4

Generalized Lattices Express Parallel Distributed Concept Learning

Michael J. Healy and Thomas P. Caudell

University of New Mexico mjhealy@ece.unm.edu
Albuquerque, New Mexico 87131 tpc@ece.unm.edu

Summary. Concepts have been expressed mathematically as propositions in a distributive lattice. A more comprehensive formulation is that of a generalized lattice, or category, in which the concepts are related in hierarchical fashion by lattice-like links called concept morphisms. A concept morphism describes how an abstract concept can be used within a more specialized concept in more than one way as with "color", which can appear in "apples" as either "red", "yellow" or "green". Further, "color" appears in "apples" because it appears in "red", "yellow" or "green", which in turn appear in "apples", expressed via the composition of concept morphisms. The representation of such concept relationships in multi-regional neural networks can be expressed in category theory through the use of categories, commutative diagrams, functors, and natural trasformations. Additionally, categorical model theory expresses the possible worlds described by concepts. The analysis of morphisms between the possible worlds highlights the importance of reciprocal connections in neural networks.

4.1 Introduction

The contributions in this volume discuss lattice theory from different perspectives. Some relate it to neural network algorithms in which the partial order relation ($\leq$) and the meet/join operations ($\wedge$, $\vee$), regarded as min/max operations, are part of the computational model. There is a history of lattice theory in neural network theoretical models [12, 18, 28], and in some the meet/join operations have other interpretations. One of these involves symbolic languages for modeling the semantics of neural computation. Symbolic languages including formal logics have appeared over the years in attempts to obtain a mathematically precise expression of the semantics of neural networks [2, 3, 6, 12, 13, 17, 19, 24, 26, 27]. In addition to classical propositional and first-order logic [3], the logics used have included fuzzy logic [24], geometric logic [12], and non-monotonic logics [26]. In the semantic model of [12], the input data for a neural network represent paired entities from two domains. The network's long-term adaptation is explained as the learning of

M.J. Healy and T.P. Caudell: *Generalized Lattices Express Parallel Distributed Concept Learning*, Studies in Computational Intelligence (SCI) **67**, 59–77 (2007)
www.springerlink.com

a geometric logic theory about each domain while simultaneously learning an inference relation between the theories that expresses a mapping between their domains. Because of the form of geometric logic used, this yields lattices coupled by a very special type of lattice homomorphism. In the process of learning, the network modifies its connection weights based upon input data, and this effectively combines domain knowledge extracted from the data with pre-existing knowledge to derive new theories and new inferences between their formulas. Each initial theory consists of a set of predicates representing observable features of the entities, and proceeds through the adaptation process to include derived formulas and their relationships, and these and the modified inferencing express the information gained.

Vickers applies geometric logic, a non-Boolean logic similar to intuitionistic logic ([29, 30]), to domain theory. The logic is applied as a "logic of finite observations". This derives from the interpretation of a statement Q in the logic as affirmable: It cannot be falsified, only affirmed through demonstration (observation or proof). The axioms of a geometric theory specify basic knowledge about some domain. As with most logics, the propositions, predicates, the formula-constructing operations—in this case conjunction and disjunction along with the existential quantifier $\exists$—- and the proof theory of the logic, resulting in entailments such as $\exists x P(x) \vdash \exists x Q(x)$, are used to reason about any domain to which the theory applies. In the logic of finite observations, however, negation $\neg$, implication formulas $P(x) \rightarrow Q(x)$ and the universal quantifier $\forall$ are used only in formulating the axioms. Further, as with intuitionistic logic, the Law of the Excluded Middle (LEM, $\forall x(P(x) \vee \neg P(x))$ is not an axiom of geometric logic: It must be either established as valid when it applies or assumed as an axiom of a theory where its use is desired. The analyst has the task of formulating axioms for a theory sufficiently detailed to express all relevant, pre-existing knowledge about the domain of investigation and then applying the geometric operations to make inferences about new data.

Propositional geometric logic is a special case of this. For a propositional theory formulated in geometric logic, a lattice can be constructed from the axioms, other propositions, and the entailments, where $\vdash$ becomes the lattice order relation $\leq$. Inference is then carried out based solely upon the lattice operations of meet ($\wedge$) and join ($\vee$), interpreted as conjunction and disjunction. The full logic with predicates, quantifiers and open formulas, on the other hand, requires a structure more general than a lattice— it requires category theory.

Yet, single-argument predicates of the form $P(x)$ are used within a lattice in [12]. The explanation for this is that the argument x represents the observations in a domain, not individual entities discussed within the domain theory, and the predicate is affirmed when an observation applies to it. This notation is a convenience for conducting an analysis over multiple domains while maintaining the utmost simplicity. The different domains are obtained by coupling networks with systems of interconnects to form a composite or

multi-regional network. The example examined in [12] is that of a LAPART network [13], a coupling of two ART-1 networks in such a way that the composite network can learn inferences across theories in addition to learning the individual theories. From this, a new theory can be formed that combines the newly-formed sub-network theories together with the newly-formed implications between their formulas, all based upon observation. A composite domain accompanies the composite theory, having the form of a product—if the domains are thought of as sets, the composite in this case is a cartesian product. A finding of this analysis is that most of the structure supporting the learning of coupled domain theories is missing from the example architecture.

The domain theories in propositional geometric logic yield upper-complete distributive lattices, having infinite as well as finite joins and having finite meets (the lattices have infinite meets, but they are non-geometric, so are not used). Mathematically, an advantage of this use of geometric logic is that a theory in the logic directly corresponds to a topology for its domain. This makes it possible to perform model-theoretic analyses in terms of topological spaces and continuous functions (although this is not quite point-set topology, since a slight modification to the set union operation must be made). Thus, the application of geometric propositional logic combines the advantages of logical inference, lattice theory, topology, and domain theory in describing the semantics of neural networks. This facilitates the design of improved neural networks by noting which entailments, lattice meets and joins, and continuous structures are missing from an existing architectural model. The analysis is not restricted to binary-input neural networks because graded values can be represented in a quantized form (see, for example, [13]). With all its advantages, however, this kind of analysis reveals only a part of the gap between present-day architectures and the full potential of neural networks.

More recently, we have been experimenting with a much more comprehensive semantic theory for neural networks based upon category theory [16]. Here, the main application of category theory is not to represent the predicate version of geometric logic and assign formulas with predicates and quantifiers to network nodes. Instead, neural network nodes are still regarded as representing closed logical fragments, but these are whole domain theories and the theories have quantities of more than one type. They are associated with network nodes based upon the nodes' input connection pathways traced back to the input nodes, which are assigned theories that describe the properties of the input features. Also, the logic in which the theories are expressed is left to the analyst's choice; it can be geometric, intuitionistic, classical first-order, or fuzzy, and so forth. When adopted in full, this strategy enables the expression of neural network semantics in the framework of categorical logic and categorical model theory, resulting in a deeper and more useful analysis. One example of this is an experiment with an application of a neural network to multi-spectral imaging, in which an existing architecture was re-designed to produce images with a significantly higher quality [15]. Here, we provide a brief recounting of the categorically-based semantic theory.

4.2 Lattices and Categories

A category can be thought of as a system of mathematical structures of some kind, concrete or abstract, together with the relationships between them that express that type of structure [1, 7, 20, 21, 25]. Each relationship, called a morphism or arrow, has the form $f\colon a \longrightarrow b$ with a *domain* object a and a *codomain* object b. A lattice is a special case of a category in which the morphisms are the relations $a \leq b$. In a *category* C, each pair of arrows $f : a \longrightarrow b$ and $g : b \longrightarrow c$ (with a head-to-tail match, where the codomain b of f is also the domain of g as indicated) has a *composition* arrow $g \circ f : a \longrightarrow c$ whose domain a is the domain of f and whose codomain c is the codomain of g. In a lattice, of course, this expresses the transitivity of $\leq$. Composition satisfies the familiar associative law, so that in triples which have a head-to-tail match by pairs, $f\colon a \longrightarrow b$, $g\colon b \longrightarrow c$ and $h\colon c \longrightarrow d$, the result of composition is order-independent, $h \circ (g \circ f) = (h \circ g) \circ f$. Also, for each object a, there is an *identity morphism* $\mathtt{id}_a\colon a \longrightarrow a$ (in a lattice, $a \leq a$) such that the identities $\mathtt{id}_a \circ g = g$ and $f \circ \mathtt{id}_a = f$ hold for any arrows $f\colon a \longrightarrow b$ and $g\colon c \longrightarrow a$.

Unlike in a lattice, there can be many morphisms in either or both directions between a pair of objects a and b. The category **Set** of sets and functions (with composition of functions) is a familiar example, for given arbitrary sets α and β, there can be many functions with either set as domain or codomain. With a multiplicity of morphisms existing between two objects in a typical category, and given the notion of composition, the notion of a *commutative diagram* takes on great significance. A diagram in a category C is simply a collection of objects and morphisms of C (the domain and codomain objects of a morphism are always included with it). In a commutative diagram, any two morphisms with the same domain and codomain, where at least one of the morphisms is the composition of two or more diagram morphisms, are equal. *Initial and terminal objects* (the bottom and top elements in a lattice) are also important when they exist in a category C. An initial object i is the domain of a unique morphism $f\colon i \longrightarrow a$ with every object a of C as codomain. A terminal object t has every object a of C as the domain of a unique morphism $f\colon a \longrightarrow t$ with t as codomain.

The principle of duality is a fundamental notion in category theory. The dual or opposite C^{op} of a category C has the same objects, and the arrows and compositions $g \circ f$ reversed, $f^{\mathrm{op}} \circ g^{\mathrm{op}}$. The *dual of a statement* in category theory is the statement with the words "domain" and "codomain", "initial" and "final", and the compositions reversed. If a statement is true of a category C, then its dual is true of C^{op}; if a statement is true of all categories, the dual statement is also true of all categories because every category is dual to its dual. Roughly speaking, "half the theorems of category theory are obtained for free", since proving a theorem immediately yields its dual as an additional theorem (see any of [1, 21, 25]).

In addition to the widely-used notion of duality, category theory provides a mathematically rigorous notion of "isomorphism", a term which is often used in a loose, intuitive sense. One sometimes hears a statement such as "the two [concepts, data types, program constructs, etc.] are in some sense isomorphic". If the entities under discussion can be formalized as objects in a category, one can make such statements with mathematical rigor. If a, b are objects of a category C such that there exist arrows $f\colon a \longrightarrow b$ and $g\colon b \longrightarrow a$ with $f \circ g = \mathtt{id}_b$ and $g \circ f = \mathtt{id}_a$, then the morphism f is called an *isomorphism* (as is g also) and g is called its *inverse* (and f is called the inverse of g), and the two objects are said to be isomorphic. The property of an identity morphism ensures that isomorphic objects in a category are interchangeable in the sense that they have the same relationships with all objects of the category. It is easily shown that all initial objects in a category are isomorphic, and the same holds for terminal objects (see the above definitions for initial and terminal objects).

Let Δ be a diagram in a category C, shown in Fig. 4.1 with objects a_1, a_2, a_3, a_4, a_5 and morphisms $f_1\colon a_1 \longrightarrow a_3$, $f_2\colon a_1 \longrightarrow a_4$, $f_3\colon a_2 \longrightarrow a_4$, $f_4\colon a_2 \longrightarrow a_5$. Also shown are two cone-like structures, K' and K''. Each of these *cocones* extends Δ, forming a commutative diagram: For example, K' adds an object b' and morphisms $g'_i\colon a_i \longrightarrow b'$ $(i = 1, \ldots, 5)$ such that $g'_3 \circ f_1 = g'_1 = g'_4 \circ f_2$ and $g'_4 \circ f_3 = g'_2 = g'_5 \circ f_4$. The cocones for Δ are objects in a category $\mathbf{coc}_\Delta$ whose morphisms are morphisms of C from one cocone apical object to the other, $h\colon b' \longrightarrow b''$, having the property that the legs of the codomain cocone K'' factor through the legs of the domain K' and the morphism h,

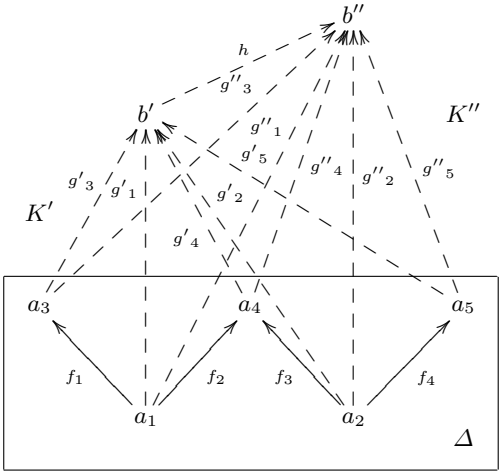

Fig. 4.1. A cocone morphism $h\colon K' \longrightarrow K''$ in $\mathbf{coc}_\Delta$ is a morphism $h\colon b' \longrightarrow b''$ in C between the apical objects b' and b'' of cocones K' and K'', respectively, that is a factor of each leg morphism $g''_i\colon a_i \longrightarrow b''$ of K'', with $g''_i = h \circ g'_i$

$$g''_i = h \circ g'_i \quad (i = 1, \ldots, 5)\,. \tag{4.1}$$

With morphisms so defined, the composition of cocone morphisms follows directly. A *colimit for the diagram* Δ is an initial object K in the category $\mathbf{coc}_\Delta$. That is, for every other cocone K' for Δ, there exists a unique cocone morphism $h\colon K \longrightarrow K'$. The original diagram Δ is called the *base diagram* for the colimit and the diagram $\overline{\Delta}$ formed by adjoining K to Δ is called its *defining diagram*. By *initiality*, all colimits for a given base diagram are isomorphic. A *coproduct* is the colimit for a discrete diagram, one having objects but no morphisms among them except the identity morphism for each diagram object, which is always assumed to be present in a diagram. In **Set**, for example, coproducts are disjoint unions of the component sets.

Limits are the dual notion to colimits, obtained by "reversing the arrows" and interchanging "initial" and "terminal". By duality, a limit for a diagram Δ, if one exists, is a terminal cone, whose morphisms are directed *into* the diagram. A limit for a discrete diagram is called a *product*, and the leg morphisms are called projections. The familiar cartesian product of sets is an example in the category **Set**. Limits and colimits are useful constructs where they are available, and colimits have a history of use in categorical logic and computer science ([11, 32]). A theorem in category theory can be used to derive an algorithm for calculating limits in any category that contains limits for all of its diagrams, and similarly for colimits by dualization (see The Limit Theorem in [25]).

4.3 A Category of Concepts

We express the semantics of a system in terms of a distributed system of concepts about the system's environment as well as the system itself. This manner of describing a system can be thought of as a knowledge representation that is implicit in the system's disposition to sample its environment and act upon it. The concept system, of which the knowledge representation is a part, is an ontology for the system's environment as experienced through the system inputs, both external to the system and possibly internally-generated as well. The ontology is expressed mathematically as a category **Concept** whose objects are symbolic descriptions (concepts) of domains of items (sensed entities, events, situations, and their parts). The concepts are represented incrementally in an adaptive system based upon sensor input, and, where the capabilities exist, are based upon the interplay of sensor input with efference copy (where motor command outputs are used also as system inputs) and with feedback from higher-level processing to levels of processing more directly associated with the sensors. For example, the concept representations in a neural network are formed through modification of connection weights, which in some networks exist in both feedforward and feedback connection pathways. Determining the knowledge representation capability of a given

neural architecture, and designing architectures with a desired capacity, are major goals of analysis using the semantic theory.

Through feedback, sensor inputs can be filtered and adaptively-formed concept representations learned from them can be made consistent with the already-existing knowledge representation store in, say, a cognitive system. We refer to concept representations formed at or near the sensor level through this two-way processing as *percepts*, even when they are not associated with conscious awareness or a biological organism. Concept representations formed through further processing at a "higher" level are, as a consequence, based in perception. This view of concept representation based in perception has important implications for theories of cognition and intelligence. Indeed, it is consistent with converging evidence and theory in cognitive neuroscience [4, 8, 9, 10, 22, 31]. Toward the end of this chapter, we discuss categorical model theory and suggest that it indicates the desirability of feedback in a neural network. It also provides a mathematical foundation for perceptual knowledge representations.

As explained in Sect. 4.5, mathematical rigor is maintained in this formulation by describing structure-preserving mappings from **Concept** to a category representing the neural system's computational structure, thus showing how concepts, their morphisms, and the consequences of composition including commutative concept diagrams are represented in the system computations. A **Concept** morphism $s\colon T \longrightarrow T'$ is an association of the description constituting concept T with a subconcept, or logical part, of the description constituting concept T', with the property that the axioms of the domain T of the morphism are mapped to axioms or theorems of the codomain T'. We use an already-available mathematical convenience, a category of formal logic theories and theory morphisms ([7, 11, 23]) for the category **Concept**. The category has an overall hierarchical structure, since the more abstract theories are represented within the more specialized ones and the morphisms convey this. We use the terms "theory" and "concept" interchangeably.

As shown in the next section, colimits provide a means of combining concepts to derive concepts of greater complexity. Because they are more complex, these concepts are also more specific, or specialized, than any of the concepts in their diagrams. In the lattice formulation for geometric propositional logic, the lattice meet $\wedge$ operation plays an analogous role, since it forms a proposition of greater specificity than those being combined. However, because colimits are based upon diagrams in a category of theories, more can be expressed in this manner than with lattices. The same can be said for limits relative to the lattice join operation $\vee$. Limits form simpler, hence, more abstract, concepts than those in their base diagrams. But asserting that a *category* of theories is more expressive than a lattice begs the question of whether the theory category is not itself a lattice. After all, a lattice is a category, and in that sense a category is a sort of generalized lattice. How is the concept category not a lattice?

The key point is that a theory category, like most categories, has many "edges" between a given pair of "nodes"; that is, there can be many ways in

which an object a is related to an object b (and, in many categories, b can be related to a as well). This is a consequence of the complexity allowed in the morphisms, which is two-fold: First, a theory morphism must map all symbols (sorts, operations, constants) of the domain to symbols of the same type in the codomain. Second, each axiom of the domain theory, when transformed by symbol substitution, must become an axiom or theorem of the codomain theory.

Consider a concept of color stated as a theory, as follows:

```
Concept T
  sorts Colors, Items
  op has_color: Items*Colors -> Boolean
  const c: Colors
  Axiom color-is-expressed is
    exists (it: Items) (has_color (it, c))
end
```

The statement line `sorts Colors, Items` introduces the basic sorts, or "logical containers", of T; logic in this form is called *sorted*. The explicit typing of variables and constants in sorted theories such as T is a convenient alternative to using predicates to qualify the quantities in every formula, and it allows operations to be interpreted as total (as opposed to partial) functions whose domains are expressed in the theory as sorts. The line `op has_color: Items*Colors -> Boolean` specifies an operation `has_color`. This operation acts as a predicate with items and colors as arguments, where `Items*Colors` is a product sort—a "container" for ordered pairs of items and colors. The `Boolean` sort contains the truth values `true` and `false`. It is part of a theory of logical operations that is implicitly included in every concept (it is an initial object of the concept category). Notice that T also contains a constant `c` of sort `Colors`; this can represent, for example, a feature always observed via one sensor element of a system. The axiom `color-is-expressed` states that the color `c` has some item that expresses it (in a sorted predicate calculus, simply asserting a formula as shown is the same as stating that it is true). This color is arbitrary, having no definition, but as a constant it is regarded as a definite color nevertheless. In actuality, T does not have enough content to constitute a theory about color, but it will suffice for this simple example.

To show how such an abstract theory might be used, let there be morphisms $s^{(i)}: T \longrightarrow T^{(i)}\,(i \in \{1, 2, 3\})$, where T', T'' and $T^{(3)}$ are concepts representing three specific colors, `red`, `yellow` and `green`. For example:

```
Concept T'
  sorts Colors, Items
  op has_color: Items*Colors -> Boolean
  const red: Colors
  Axiom red-is-expressed is
    exists (it: Items) (has_color (it,red))
end
```

Theories T'' and $T^{(3)}$ are identical except that `red` is replaced by `yellow` and `green`, respectively. The morphism $s'\colon T \longrightarrow T'$ has the form

Morphism s': `Colors` $\mapsto$ `Colors`
`Items` $\mapsto$ `Items`
`has_color` $\mapsto$ `has_color`
`c` $\mapsto$ `red`

Morphisms s'' and $s^{(3)}$ are identical except for a color-symbol change. The axiom re-naming (`red-is-expressed` replaces `color-is-expressed`) need not be included in stating a morphism, since axiom names are simply a cosmetic touch. Also, it is customary to omit symbols that remain unchanged when stating a morphism, in this case `Colors, Items` and `has_color`. Continuing, T', T'' and $T^{(3)}$ are the domains for morphisms $t^{(i)}\colon T^{(i)} \longrightarrow T^{(4)}$ ($i \in \{1, 2, 3\}$) which have a common codomain, a theory that uses the three colors:

```
Concept T(4)
  sorts Colors, Apples
  op has_color: Apples*Colors -> Boolean
  const red: Colors
  const yellow: Colors
  const green: Colors
  Axiom some-apple-colors is
    exists (x, y, z: Apples)
      (has_color (x,red))
      and (has_color (y, yellow))
      and (has_color (z, green))
end
```

The morphism $t'\colon T' \longrightarrow T^{(4)}$, for example, has the form:

Morphism t': `Items` $\mapsto$ `Apples`

(The mappings of `Colors` to `Colors`, `has_color` to `has_color`, and `red` to `red` are not shown, as is customary.) Notice that the axiom of each of the theories T', T'' and $T^{(3)}$, transformed by the symbol substitution from t', t'' and $t^{(3)}$, respectively, maps to a theorem of $T^{(4)}$: For example, `exists (x: Apples) (has_color (x,red))` is an immediate consequence of the axiom `some-apple-colors` of $T^{(4)}$.

Finally, the compositions $t' \circ s'$, $t'' \circ s''$ and $t^{(3)} \circ s^{(3)}$ are three distinct morphisms with domain T and codomain $T^{(4)}$. For example, $t' \circ s'\colon T \longrightarrow T^{(4)}$ is as follows, by tracing the symbol mappings:

Morphism $t' \circ s'$: `Items` $\mapsto$ `Apples`
`c` $\mapsto$ `red`

Therefore, the category **Concept** is not a lattice. Of course, there would have been only one morphism from T to $T^{(4)}$ had T not included the constant

`c: Color`. However, most theories are much more complex than the ones in this example, increasing the possibilities for morphisms.

Another distinguishing feature of a category of theories is that colimits and limits are more expressive than joins and meets, for they express the contents of their base diagrams, not just the concepts. For example, colimits "paste together" or "blend" the concepts in a diagram along shared concepts as indicated in the diagram morphisms.

4.4 Colimits — An Example

The concept of a triangle can be derived as a colimit for a diagram involving concepts about points and lines (the alternative of defining triangles in terms of angles would complicate the example). We can start with a rather abstract concept, a theory that defines lines in terms of undefined quantities, or primitives, called points. The definition is expressed using a predicate `on` with two arguments, a point and a line, and is true just in case the point "lies on" the line (see [5] for a discussion of geometries based upon this definition).

```
Concept T1
  sorts Points, Lines
  const p1: Points
  const p2: Points
  const p3: Points
  op on: Points*Lines -> Boolean
  Axiom Two-points-define-a-line is
    forall(x, y:Points)
      ((x not= y) implies
        (exists L:Lines)
          (on (x, L) and on (y, L) and
            ((forall m:lines) (on (x, m) and
              on (y, m)) implies (m = L) ))
end
```

Notice that T_1 also contains constants representing three arbitrary points `p1`, `p2` and `p3`. Three other concepts T_2, T_3 and T_4 share T_1 except with different names for the point constants in each. The latter concepts also include a line constant and associate two of the point constants with it via the `on` predicate. This is specified with an additional axiom (name omitted) which also states that the two point constants denote distinct points. For example:

```
Concept T2
  sorts Points, Lines
  const pa1: Points
  const pa2: Points
  const paext: Points
  const La: Lines
```

```
  op on: Points*Lines --> Boolean
  Axiom Two-points-define-a-line is
    forall(x, y:Points)
      ((x not= y) implies
        (exists L:Lines)
          (on (x, L) and on (y, L) and
            ((forall m:lines) (on (x, m) and
              on (y, m)) implies (m = L) ))
  on (pa1, La) and on (pa2, La)
    and (pa1 not= pa2)
end
```

Concepts T_3 and T_4 are identical, except with the names `pa1`, `pa2`, `paext`, and `La` replaced with `pb1`, `pb2`, `pbext`, `Lb` in T_3 and `pc1`, `pc2`, `pcext`, `Lc` in T_4. A morphism $s_1 : T_1 \longrightarrow T_2$ maps the sort symbols `Points` and `Lines` and the `on` predicate symbol to the corresponding symbols in T_2, which happen to be identical. We reformulate all statements of T_1 by term replacement in accordance with the symbol mapping to form their image statements in T_2. As a consequence, the axiom of T_1 relating points to lines maps to itself as an axiom of T_2. The point constants `p1`, `p2` and `p3` map to the point constants `pa1`, `pa2` and `paext`. In T_2, `pa1` and `pa2` are associated with the line `La` via the `on` predicate, and `paext` is intended as a point "external to" `La`.

Morphism s_1 : `p1` $\mapsto$ `pa1`
`p2` $\mapsto$ `pa2`
`p3` $\mapsto$ `paext`

Morphisms $s_2 : T_1 \longrightarrow T_3$ and $s_3 : T_1 \longrightarrow T_4$ are similar to s_1 but with different point constant targets `pb1`, `pb2`, `pbext` and `pc1`, `pc2`, `pcext`:

Morphism s_2 : `p1` $\mapsto$ `pbext`
`p2` $\mapsto$ `pb1`
`p3` $\mapsto$ `pb2`

Morphism s_3 : `p1` $\mapsto$ `pc2`
`p2` $\mapsto$ `pcext`
`p3` $\mapsto$ `pc1`

In T_3, it is the images `pb1` of `p2` and `pb2` of `p3` that are associated with the line constant, `lb`, while the image `pbext` of `p1` is the "external" point. The associations are similarly reordered in T_4; the point-to-line associations in each concept can be seen by noticing which points are the targets of those of T_1 under the appropriate morphism and applying term substitution. A colimit for the diagram Δ with objects T_1, T_2, T_3, T_4 and morphisms s_1, s_2, s_3 (which always exists in the category **Concept**) has a cocone as shown in Fig. 4.2, with apical object T_5 and leg morphisms $\ell_1 : T_1 \longrightarrow T_5$, $\ell_2 : T_2 \longrightarrow T_5$, $\ell_3 : T_3 \longrightarrow T_5$, and $\ell_4 : T_4 \longrightarrow T_5$. With Δ as the base diagram, the defining diagram of the colimit, $\overline{\Delta}$, is commutative, with

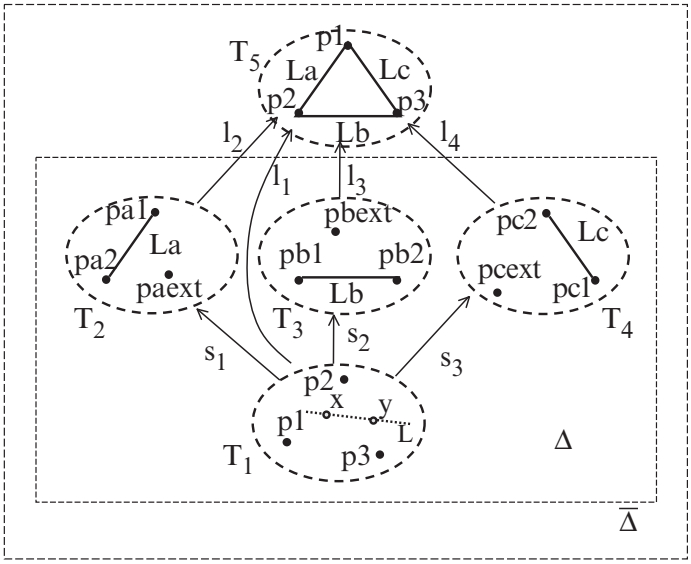

Fig. 4.2. A pictorial illustration of the colimit base and defining diagrams Δ and $\overline{\Delta}$. The contents of the concepts involved are pictured along with the diagrammatic structure. Solid dots and lines signify point and line constants. Concept T_1 has no line constant, so it is shown containing a dashed line with two open dots, representing the axiom relating points and lines which is present in all the concepts

$$\ell_1 = \ell_2 \circ s_1 = \ell_3 \circ s_2 = \ell_4 \circ s_3 \,. \tag{4.2}$$

The resulting colimit object, T_5, is as follows:

```
Concept T5
  sorts Points, Lines
  const p1: Points
  const p2: Points
  const p3: Points
  const La: Lines
  const Lb: Lines
  const Lc: Lines
  op on: Points*Lines -> Boolean
  Axiom Two-points-define-a-line is
    forall(x, y:Points)
      ((x not= y) implies
        (exists L:Lines)
          (on (x, L) and on (y, L) and
            ((forall m:lines) (on (x, m) and
              on (y, m)) implies (m = L) ))
```

```
  on (p1, La) and on (p2, La) and
    (p1 not= p2)
  on (p2, Lb) and on (p3, Lb) and
    (p2 not= p3)
  on (p3, Lc) and on (p1, Lc) and
    (p3 not= p1)
end
```

As a consequence of the commutativity of the defining diagram $\overline{\Delta}$ of the colimit, its apical concept T_5 is a "blending" or "pasting together" of T_2, T_3 and T_4 along their common sub-concept T_1. That is, for the equality (4.2) to hold, separate symbols of T_2, T_3 and T_4 that are images of the same symbol of T_1 under the three diagram Δ morphisms s_1, s_2 and s_3 must merge into a single symbol in the colimit apical concept T_5. To make this clear, each symbol in T_5 that is a merging of symbols has been assigned the name of the T_1 symbol underlying the merging. Thus, symbols such as `Points`, `Lines` and `on` appear in T_5, and appear only once, since they are mapped to themselves by each of the morphisms s_1, s_2 and s_3. The point constants `p1`, `p2`, `p3` also appear. However, in T_5, each one represents a merging of two point constants from T_2, T_3 and T_4 and as a consequence appears in the definition of two different lines. In two of these concepts, the image of each single point constant appears in the definition of a line, but as a different point on a different line in each of the two concepts. In the third concept, it appears as an "external" point, not on the line named in that concept. For example, `p1` in T_1 is mapped to `pa1` in T_2 via s_1, to `pbext` in T_3 via s_2, and to `pc2` in T_4 via s_3. In T_5, therefore, it forms the point `p1` at the intersections of lines `La` and `Lc`, and lies external to line `Lb`. Because of the *initiality* of the colimit cocone, any other cocone for Δ is the codomain of a unique cocone morphism whose domain is the cocone containing T_5. Therefore, T_5 adds no extraneous information to that in Δ, and any other apical concept of a cocone for Δ is either isomorphic with T_5 or extends it without duplicating it.

Because it is more complex, the colimit apical object is more specific, or more specialized, than any of the concepts in its base diagram. Because it expresses the "pasting together" of the base diagram concepts around their shared sub-concepts, it expresses the concept relationships (morphisms) as well as the concepts in its base diagram. Because of the Colimit Theorem, the calculation of concept colimits can be automated. For example, the apical object T_5 and leg morphisms ℓ_1, ℓ_2, ℓ_3, and ℓ_4 above can be derived automatically from the objects and morphisms of the base diagram, Δ. Other examples are given in [32] for an application of engineering software synthesis via category theory. Where they are available, limits can be calculated also, yielding less complex or abstract theories and their attendant morphisms. This facilitates the generalization of learned concepts to new contexts by extracting useful invariants from them.

The ability to calculate concept colimits and limits suggests the ability to "flesh out" an ontology in an incremental fashion. This process can begin with a collection of concepts and morphisms describing the most basic properties of observable quantities and also any desired assumptions about the environment and the operation of a system within it. More specialized theories can be calculated as colimits, and more abstract ones as limits, through re-use of pre-existing concepts and morphisms. Taken together, the discussions in this section and the previous one suggest the use of category theory in the study of knowledge systems, learning, and the semantics of distributed, adaptive systems such as neural networks. Conducting these studies while utilizing the mathematical rigor of category theory requires the availability of categories that express the system computations together with structure-preserving mappings from **Concept** to these categories. Any such mappings must be capable of conveying only part of the ontology expressed in **Concept**, for two reasons: (1) **Concept** is infinite and realizable systems are finite, and (2) in an adaptive system, only a part of the ontology will have been learned at any one time. Another issue arises in systems with multiple sources of inputs working in parallel, such as neural networks with more than one sensor. Simultaneous inputs from the different sensors obtain different types of information about the same events. Fusing the information across sensors requires a multi-component system that can make several knowledge representations act as one.

4.5 Structural Mappings and Systems

A *functor* $F : C \longrightarrow D$, with domain category C and codomain category D, associates to each object a of C a unique image object $F(a)$ of D and to each morphism $f : a \longrightarrow b$ of C a unique morphism $F(f) : F(a) \longrightarrow F(b)$ of D. Moreover, F preserves the compositional structure of C, as follows. Let $\circ_C$ and $\circ_D$ denote the separate composition operations in categories C and D, respectively. For each composition $g \circ_C f$ defined for morphisms of C, $F(g \circ_C f) = F(g) \circ_D F(f)$, and for each identity morphism of C, $F(\mathtt{id}_a) = \mathtt{id}_{F(a)}$. It follows that the images of the objects and morphisms in a commutative diagram of C form a commutative diagram in D. This means that any structural constraints expressed in C are translated into D and, hence, F is a structure-preserving mapping. Functors can be many-to-one, and by this means much of the structure of the domain category of a functor can be "compressed", so that not all the structure is represented explicitly in the codomain. Functors can also be *into* mappings, so that the codomain need not be entirely utilized in representing the domain; this leaves room for a functor to be used in defining an extension of a given category, or in representing a structure in the context of one with a greater complexity.

We have proposed elsewhere (for example, [15, 16]) that a neural network at a given stage of learning can be associated with a category $\mathbf{N}_{A,w}$ that expresses both its connectivity and potential state changes in response to its

next input. Here, A represents the architectural design including dynamic properties (the dynamics are represented only in summary fashion as rules for state changes), while w is the current connection-weight array. An analysis of the network determines whether it is possible to define a functor $M\colon \mathbf{Concept} \longrightarrow \mathbf{N}_{A,w}$. This allows the knowledge-representation capabilities of an existing architecture to be evaluated with mathematical rigor, and can lead to insights for possible design improvements or for the design of entirely new architectures. Since functors are many-to-one mappings, concepts which have not been acquired at some stage of learning or simply cannot be represented by a given neural network can be included in the analysis.

A further advantage of the categorical approach is the notion of *natural transformations*, of the form $\alpha : F \longrightarrow G$ with domain functor $F : C \longrightarrow D$ and codomain functor $G : C \longrightarrow D$. A natural transformation α consists of a system of D-morphisms α_a, one for each object a of C, such that the diagram in D shown in Fig. 4.3 commutes for each morphism $f\colon a \longrightarrow b$ of C. That is, the morphisms $G(f) \circ \alpha_a : F(a) \longrightarrow G(b)$ and $\alpha_b \circ F(f) : F(a) \longrightarrow G(b)$ are actually one and the same, $G(f) \circ \alpha_a = \alpha_b \circ F(f)$. In a sense, the two functors have their morphism images $F(f)\colon F(a) \longrightarrow F(b)$, $G(f)\colon G(a) \longrightarrow G(b)$ "stitched together" by other morphisms α_a, α_b existing in D, indexed by the objects of C. In a multi-sensor system, with each sensor having its own dedicated processing network and the networks connected into a larger system, natural transformations express *knowledge coherence*. They accomplish this by unifying the knowledge representations of the subnetworks of a neural network, which can include subnetworks for sensor association, planning and other functions (see [14] for an example network). Figure 4.4 illustrates this notion for two functors $M_1\colon \mathbf{Concept} \longrightarrow \mathbf{N}_{A,w}$ and $M_2\colon \mathbf{Concept} \longrightarrow \mathbf{N}_{A,w}$ representing sensor-specific processing knowledge and $M_3\colon \mathbf{Concept} \longrightarrow \mathbf{N}_{A,w}$ representing knowledge fusion in an association region of a multi-regional network. Natural transformations $\gamma_1\colon M_1 \longrightarrow M_3$ and $\gamma_2\colon M_2 \longrightarrow M_3$ provide knowledge coherence between the separate knowledge representations.

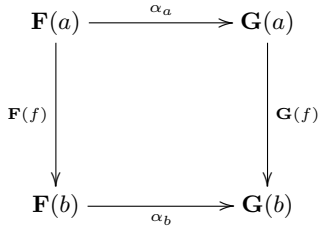

Fig. 4.3. A commutative diagram associated with a natural transformation. The morphisms $G(f) \circ \alpha_a : F(a) \longrightarrow G(b)$ and $\alpha_b \circ F(f) : F(a) \longrightarrow G(b)$ are one and the same, $G(f) \circ \alpha_a = \alpha_b \circ F(f)$

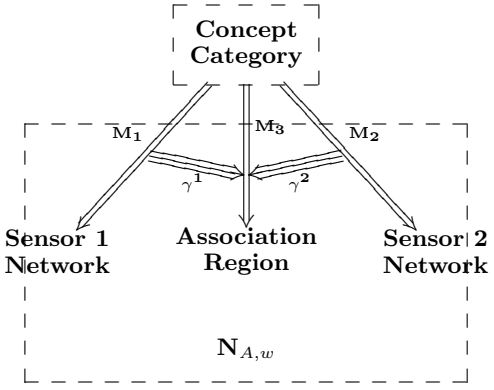

Fig. 4.4. Functors map the hierarchy of a concept category to multiple regions. Natural transformations represent coherent interconnections between hierarchy representations

4.6 From Points to Models

In the topological-lattice account of [12, 29, 30], the instances of propositions are referred to as *points*. This is suggestive, for the semantics of propositional geometric logic is analyzed in terms of topological systems (similar to the spaces of point-set topology) and continuous functions. However, as mentioned in the aforementioned papers by Vickers, category theory is required for expressing the semantics of the geometric predicate calculus. In this context, the collection of points associated with a predicate formula is a space of models, or possible worlds in which the predicate has a valid interpretation. Category theory is required to express the internal structure of theories, and their model spaces are also categories. The semantic theory presented here replaces formulas with whole theories, with functors associating theories and their morphism with objects and morphisms of a neural category . Here, the models are possible worlds or situations in which a theory is valid.

Each concept morphism $s\colon T \longrightarrow T'$ has an associated *model-space morphism*, a functor $\texttt{Mod}(s)\colon \texttt{Mod}(T') \longrightarrow \texttt{Mod}(T)$. Here, $\texttt{Mod}(T)$ and $\texttt{Mod}(T')$ are the model categories for T and T', respectively. Since $\texttt{Mod}(s)$ reverses the direction of s, each instance of T' has a corresponding instance of T. *This fact has great significance for knowledge representation.* In particular, it suggests a design principle for neural networks, as follows: Suppose that neural category objects $M(T)$ and $M(T')$ are the images of objects (concepts) T and T' under a functor $M\colon \textbf{Concept} \longrightarrow \mathbf{N}_{A,w}$, and that $M(s)\colon M(T) \longrightarrow M(T')$ is the image of a concept morphism $s\colon T \longrightarrow T'$. We associate the activating inputs for the objects $M(T)$ and $M(T')$ with objects in the model categories $\texttt{Mod}(T)$ and $\texttt{Mod}(T')$, respectively. Given this association, *every input that activates* $M(T')$ *must also activate* $M(T)$, a consequence of the existence of the model-space morphism $\texttt{Mod}(s)\colon \texttt{Mod}(T') \longrightarrow$

$\texttt{Mod}(T)$. This provides a mathematical justification—in fact, an imperative—for the presence of feedback in neural networks. Further, this principle applies in some appropriate form to the design of any knowledge representation system.

The model-space morphism principle has important implications for limit and colimit representations. For example, let T be the apical concept of a limit cone for a diagram Δ in **Concept** and let $\ell\colon T \longrightarrow T'$ be one of the leg morphisms for the limit cone, where T' is an object in the base diagram Δ. Then, $M(T')$ is an object in the image diagram $M(\Delta)$, and the model-space morphism principle dictates that the image $M(T)$ of the limit apical object *must be activated through* $M(\ell)$ *whenever* $M(T')$ *is active*. The reverse is true for concept colimits: *Every instance of the functorial image of a colimit apical object must also be an instance of the objects in the image of its base diagram*. One consequence is that the analyst can detect limit and colimit representations in a given neural network, and distinguish between them.

4.7 Conclusion

We express the semantics of a system in terms of a distributed system of concepts. The system is an ontology for the system's environment as experienced through the system inputs, both external to the system and possibly internally-generated as well. The ontology is expressed mathematically as a category **Concept** whose objects are symbolic descriptions (concepts) of a domain of items (sensed entities, events, situations, and their parts). We have discussed a sense in which this category is a kind of generalized lattice, differing from a lattice in having morphisms, which are more expressive than edges, with a consequent multiplicity of relationships between objects and with diagrammatic constructions which provide automated concept derivations useful in expressing learning. Two modes of learning can be expressed, specialization and abstraction. In a system such as a neural network with incremental knowledge gain, these two modes serve to "fill out" a representation of an ontology for the environment and functionality of the system, beginning with basic knowledge about inputs. Mathematical rigor is maintained through structure-preserving mappings from **Concept** to a category representing the system's computational structure, thus showing how concepts, their morphisms, and the consequences of composition including commutative concept diagrams are represented in the system computations. Natural transformations formalize knowledge coherence, the unified operation of the separate knowledge representations in the different subnetworks of a multi-regional architecture. Another system of structure-preserving mappings formalizes the relationship between possible worlds for the concept representations. The semantic theory thus offers a comprehensive mathematical theory to support investigations in concept learning in adaptive, distributed systems.

References

1. Adamek J, Herrlich H, Strecker G (1990) Abstract and Concrete Categories. Cambridge University Press, Cambridge New York
2. Andrews R, Diederich J, Tickle A (1995) Survey and critique of techniques for extracting rules from trained artificial neural networks. Knowledge-Based Systems 8:373–389
3. Arbib M (1987) Brains, Machines, and Mathematics. Springer, Berlin Heidelberg New York
4. Barsalou L (1999) Perceptual symbol systems. Behavioral and Brain Sciences 22:577–660
5. Bennet M (1995) Affine and Projective Geometry. John Wiley and Sons, New York
6. Craven M, Shavlik J (1993) Learning symbolic rules using artificial neural networks. In: Proc 10th Intl Machine Learning Conference pp 73–80
7. Crole R (1993) Categories for Types. Cambridge University Press, Cambridge
8. Damasio A (1989) Time-locked multiregional retroactivation: a systems-level proposal for the neural substrates of recall and recognition. Cognition 33:25–62
9. Damasio A (1999) Decartes' Error. Putnam, New York
10. Eichenbaum H (2004) Hippocampus: cognitive processes and neural representations that underlie declarative memory. Neuron 44:109–120
11. Goguen J, Burstall R (1992) Institutions: abstract model theory for specification and programming. J Assoc Computing Machinery 39:95–146
12. Healy M (1999) A topological semantics for rule extraction with neural networks. Connection Science 11:91–113
13. Healy M, Caudell T (1997) Acquiring rule sets as a product of learning in a logical neural architecture. IEEE Trans Neural Networks 8:461–474
14. Healy M, Caudell T (2003) From categorical semantics to neural network design. In: Proc Intl Joint Conf Neural Networks pp 1981–1986
15. Healy M, Olinger R, Young R, Caudell T, Larson K (2005) Modification of the ART-1 architecture based on category theoretic design principles. In: Proc Intl Joint Conf Neural Networks pp 457–462
16. Healy M, Caudell T (2006) Ontologies and worlds in category theory: implications for neural systems. Axiomathes 16:165–214
17. Heileman G, Georgiopoulos M, Healy M, Verzi S (1997) The generalization capabilities of ARTMAP. In: Proc Intl Conf Neural Networks (ICNN)
18. Kaburlasos V, Petridis V (2000) Fuzzy lattice neurocomputing (FLN) models. Neural Networks 13:1145–1170
19. Kasabov N (1996) Adaptable neuro production systems. Neurocomputing 13:95–117
20. Lawvere F, Schanuel S (1997) Conceptual Mathematics: A First Introduction to Categories. Cambridge University Press, Cambridge
21. Mac Lane S (1971) Categories for the Working Mathematician. Springer, Berlin Heidelberg New York
22. McClelland J, McNaughton B, O'Reilly R (1995) Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. Psychological Review 102:419–457
23. Meseguer J (1989) General logics. In: Ebbinghaus H-D, et al (eds) Logic Colloquium '87. Science Publishers B V, North-Holland

24. Mitra S, Pal S (1995) Fuzzy multi-layer perceptron, inferencing and rule generation. IEEE Trans Neural Networks 6:51–63
25. Pierce B (1991) Basic Category Theory for Computer Scientists. MIT Press, Cambridge, Mass London
26. Pinkas G (1995) Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. Artificial Intelligence 77:203–247
27. Sima J (1995) Neural expert systems. Neural Networks 8:261–271
28. Sussner P (2003) Generalizing operations of binary autoassociative morphological memories using fuzzy set theory. J Math Imaging Vision 19:81–93
29. Vickers S (1992) Geometric theories and databases. In: Fourman M, et al (eds) Applications of Categories in Computer Science, Proc LMS Symp Lec Note Ser 177. Cambridge University Press, Cambridge
30. Vickers S (1993) Topology via Logic. Cambridge University Press, Cambridge
31. Wickelgren I (1997) Getting a grasp on working memory. Science 275:1580–1582
32. Williamson K, Healy M, Barker R (2001) Industrial applications of software synthesis via category theory-case studies using specware. Automated Software Eng 8:7–30