array}
\]

where function \(1_v\) assigns the value \(v\) to the empty list.

The function \(\text{fold}(0, +)\) is an instance of the catamorphism given by Diagram 3, i.e., \(1_v = 0\), the constant function that returns zero, \(f = +\), and hence \(\langle l_v, f \rangle = \text{fold}(0, +)\). Object \(A = X = \mathbb{N}\), the set of natural numbers.

For our model of learned associations, we instantiate Diagram 3 as follows:

- \(A\) is the set of paired associates \(P = \{(a, b) \in S \times S\}\), where \(a\) and \(b\) are (respectively) the source and target entities from a set of entity representations \(S\);

- \(X\) is the set of (labeled) directed graphs (association networks) \(G\), where each graph \(g \in G\) is a pair \((E, V)\) consisting of a set of edges \(E\) and a set of vertices \(V\), and each edge is a triple \((s, \sigma, t)\),...
Second-order systematicity of associative learning

where \( s \) and \( t \) are the source and target vertices and \( \sigma \) is the strength of association;

- \( \text{I}_e \) is the function \( \text{I}_e \) that assigns the empty list to the empty graph \( e \); and
- \( f \) is the function \( \text{assoc} \) that takes a paired associate and a network graph to returns a network graph.

Hence, our model for learned associations is the catamorphism \( (\text{I}_e, \text{assoc}) : L \rightarrow G \). The function \( \text{assoc} : P \times G \rightarrow G \) creates a network of associations from a paired associate and an association network. Informally, if the graph does not contain an edge corresponding to the given pair, then a new edge with an initial strength of association is added to the graph, else the strength of association for the existing edge is updated.

A simple example serves to illustrate the mechanism of associative learning as a catamorphism. Suppose we have the following list of paired associates: \([(\text{bread}, \text{butter}), (\text{fork}, \text{knife}), (\text{knife}, \text{butter})]\). Observe that the catamorphism \( (\text{I}_e, \text{assoc}) : L \rightarrow G \) is obtained from the composition of three functions by traversing Diagram 3 from \( L \) to \( X = G \) counterclockwise, i.e., \( (1) [\text{empty}, \text{cons}]^{-1}, (2) 1 + 1 \times (\text{I}_e, f), \) and \( (3) [\text{I}_e, f] \). The first function pulls apart the list of paired associates. This inverse function is guaranteed to exist because \( [\text{empty}, \text{cons}] \) is an isomorphism. The second function, which contains the catamorphism, is also (recursively) obtained by the composition of three functions whenever the list is not empty. The third function constructs the associative network from the current paired associates and the network (recursively) constructed from the previous list. Thus, for our example, the pair \( (\text{bread}, \text{butter}) \) is added to the network constructed from the list \([(\text{fork}, \text{knife}), (\text{knife}, \text{butter})] \), which is constructed by adding \( (\text{fork}, \text{knife}) \) to the network constructed from the list \([(\text{knife}, \text{butter})] \), which is constructed by adding \( (\text{knife}, \text{butter}) \) to the network constructed from the empty list, which is just the empty graph. So, the catamorphism constructs the following network, with associative strengths \( \sigma_i \):

\[
\begin{align*}
\text{bread} & \xrightarrow{\sigma_1} \text{butter} \\
\text{knife} & \xrightarrow{\sigma_2} \text{fork}
\end{align*}
\]

Second-order systematicity says that if we can learn one collection of word associations (e.g., in English) then we can learn another collection of associations (e.g., in Japanese), assuming of course capacity for English and Japanese words. Such instances of associative learning are catamorphisms from
the same initial algebra, as described by Diagram 3, i.e., replace $X$, which is instantiated as the set of graphs on English words in the discussion above, with the set of graphs on Japanese words. Moreover, initial algebras are instances of universal morphisms, specifically initial morphisms.

References

1. Bird R, de Moor O (1997) Algebra of programming. Harlow, England: Prentice Hall.

2. Meijer E, Fokkinga M, Paterson R (1991) Functional programming with bananas, lenses, envelopes and barbed wire, Berlin, Germany: Springer-Verlag, volume 523 of Lecture Notes in Computer Science. pp. 125–144.