# Knowledge Representation in Neural Networks

December 5, 2013

## 1 Introduction

Recall the earlier definition of **intelligence** as *doing the right thing at the right time, as judged by an outside human observer*. As a key facilitator of intelligence, **knowledge** can then be defined as *background information or general understanding (of a variety of domains) that enhances the ability to act intelligently*, where domains include the natural, social, and even virtual worlds, as well as mathematics, music, art, etc. A **knowledge representation** is an encoding of this information or understanding in a particular substrate, such as a set of if-then rules, a semantic network, conditional probability tables, a Venn diagram, a mind map, or the axioms of formal logic. Thus, patterns (a.k.a. relationships among primitive elements) within the substrate correspond to patterns in the target domain. The relationships between the patterns (e.g., how each promotes or inhibits others) are key determinants of intelligence.

Neural networks (both natural and artificial) are the focal substrate of this chapter, with the key question being how patterns of both neural connectivity and activity encode knowledge. Many of the substrates listed above have such strong roots in natural language that their knowledge content is completely transparent. For example, we can easily translate a mind map or a set of logical axioms into an explicit natural language description of knowledge content, such as *the combination of air masses from Canada, the desert Southwest and the Gulf of Mexico facilitate tornado formation in the Midwest*.

Unfortunately, deciphering the knowledge content of a neural network requires much more work; in many cases, the salient information does not map nicely to natural language, or, when it does, the apparent *concepts* of import lack the crisp definitions preferred by philosophers and GOFAI knowledge engineers. Instead, a neural pattern might encode a large, and vague, set of preconditions that embodies a complex definition with a slew of exceptions. The interactions between these patterns produce highly intelligent behavior, but reduction of that behavior to the primitive patterns fails to produce the normal satisfaction that one gets from, for example, decomposing the description of a play into its characters, plot, conflicts, resolutions, etc.

A good deal of philosophical quicksand surrounds the concept of knowledge, with distinctions often drawn between simple pieces of data, such as a random 10-digit number, and data that *connects* to other information, such as a 10-digit number that happens to be your brother's phone number, or the population size of China. The data acquires significance or *meaning* via these connections; and only via these links can the data contribute to reproducible behavior. Hence knowledge, as a key contributor to *doing the right thing*, needs these ties to distinguish it from untethered (and essentially meaningless) data. Furthermore, *doing* may constitute actual physical or subconscious mental activity - in which case the knowledge is often termed *procedural* - or explicit cognitive events - wherein the knowledge is *declarative*.

In a physical system (e.g. animal, robot, smart room, etc.) controlled by a neural network, patterns of neural activity acquire meaning (a.k.a. semantics) via connections to the external world, the system's physical body, or other

neural activity patterns. These connections, manifest as temporal correlations that the brain becomes wired to sustain, facilitate recreation or *re-presentation*, since the occurrence of one pattern helps invoke another. Causality (in terms of causes preceding effects) plays no specific role: a sensory situation may precede the neural pattern that represents it, just as a neural pattern may precede a body's motor sequence. Alternatively, a neural pattern may predict an impending sensory experience; or the pattern, experience and motor sequence may co-form as part of the looping brain-body-world interaction. Correlation, not purported causality, is crucial; and patterns lacking it may have short-term behavioral significance but little representational import. This correlational view of representation paves the way for formal tools, such as information theory, to help clarify network functionality.

Intertwined with this level of representation is another: synaptic networks and their ability to stimulate neural interactions. The complex matrix of synaptic strengths forms attractors in the landscape of neural activity patterns such that some patterns (and temporal sequences of patterns) become much more probable than others. Thus, the synapses represent the firing patterns, which represent knowledge. Since neural activity affects synaptic strength via learning, the relationships quickly become circular, thus precluding a straightforward linear interpretation of this representational hierarchy. However, the review of each factor in (relative) isolation has obvious explanatory merit, so this chapter treats each of these representation forms separately.

## 2 Representations in Firing Patterns

Any thorough discussion of firing patterns as representations begins with the atomic components of a pattern, which, like so many other aspects of intelligence, remain controversial. At the level of an individual neuron, there is considerable debate as to the fundamental carrier of information, with the two main candidates being the firing time of a neuron versus its overall firing rate, as thoroughly explained in [10].

Firing-rate coding implies that the information encoded in a neuron is due to the average number of spikes that it emits per time period, e.g. per second. Spike-timing code does not refer to the absolute time point when a neuron spikes, rather a relative temporal delay: the phase (of some ambient oscillation) at which a neuron spikes (or possibly begins to emit spikes). These background oscillations stem from a large population of neurons that simultaneously fire, thus exciting or inhibiting downstream neighbors. More generally, a spike-timing code looks at a spike train starting at some reference time, 0, (e.g. the peak of a particular background oscillation) and attributes information to the exact times (after 0) at which spikes occur, whereas a rate code only counts the total number of spikes and divides by the duration of the spike train.

Instead of delving deeply into this debate, it suffices for our current purposes to view the two codes as more-or-less equivalent, using a slightly simplified version of the argument in [9]. Consider the situation in Figure 1 in which two neurons, 1 and 2, are affected by the same inhibitory oscillation (drawn as a sine curve in the figure), but due to different levels of excitation, neuron 1 is able to overcome inhibition and begin emitting spikes at a phase (P1) when inhibition is higher than during phase P2, where neuron 2 overcomes inhibition. A similar, but inverse, argument holds for an excitatory oscillating input.

The dashed lines indicate time windows during which the neuron's excitation dominates inhibition, and thus the neuron emits spikes. From the figure, it is clear that neuron 1 is emitting spikes over larger windows (A and B), than is neuron 2 (with windows C and D). Thus, if we average over the complete time window (here composed of 3 inhibitory cycles), neuron 1 would be spiking more often ($A + B > C + D$) and would thus have a higher calculated firing rate than neuron 2. This appears to be a relatively monotonic progression in that the higher up on the inhibitory curve a neuron begins to fire (i.e. the phase at which it fires), the higher will be its firing rate.

So in this fairly simple sense, the spike-time code (expressed as the phase at which the neuron overcomes inhibition) correlates with the rate code. So although spike timing appears to play a key role in synaptic tuning via STDP (as explained earlier), the useful information in a relative spike time seems no different (nor more significant) than that

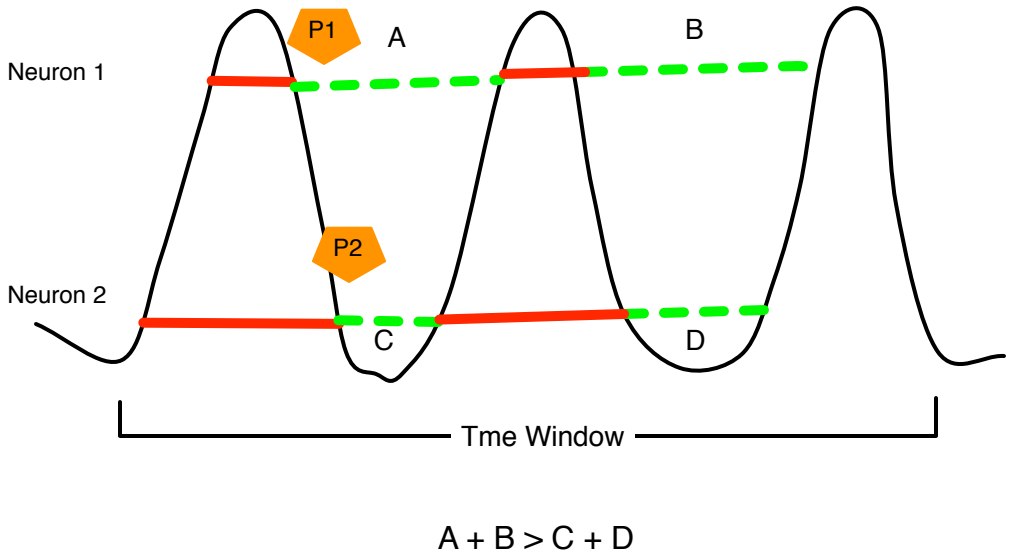


Figure 1: Illustration of two neurons that begin firing (i.e. overcome inhibition) at different phases of the inhibitory cycle.

encoded by a firing rate. Though this explanation certainly does not resolve the debate (and surely irks those who champion the importance of spike timing), it justifies ignoring it for the time being and simplifying the analysis of neural representation to that of the collective firing rates of neurons. Furthermore, I will follow a good deal of the neuroscience and connectionism literature by abstracting the state of any neuron to a binary, on-off, value. Since neurons are never completely dormant, the distinction between on and off is typically characterized by being above or below some *normal* firing rate, respectively.

## 2.1 The Syntax of Neural Information

Given a population of neurons, each of whose individual information content is now assumed to be a single bit (on or off), the next important issue is how this collection represents a particular piece of information or concept C (using the term very loosely). Does the network encode C using one or just a few neurons, or does it employs hundreds, thousands or even millions of them? This is the distinction between *local* and *distributed* coding.

In their extreme forms, they dictate that if a collection of n neurons are used to store information, then a local coding scheme would use each of the n cells to represent a single concept, while the distributed code would use a sizeable fraction of all n neurons (and the synapses between them) to represent **every** concept. Thus, in the latter case, each concept is distributed over many nodes, and an entire suite of concepts *shares space* in the neurons and synapses of a single network.

A further distinction between sparsely- and fully-distributed codes is also commonly drawn. A sparse code uses only a few of the nodes, per concept, and is thus *semi-local* (another frequently-used term), while a fully distributed scheme uses a large number of neurons per concept, with $\frac{n}{2}$ being the optimal number for representing the most concepts.

To understand this optimality, consider that the number of different concepts using exactly k nodes (i.e. k neurons are on for each concept) that can be represented in an n-node network is $\binom{n}{k}$, which has a maximum at $k = \frac{n}{2}$.

Figure 2.1 portrays these three representational forms: local, semi-local/sparsely distributed and fully distributed. Cells that perform local coding are often refered to as *grandmother cells*, meaning that individual cells represent specific concepts such as bicycle, ice cream or grandmother.

The number of distinct patterns that can (theoretically) be stored in an n-neuron network (for n = 20) is summarized below:

- Local: $\binom{n}{1} = n$ patterns
- Sparsely Distributed: $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ patterns. n = 20, k= 3 $\rightarrow$ 1140 patterns
- Fully Distributed: $\binom{n}{\frac{n}{2}} = \frac{n!}{(\frac{n}{2}!)^2}$ patterns. n = 20 $\rightarrow$ 184756 patterns

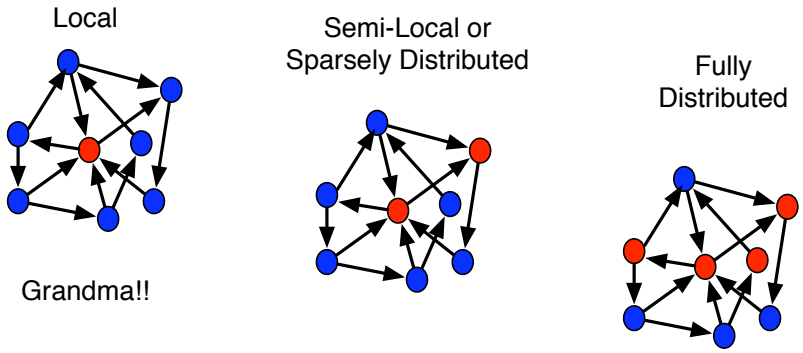


Figure 2: Rough sketch of three general coding schemes used in neural networks. Red neurons are considered active (and blue inactive) when the concept is being processed.

As discussed below, these capacities derive purely from the nature of binary codes. For distributed representations, the theoretical values are several orders of magnitude greater than the practical storage limits of neural networks.

## 2.2 Pattern Completion in Distributed Memories

The mathematics of distributed representations always sounds so impressive. In fact, one could load all 184756 patterns into a 20-node network, and get a unique signature of on/off neurons for each. But most of them could never be unambiguously retrieved from the network based on partial information. So you could put all 184756 patterns in but only get a few of them out.

Distributed memories typically operate in an *associative, content addressable* manner, meaning that memories are indexed and cued by portions of themselves (or of other, related memories). Retrieval works by presenting the network with a partial pattern or cue, which is then completed via the exchange of signals between the nodes (as in the Hopfield networks described in a later chapter). Memory retrieval is simply distributed pattern completion.

So, indeed, with n bits, one can represent $2^n$ different patterns - but not at the same time nor in the same place! In an n-node, fully connected, distributed-coded neural network, multiple patterns need to be stored across the $\frac{n(n-1)}{2}$ weights such that they can be retrieved and displayed on the same set of n neurons. The presence of a quadratic number of weights does help, but $\frac{n(n-1)}{2}$ is still much less than $2^n$ for n larger than 15 or 20, so the naive representational promise of distributed coding is completely unrealistic when the $2^n$ (or any significant fraction thereof) patterns must share

space. To get a quantitative feel for the ubiquity of pattern interference in distributed memories, see the discussion of k-m predictability in Appendix (appendix label).

### 2.2.1 Sequential Pattern Retrieval

In a single-pattern completion task, a partial pattern cues the rest of the pattern. In sequential pattern retrieval, one complete pattern cues the next pattern, which, in turn, cues the next pattern. In theory, the network can complete the entire sequence when given only the first pattern.

Sequence retrieval also illustrates the differences between local and distributed coding. In the former, each sequence element corresponds to a single neuron, which provides an excitatory link to the neuron representing the next item in the sequence. The key advantage of this representation is that n linearly-interlinked concepts are easily incorporated into an n-neuron network with only n-1 links. The key disadvantage is fault tolerance: if any of the neurons, n*, in the sequence is damaged, all concepts represented by downstream neurons from n* cannot be cued for recall and are thus inaccessible.

Conversely, for a sequence of items stored in a distributed rmanner, a faulty neuron may have little adverse effect, since each concept is coded by a combination of many neurons. Unfortunately, to pack all of those distributed codes into the same network can quickly lead to ambiguities during retrieval, due to overlap (i.e. interference) between the patterns. Thus, a single concept would have several, not one unique, successor pattern.

## 2.3 Connecting Patterns: The Semantics of Neural Information

The most intuitive way to add meaning to neural patterns is to ground them in reality by finding correlations between external events (such as the lightning bolts of Figure 3) and combinations of active neurons, termed *cell assemblies* by Donald Hebb [5]. These neural activity patterns can later serve as *stand ins* or *representations* for the actual event. These correlations arise by learning and may require several exposures to similar events before they consistently evoke the same activity pattern.

Sameness is an elusive relationship in these situations; for mammals and their large brains, two neural states at two different times are surely never identical due to the ever-changing nature of complex natural neural networks. However, two states can probably be very similar, and more similar to each other than to any other states, for example. To further complicate the issue, the complete brain state - which takes into account all of the brain's neurons - reflects many more sensory and proprioceptive factors than, for example, the sight of a lightning bolt. Whether one sees lightning while sitting indoors versus running down the street will surely impact the complete mental state. Even though the lightning concept could evoke the same cell assembly in both cases, the complete brain states could differ significantly. For example, the neural effects of fear could be widespread in the running scenario but not in the indoor setting. Regardless of the unavoidable philosophical quicksand related to mental states and their meanings, the general fact that certain mental states do seem to correlate with particular sensory experiences provides some solid footing for explorations into the pattern-processing functions of the brain.

Once neural correlates of world events exist, they can become linked to one another via synaptic ties between the respective cell assemblies, as roughly depicted in Figure 4. This allows the presence of one event, such as lightning, to evoke the thought of another event, e.g., thunder, prior to or in the complete absence of that event.

When cell assemblies include many intra-group synapses among the neurons, pattern completion becomes possible. Thus, as shown in Figure 5, when a partial sensory experience, such as an occluded view of a lightning cloud, stimulates part of the assembly, the internal synapses quickly activate the rest of the assembly, thus putting the brain into

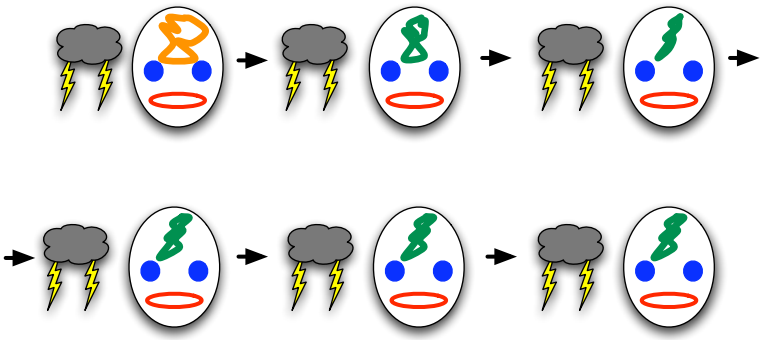

Figure 3: One of the brain's key functions is to form correlations between real-world events and brain states. Repeated exposures to the event may be required to form a consistent correlation, as shown by the sequence of action-brain-state pairs.

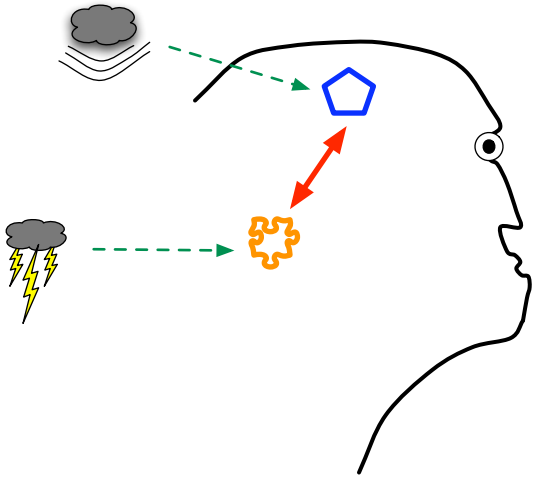

Figure 4: Another important brain function is to achieve correlations between brain states whose association has some utility for the organism. These states may or may not correlate with external events.

it's normal lightning-viewing state, from which it can predict normal lightning successors, such as thunder, and plan normal lightning-response actions, such as running for cover.

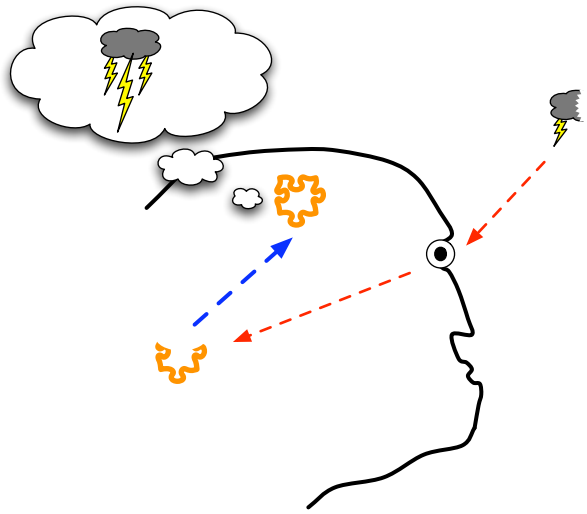

Figure 5: Another key function of brain is to complete partial patterns, thus putting the brain into a similar state to that achieved by a more complete sensory input, and allowing the brain to predict similar consequences and plan similar responses.

Although two events in the world are never truly identical, living organisms often need to *treat* them so. Otherwise, the appropriate responses learned from prior experience could never be reused in similar situations, since the organism would never recognize the similarity. To fully exploit the advantages of a learned association between a stimulus and a response, an organism must have the ability to generalize across stimuli. Thus, at some level of mental processing, two similar stimuli should evoke approximately the same cell assembly, which could then evoke the same predictions or actions associated with that general class of stimuli.

For example, if a small child learns that small, dark, short-haired dogs are dangerous (due to the unfortunate accident of stepping on one's leg), he may also begin to watch his step around small, light, long-haired dogs as well. The child thus exhibits a general cautious behavior around all small dogs based on one experience. So in terms of the sensory preconditions for that behavior, the child simplifies considerably, ignoring most of the significant differences between dog species, and mapping all small-dog experiences to a *tread lightly* behavior. In psychological or connectionist terminology, the child has *categorized* these animals into a group that most adult observers would call *small dogs*, and then the child chooses the same behavior in response to all members of that group. The categorization may be purely implicit, as only the child's actions reveal the underlying generalization; he cannot consciously contemplate nor discuss it with anyone.

Figure 6 depicts this general mechanism, known as *sparsification*, which has the character of a reductive process in that many different events or neural patterns reduce to a single pattern, thus requiring a sparser population of neurons in the downstream assembly to capture a given concept.

Conversely, the process of orthogonalization, shown in Figure 7, accentuates the differences between world events or upstream cell assemblies. Thus, even if two stimuli overlap significantly, the brain may map each to a different response. Much of the detailed knowledge of expertise would seem to involve orthogonalization, as small differences in problem contexts may invoke vastly diverse actions by an expert, who notices salient discrepencies that a novice might overlook. In the case of the canine-wary child, orthogonalization means differentiating small dogs that are playful from those that are angry.

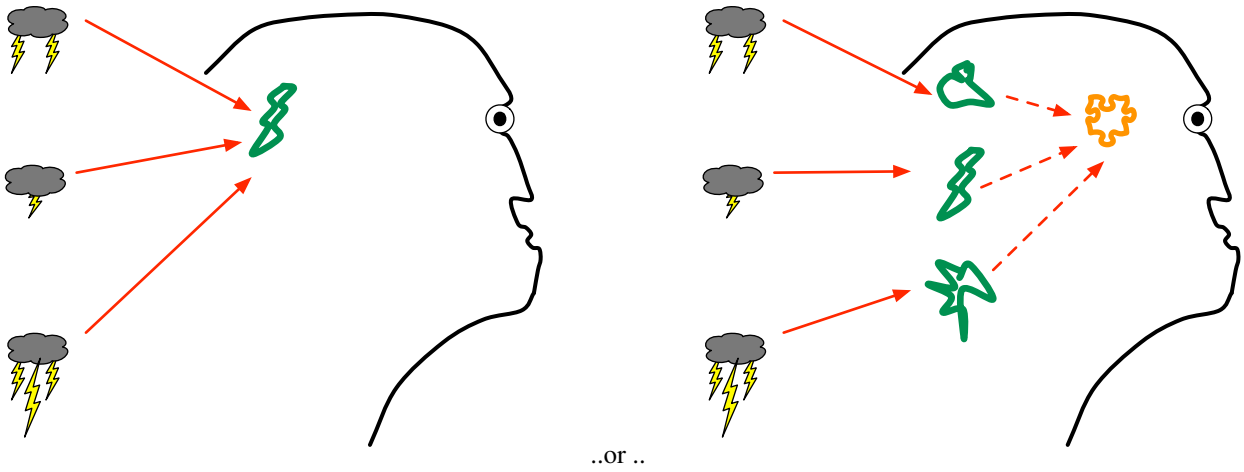


Figure 6: Sketch of the basic neural process of sparsification, wherein the same cellular assembly is invoked either by different external events or different (prior) mental states. In short, differences are reduced by further mental processing. In these diagrams, as in the brain, primitive visual processing begins in the back of the head, with higher-level patterns forming toward the front.

Together, these pattern-processing activities facilitate intelligent behavior. Correlations between neural assemblies and world states not only enable organisms to recognize familiar situations, but also to think about those contexts when **not** actually in them. This ability to reason about situations other than those of the immediate sensory present constitute somewhat of a breakthrough in mental evolution, according to Deacon [3]; it supports complex faculties such as general symbolic reasoning and language usage.

Pattern association and completion are often touted as the hallmarks of human intelligence [2]. They are tasks that we still do much better than computers. Additionally, a good deal of our intellectual prowess stems from sparsification and orthogonalization. The former allows us to generalize over similar situations, and the latter preserves our ability to tailor special behaviors for specific contexts. An interesting combination of the two is topological mapping, wherein similar situations elicit comparable (though not identical) responses via a correlation between one neural space (e.g. sensory) and another (e.g. motor). The structural backdrop for mammalian intelligence is a mixture of sparsifying, topological and orthogonalizing circuits.

This characterization of neural representation in terms of pattern grouping and separation boils down to what (Nobel laureate) Gerald Edelman and (renowned neuroscientist) Giulio Tononi call *differences that make a difference*: distinct neural patterns that have distinct effects upon thought and action [4]. They compare the human brain and visual system to a digital camera. Both can accept megapixel inputs spanning an astronomical pattern space, but only the brain has the ability to *differentially process* a huge number of them; the camera treats most of them identically. Thus, although both systems can receive and *handle* an extraordinary number of data points, the brain, via its elaborate and nuanced scheme for *consistently linking* those input patterns to internal states and actions, exhibits more sophisticated intelligence.

Clearly, the brain's circuitry is more extensive than the camera's, but this alone gives little ground for claiming that the brain has more knowledge. After all, newborn infants have more neural connections than adults, but no more (arguably less) visual intelligence than today's cameras. It is the ability of the brain's synaptic network to create correlations among a huge number of patterns that produces intelligence. In this mature state of pattern matchmaker, the brain's neural networks embody a high level of knowledge that clearly separates us from the vast majority of our machines. **That** difference, however, is diminishing.

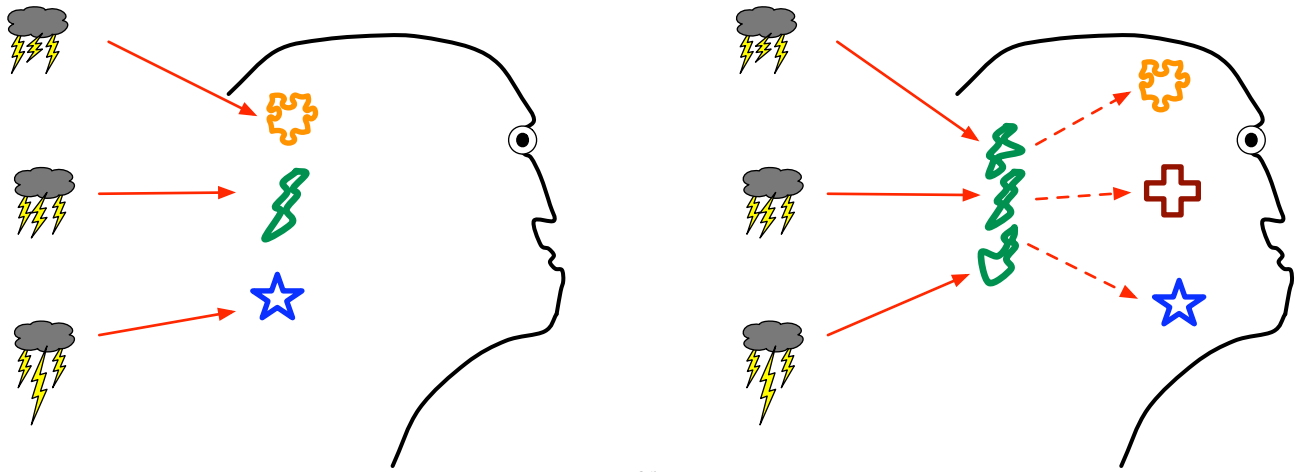


Figure 7: Sketch of the basic neural process of orthogonalization, wherein markedly different cellular assemblies are invoked by similar world events or similar (prior) mental states. In short, differences are accentuated by further mental processing.

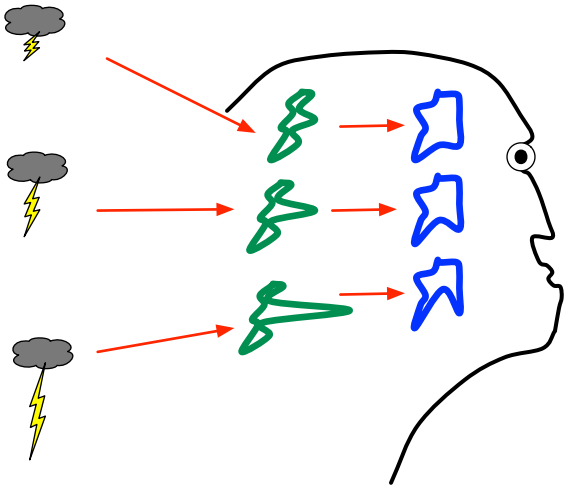

Figure 8: Sketch of the basic neural process of topological mapping across three different spaces: one of external stimuli and two of neural patterns (in different areas of the brain).

## 2.4 Hierarchical Cluster Analysis

The degree to which an ANN properly balances sparsification, orthogonalization and topological mapping to best represent a domain (and most judiciously link differences to differences) is a very important characteristic - one that ANN researchers like to gauge, either formally or informally. Quantitatively, one can measure the correlation between input patterns and output patterns, or between inputs and the activation state(s) of any hidden neuron(s). This single correlation/covariance value provides a global assessment of the ANNs *understanding* of a domain, but a finer grained assessment also has virtue, particularly in more complex networks that generate several categories and employ multiple responses.

The hierarchical cluster plot, a very popular connectionist tool, provides a very detailed, but easily understandable, overview of the implicit categories formed by an ANN. It displays each input case (C) in a graphic which shows the other input cases that the ANN treats most similarly to C. The behavioral basis for this similarity judgement can vary, but it often involves the patterns of activation of the hidden or output layers.

Consider the 6 cases in Table 1, each of which indicates the hidden-layer activation pattern produced by a hypothetical ANN upon receiving that case as input (where most of the detailed features, given as input to the ANN, are omitted from the table). Based solely on these activation patterns, we can hierarchically cluster the 6 cases to determine if, for example, the network treats dogs and cats in a distinctively different manner.

| Animal | Name | Hidden-Layer Activation Pattern |
|---|---|---|
| Cat | Felix | 11000011 |
| Dog | Max | 00111100 |
| Cat | Samantha | 10001011 |
| Dog | Fido | 00011101 |
| Cat | Tabby | 11011001 |
| Dog | Bruno | 10110101 |

Table 1: Portions of a data set and its treatment by a hypothetical 3-layered ANN with 8 hidden nodes. Each data instance presumably includes several descriptive features that serve as inputs to the ANN. For this example, the hidden-layer activation patterns that the ANN produces for each case are of interest, as these form the basis for hierarchical clustering.

A wide array of hierarchical clustering algorithms exist; this section makes no attempt to summarize them. We merely employ the following basic approach:

> Begin with N items, each of which includes a *tag*, which in this example is the hidden-layer activation pattern that it evokes.
>
> Encapsulate each item in a *singleton cluster* and form the cluster set, C, consisting of all these clusters.
>
> Repeat until size(C) = 1
>
> > Find the two clusters, $c_1$ and $c_2$, in C that are *closest*, using distance metric D.
> >
> > Form cluster $c_3$ as the union of $c_1$ and $c_2$; it becomes their parent on the hierarchical tree.
> >
> > Add $c_3$ to C.
> >
> > Remove $c_1$ and $c_2$ from C

In this algorithm, the distance metric D is simply the average hamming distance between the tags of any pair of elements in $c_1$ and $c_2$:

$$D(c_1, c_2) = \frac{1}{M_1 M_2} \sum_{x \in c_1} \sum_{y \in c_2} d_{ham}(tag(x), tag(y)) \tag{1}$$

where $M_1$ and $M_2$ are the sizes of $c_1$ and $c_2$, respectively. Applying this algorithm to the dog-and-cat data set yields the cluster hierarchy (also known as a *dendogram*) of Figure 9. In the first few rounds of clustering, the tight similarities between the tags of, respectively, Fido and Max, and Samantha and Felix, trigger those early groupings. Bruno and Tabby are then found to be closer to those new clusters than to each other, so they link to their respective 2-element clusters at the next level of the hierarchy. Finally, at the top, the dog and cat clusters link up.

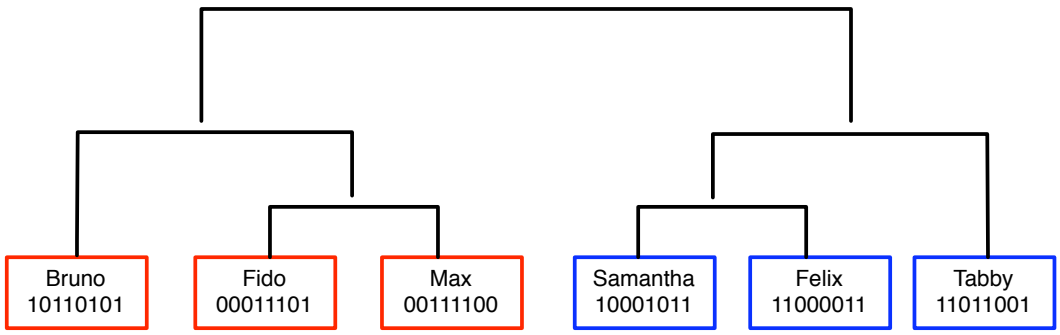


Figure 9: A dendogram for the dog-and-cat data set, using a) the hidden-layer activation pattern (displayed below each animal's name) as the basis for similarity assessments, and b) the distance metric of equation 1.

The important information in this dendogram is that the ANN is treating dogs more similarly to each other than to cats, and vice versa. Thus, the ANN implicitly **knows** something about the difference between these two species, as reflected in the behavior of it's hidden layer.

In large ANN applications - where the training method may be supervised, unsupervised or even reinforced - the dendogram can reveal intricate distinctions in the complex *decision making* performed by an ANN. In many cases, the diagrams indicate the learning algorithm's uncanny ability to form many-leveled hierarchical categorizations that closely (or exactly) parallel a human's approach to *carving up the world*. They are proof that, indeed, the ANN has learned some very significant knowledge about the data set (and how to react to it), but without that information being in any convenient human-readable form. We have to look hard to see it, and cluster analysis with dendogram visualization is a fine tool for this type of detective work.

# 3 Representations in Synaptic Matrices

Synaptic networks house the potential to create neural activation patterns, with particular combinations of strong and weak, excitatory and inhibitory, immediate and delayed, connections providing a strong bias as to which patterns form, and which ones cling together in temporal sequences. Just as genetic material, when placed in the proper environment, produces the organism that it represents, a synaptic configuration, when properly stimulated (by, for example, an oscillating background) can produce the neural firing patterns that it represents.

This synaptic perspective enables neurons to be interpreted as detectors of features or combinations thereof. The *concept* embodied in these preferred features may map nicely to standard mathematical or natural-language expressions, such as *a man wearing a bright shirt and dark pants*, or, as is often the case, they map to very complex concepts

which can only be expressed by very long descriptions (i.e., not easily compressible) or to concepts so diffuse that no expressions in natural language cover them.

When features are just dimensions in a multi-dimensional space - and individuals are points in that space - then straightforward geometric analysis produces a description of a neuron's detected concept based on its input weights.

Consider a simple neuron, z, as depicted in Figure 10, which accepts weighted inputs from neurons x and y, with weights $w_x$ and $w_y$, respectively. As shown, z has a step activation function with threshold $t_z$.

Since the sum of the weighted inputs will only exceed $t_z$, thus causing z to *fire*, under certain conditions (i.e. certain combinations of x and y values), neuron z essentially functions as a **detector** for those conditions. One very important step in understanding the knowledge content of ANNs is the determination of **what** various neurons have evolved or learned to **detect**. In the example of Figure 10, a simple algebraic analysis reveals these conditions and their geometric semantics.

To simplify the discussion, let us assume the following values: $w_x = 2$, $w_y = 5$, and $t_z = 1$, as shown in the lower network of Figure 10. This means that z will fire if and only if:

$$2x + 5y \geq 1 \tag{2}$$

At this point, we can already get a feel for what z detects: x-y pairs where the sum of twice x and five times y is greater than or equal to 1. To visualize this, we can solve for y in equation 2 to yield:

$$y \geq -\frac{2}{5}x + \frac{1}{5} \tag{3}$$

We can then draw the corresponding border (line) and region (upper side of that line) denoted by equation 3 in the cartesian plane, as shown in the lower left graph of Figure 10. This border separates positive from negative instances of the *concept* detected by neuron z. The bottom right of Figure 10 shows several of these positive and negative instances, all of which can be separated by this single line, derived directly from the firing conditions of neuron z. In this sense, neuron z cleanly separates the positive and negative instances of a concept, with no error.

Working backwards, we can receive a collection of positive and negative concept instances (i.e. a data set) and search for a line that separates them - there will either be none or an infinite number of such lines. If a line is found, its equation can easily be converted into an activation precondition (as in equation 2), from which the weights $w_x$ and $w_y$, along with the threshold $t_z$, can be found. Thus, a simple 3-node neural network can be built to detect the concept embodied in the data set.

In cases where a separating line exists, the data is said to be **linearly separable**. This concept extends from the cartesian plane to any k-dimensional space, where linear separability entails that a hyperplane of k-1 dimensions can cleanly separate the k-dimensional points of the data set. Of course, in k-dimensional space, the detector neuron z would have k input neurons, but the activation function would have the same form:

$$x_1 w_1 + x_2 w_2 + \cdots + x_k w_k \geq t_z \tag{4}$$

In general, a data set is linearly separable if and only if (iff) a neural network with one output neuron and a single layer of input neurons (whose sum of weighted activation values are fed to the output neuron) can serve as a perfect detector for the positive instances of that data set.

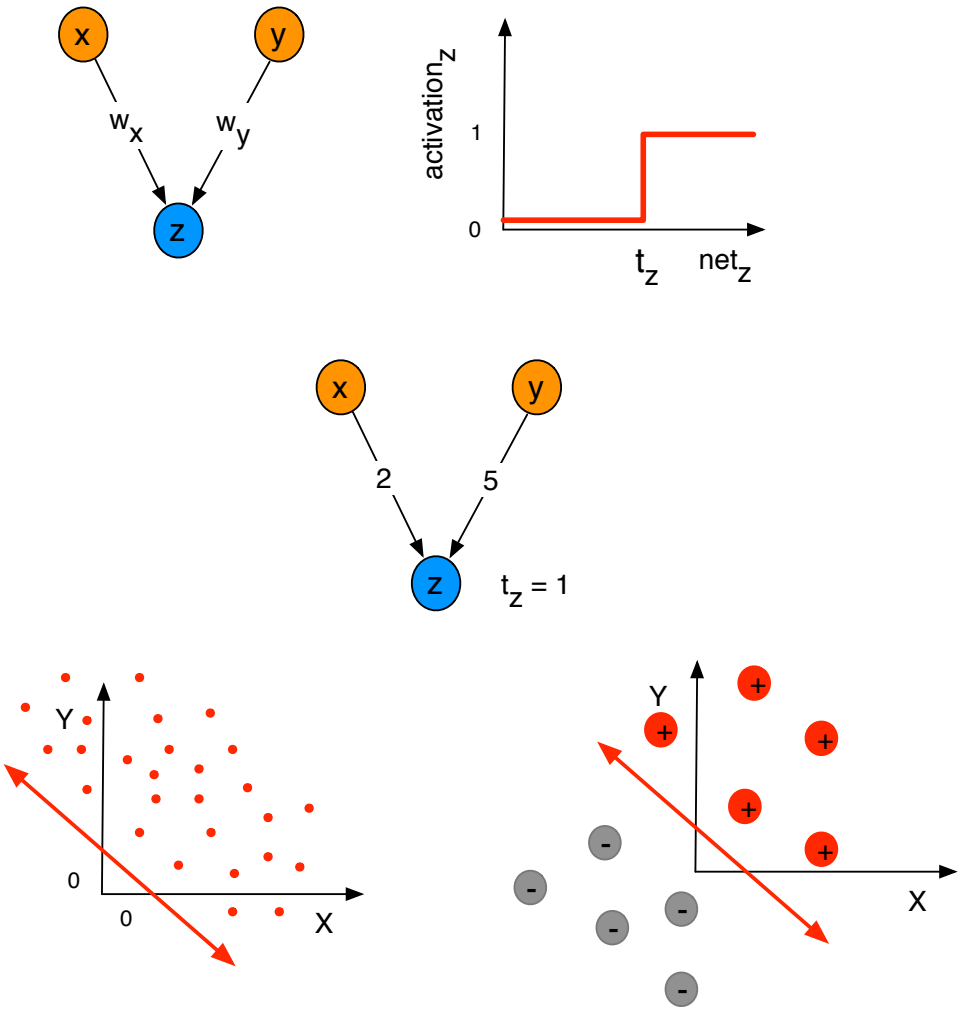


Figure 10: (Top Left) A simple neural network with 2 inputs and one output node. (Top Right) The step activation function for the output node, z. (Middle) An instantiation of this network with real values for the weights and threshold. (Bottom Left) The border (line) and region (area above the line filled in with small dots) represented by the network and expressed by equation 3. (Bottom Right) The separation of positive and negative instances of the *concept* detected by neuron z.

Interestingly enough, a data set need not be large nor exceptionally complex to be non-linearly separable. For example, as shown in Figure 11, simple boolean functions of two variables can evade linear separability. Whereas AND and OR are linearly separable, XOR is not.

To build an ANN detector for XOR, one needs a network with 3-layers: input, middle and output, as shown at the top of Figure 12. The middle layer consists of AND nodes that detect each of the two precise instances of XOR, while the output layer computes the OR of the middle-layer outputs. Clearly, this is not a very satisfying example of a *general solution*, since each of the 2 instances requires special treatment, i.e. a dedicated detector is required for each.

Building detectors for more complex functions, by hand, can be an arduous process, but if several lines can be drawn to separate the data into positive and negative instances, then a straightforward procedure using multiple layers does the job, as detailed in Appendix (appendix reference). However, the tedious nature of this process indicates why all serious ANN users employ automated learning algorithms to train their networks, i.e. to find the proper connection weights.

## 3.1 Sidebar: Sigmoids and Fuzzy Borders

The form of the output neuron's activation function will not help to *bend* the border line to separate the data. Rather, a more complex function such as a sigmoid only adds a *grey region* on each side of the border line: an area in which positive and negative instances are difficult to differentiate. This is illustrated in Figure 13. Clearly, any $net_z$ value that pushes the sigmoid upward (toward the red region or beyond) is a better candidate for concept membership than one that maps to the base of the sigmoid slope(yellow), but a precise border for concept membership, as in the case of a step function, is absent.

(END OF SIDEBAR)

In well-trained ANNs, the salient concepts (embodied in detector-neuron behavior), whether easily comprehensible to humans or not, work together with the concepts embodied by other detectors to achieve an effective overall functionality. Unfortunately, reductionist attempts to explain that functionality in terms of sequences of neural firings can be extremely difficult, because the *meanings* of those firings (in terms of the concepts detected by the neurons that fire) can be complex or diffuse.

There is no simple solution to this problem, but one can get a sense for a detectors preferred concept by examining the weights on all incoming connections. As shown in Figure 14, if the input patterns are faces, then the weights for a hidden-layer neuron can be projected back onto the plane of the input image (with pixel intensity corresponding to weight strength, relative to the other input weights to the same neuron). The resulting image gives a visual summary of the face that most strongly excites the detector neuron. This technique is commonly used in explaining the workings of the immediate downstream layer to the input layer of an ANN. When the input data is not of a visual form, such as acoustic- or infrared-sensor data, then the visualizations of preferred concepts are equally easy to produce in visual graphic form (using the same process as above) but typically harder for humans to interpret.

A classic example of this decoding process for face-recognition ANNs is found in [1], where interesting, almost eerie, *facial holons* are the preferred stimuli of hidden-layer neurons. These holons have no direct resemblance to any of the input faces, but appear to contain a potpourri of features from several of the original faces. One look at these holons provides convincing evidence that the concepts embedded in automatically-trained ANNs have few counterparts in natural language.

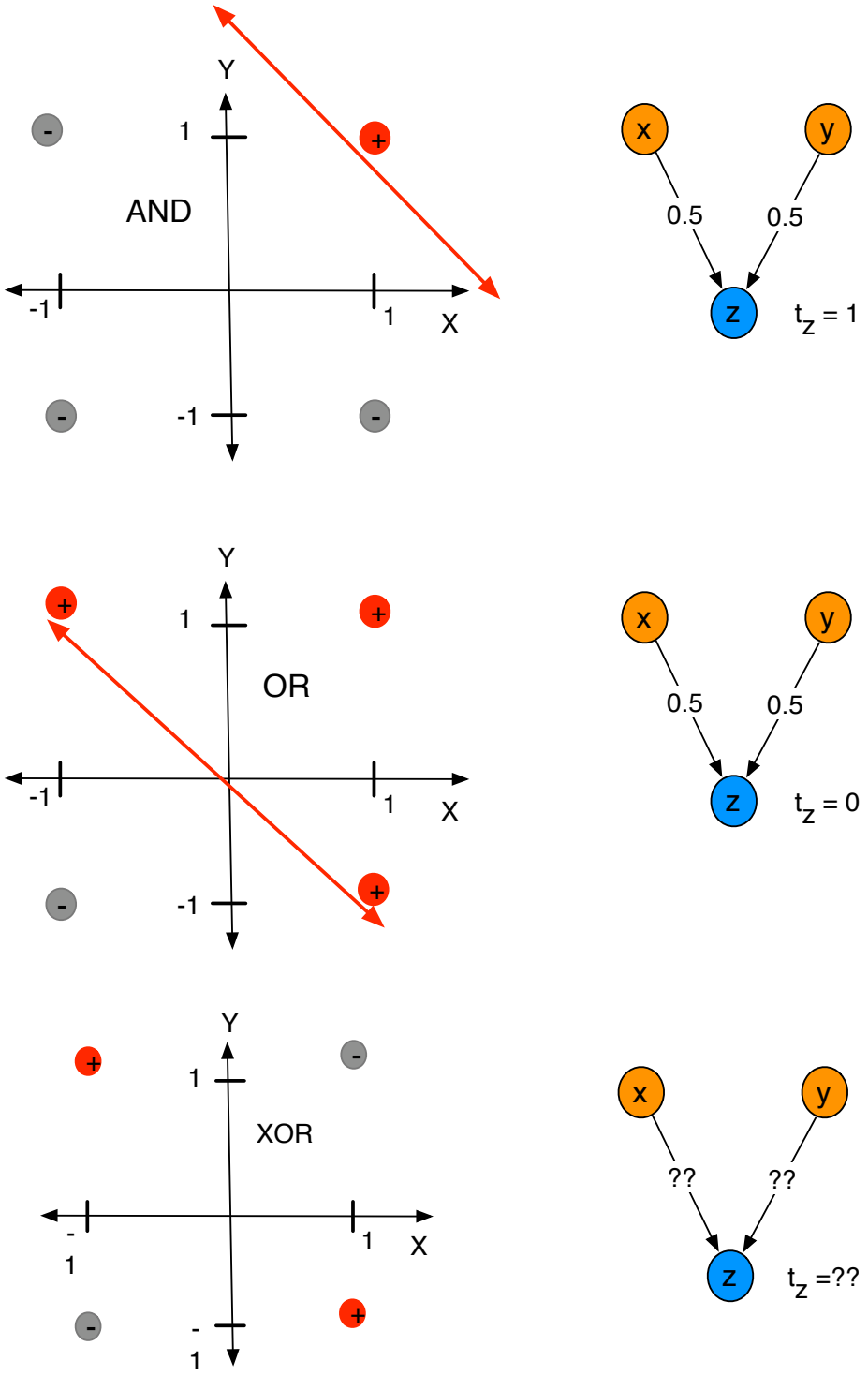


Figure 11: (Left) Cartesian plots of 3 standard boolean functions, where True = +1 and False = -1. (Right) Simple ANN detectors for the 4-element data sets for AND and OR. The borders (lines) corresponding to the activation conditions for these ANNs are drawn as long, double-arrowed lines on the cartesian plots. Since XOR is not linearly separable, no simple ANN detector exists; a multi-layer network is needed.

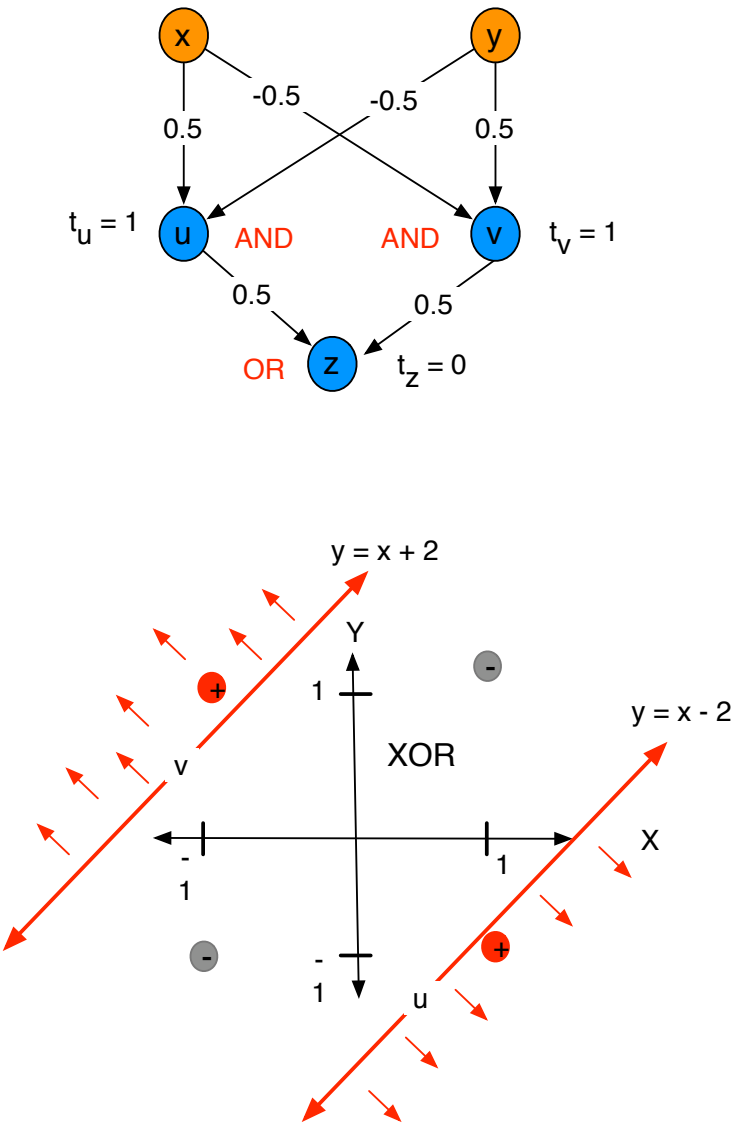


Figure 12: (Top) A 3-layered ANN that realizes the exclusive-or (XOR) function. (Bottom) The borders (long diagonal lines) and regions (arrows) delineated by the activation conditions of the AND neurons u and v. Points in either of the two regions are positive instances of XOR, while all other points are negative instances.

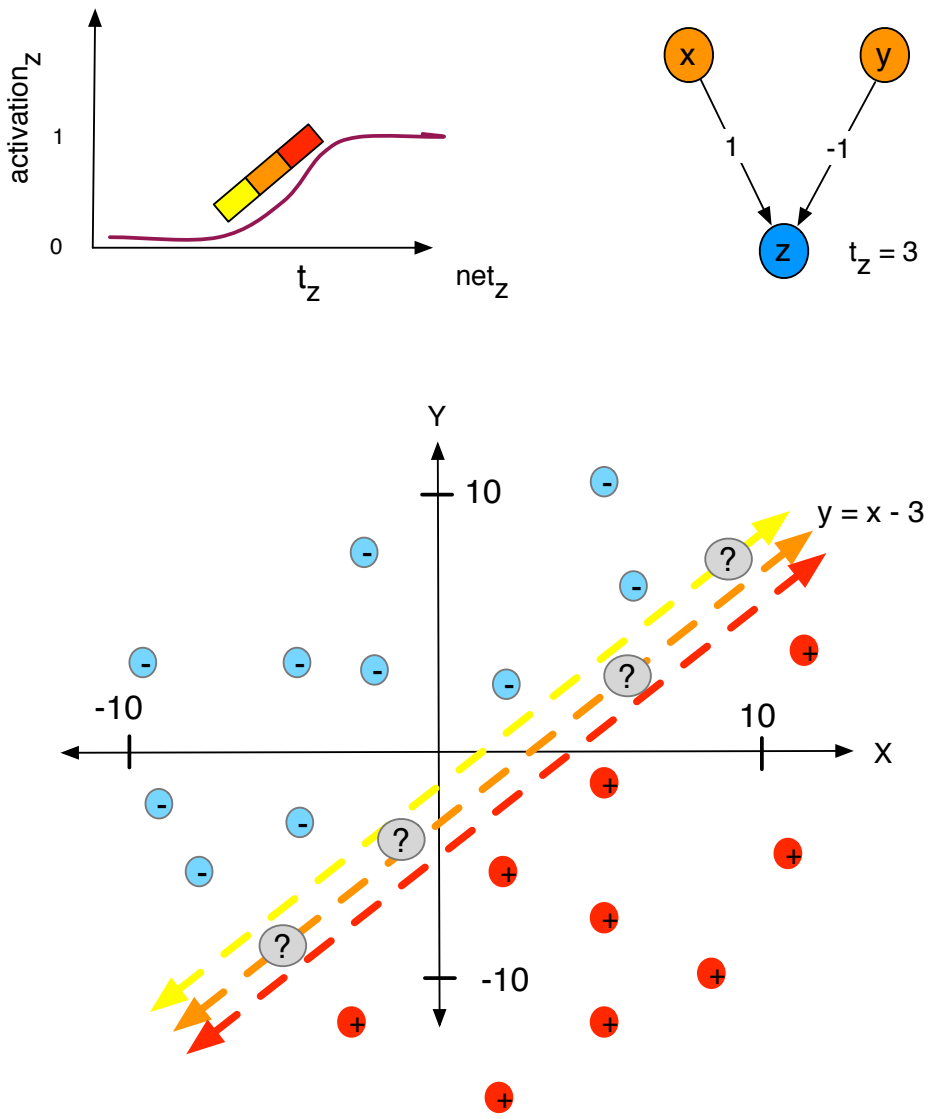


Figure 13: An illustration of the imprecision of class membership decisions incurred by the use of a sigmoidal activation function in a detector neuron. (Top Left) A basic sigmoidal curve with a multi-colored bar paralleling the upward slope. This bar color-codes the strength of class membership (where yellow is low and red is high) for a given $net_z$ value. (Top Right) The detector neuron, z, with an unspecified activation function but a threshold of 3. (Bottom) A crisp separation (given by the orange line) versus a fuzzy differentiation (given by the region between the yellow and red lines) produced in the cartesian plane as a result of a step activation function versus a sigmoidal activation function for z. In the case of the sigmoidal, the threshold of 3 is the $net_z$ value that maps to the middle of the upward slope of the curve (in the top left diagram).

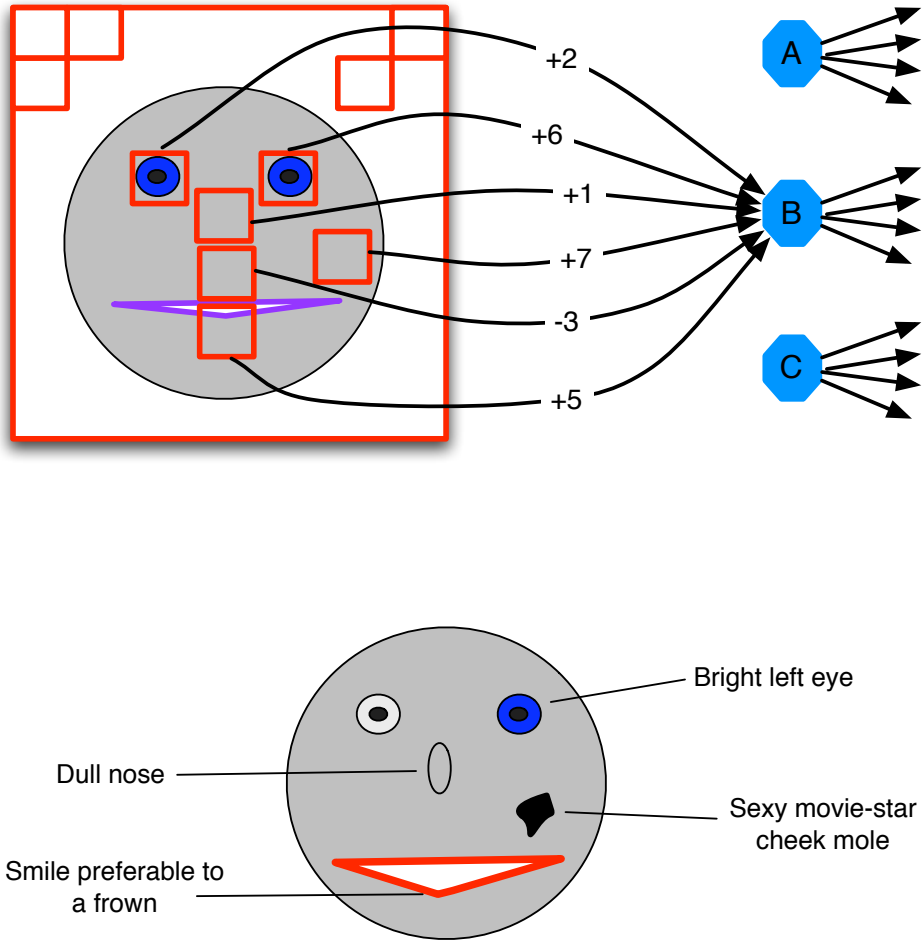


Figure 14: Deciphering the preferred input of a particular hidden node, B. (Top) Sketch of a neural network used for facial recognition. The weights on incoming connections to neuron B indicate those features of the image toward which B is most sensitive. (Bottom) Based on the incoming weights, a *dream face* for neuron B can be constructed, i.e. a face that would cause node B to fire strongly.

## 3.2 Principle Component Analysis

By properly tuning synaptic weights, a neural network can achieve an intelligent balance between orthogonalization and sparsification. In fact, Hebbian Learning can produce networks that realize a simple version of principle component analysis: they differentiatiate (orthogonalize) along the high variance component(s) and abstract (sparsify) along the rest. Thus, the network, as a pattern detector/classifier, separates input patterns based on the features that have the most variance in the sample population; and these are often considered the most significant features in a classification task. For example, when organizing a group of people to perform a complex task composed of many diverse subtasks - such as playing a team sport, giving a concert, or planning the economy of a large city - the manager normally overlooks low-variance features of individuals, such as the age of junior-high band members, and focuses on high-variance properties, such as the particular instrument a teenager plays. These sources of diversity are normally the salient pieces in the organizational puzzle.

Formally, the principle components of a data set are vectors that capture the highest amounts of variance in that data. If the data set has two attributes, student age and instrument tone (for the junior-high band members), then we expect little variance in age but considerable variance in tone. The principle component vector of this data will then have a large bias (reflected in a large vector-component value) toward tone, with a much smaller value for age.

One very significant discovery of ANN research is the following:

> **If** (a) the values of a data set are scaled (to a common range for each feature such as [0, 1]) and normalized by subtracting the mean vector from each element, (b) the modified data values are fed into a single output neuron, z, and (c) the incoming weights to z are modified by general Hebbian means (wherein the correlation between input values and z's activation level have a direct influence upon weight change), **then** z's input-weight vector will approach the principle component of the data set.

An important detail is the simple mathematical fact that **the border between regions carved out by a single output neuron is perpendicular to the weight vector**. This is easily shown by the following derivation:

$$xw_x + yw_y \geq t_z \leftrightarrow y \geq -\frac{w_x}{w_y}x + \frac{t_z}{w_y} \tag{5}$$

This defines a region[1] whose borderline has the slope $-\frac{w_x}{w_y}$. Then, any vector with slope $+\frac{w_y}{w_x}$ is perpendicular to that border. Since neuron z's incoming-weight vector is $\langle w_x, w_y \rangle$, it has slope $+\frac{w_y}{w_x}$ and is therefore perpendicular to the borderline. This implies that the border will separate the data set based on those factors with highest variance.

Consider the general significance of this property in the context of another simple example. In Figure 15, the instances of a hypothetical data set consist of two attributes: gray-scale color and size. Further, assume that the instances either involve mice or elephants. Since both mice and elephants are often gray in color, their gray-scale values will be similar, whereas their sizes will differ considerably, as the upper plot in the figure partially indicates - the actual difference is 5 or 6 orders of magnitude.

First, scale each feature value to lie within a small range, such as [0, 1]. Next, compute the average vector, which is the vector of the independent averages of each scaled feature (color and size) across the full data set; it is depicted as a dark diamond in Figure 15. Next, normalize each scaled data point by subtracting this average from it. This yields scaled, normalized data points, shown at the bottom of the figure.

[1] Of course, if $w_y$ is a negative value, then our derivation would involve a switch of the inequality, but the slope and location of the border line remains the same

Now, assume that a simple output neuron (z) with two inputs (x and y), one coding for size and the other for gray-scale color, were trained on this data. Postponing (for a moment) the details, if this training involved Hebbian learning, then the input-weight vector for z would become tuned to something akin to the nearly-horizontal, green arrow at the bottom of Figure 15. This vector contains a large x component and a small y component - not because elephant sizes are so large, but because the **variance** in size between mice and elephants is so much larger than the variance in color. Remember that we've already subtracted out the average vector, so if all animals in the data set were large (i.e., elephant sized), then our normalized x values would be small.

Since the borderline vector (for the concept detected by z) is perpendicular to z's weight vector (as explained earlier), it resembles the nearly-vertical, orange arrow at the bottom of Figure 15. This border separates positive from negative examples of z's concept, and the groups that it most clearly differentiates are small animals (e.g., mice) and large ones (e.g., elephants). That is, the neuron will only fire on data points from one of those categories, not both. It detects the **most significant difference** in the data set: the *principle component*.

If the data set consisted solely of African elephants (the largest species of land animal), then the weight variance would be greatly reduced, compared to the mice-elephant scenario. This would reduce the extreme horizontality of the weight vector, thus reducing the verticality of the borderline, possibly to a level at which neuron z would appear to primarily differentiate between light and dark animals. If the variances in color and size were of similar magnitude, then the borderline (perpendicular to the weight vector, determined by Hebbian Learning) might only differentiate between concepts that, to a human observer, seem useless, such as *very large, light-gray animals* versus *moderately large, dark-gray animals*.

To see how Hebbian Learning nudges weight vectors in the direction of the principal component, consider a quantitative version of the mice-elephant scenario. First, all inputs to an ANN should be scaled to within the same range, such as [0, 1] or [-1, 1] to insure that each feature has an equal opportunity to affect the behavior of the network, both during and after learning.

Assume an initial data set (size, gray-scale) consisting of the six raw data points shown in Table 2. The data points are then scaled and normalized, as described in the figure. The values in the rightmost column are then fed into an ANN with 2 inputs (one for size and one for color) and one output.

| Animal | Raw Data | Scaled Data | Normalized Data |
|---|---|---|---|
| Mouse | (0.05, 60) | (0, 0.6) | (-0.27, -0.04) |
| Mouse | (0.04, 62) | (0, 0.62 | (-0.27, -0.02) |
| Mouse | (0.06, 68) | (0, 0.68) | (-0.27, 0.04 |
| Elephant | (5400, 61) | (0.54, 0.61) | (0.27, -0.03) |
| Elephant | (5250, 66) | (0.53, 0.66) | (0.26, 0.03) |
| Elephant | (5300, 69) | (0.53, 0.69) | (0.26, 0.05) |

Table 2: A hypothetical data set consisting of 3 mice and 3 elephants, with features being size (in kg) and gray-scale color (in the range 0 (black) to 100 (white)). The data are first scaled to the ranges (0, 10,000), for size, and (0,100), for color. They are then normalized by subtracting the average vector (0.2700, 0.64).

Assume that the initial weight on both connections is a small positive value, 0.1. Also assume that the activation function for the output node is straightforward linear scaling: the activation level directly equals the sum of weighted inputs. Finally, assume the following Hebbian learning rule:

$$\triangle w_i = \lambda x_i y \tag{6}$$

where $x_i$ is the activation level of the ith input node, y is the activation level of the output node, $w_i$ is the weight on the arc from the ith input to the output node, and $\lambda$ is the learning rate, which we assume to be 0.2.

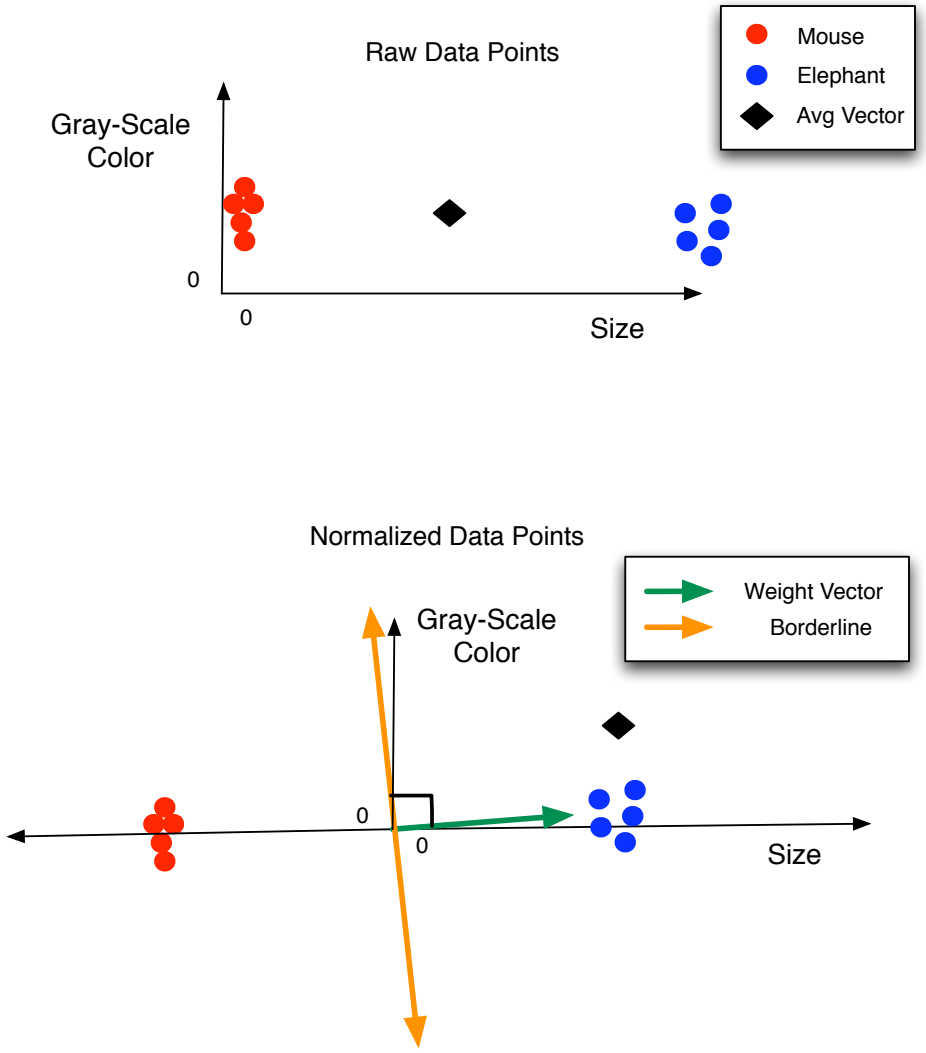


Figure 15: (Above) A sketch of hypothetical data points taken from mice and elephants, where each point consists of size and color attributes. No scale is specified, although it would have to be logarithmic to capture the huge discrepancy in size. The average data point/vector (dark diamond) appears in the middle. (Below) After scaling and then subtracting the average vector from each data point, the plots of these scaled, normalized data points straddle the x and y axes. When a simple neural network is trained (using Hebbian Learning) on these normalized data points, the weight vector of the ANNs output node would show a strong bias toward size, since it has the highest variance. This produces a very horizontal weight vector, and thus a very vertical borderline to separate the positive and negative instances of the concept detected by the output neuron.

As shown in Table 3, the changes to the weight between $x_{size}$ and the output are much larger than those from $x_{color}$. Hence, after one round of batch learning - No weights actually change until after each data point has been processed and its recommended change is recorded - their will be an increase in $w_{size}$ that is nearly 20 times larger than the increase in $w_{color}$. After several rounds of learning, $w_{size}$ will completely dominate the detector's weight vector.

Why did this dominance occur? In looking at the data of Figure 3, note the order-of-magnitude difference in the absolute values of the scaled, normalized size values compared to those of the color values. Remember that this no longer reflects any bias stemming from the fact that elephants are so large; it arises solely due to the large variance in size between mice and elephants compared to the small variance in color between the two animal species. This variance imbalance entails that the size value will dominate the color value during each of the 6 runs of the ANN. Hence, the output value will have the same sign and general magnitude as the size input. Since the Hebbian learning rule multiples these two values (along with $\lambda$), and since they have the same sign, the weight change will be positive.

Conversely, the color input may or may not match the sign of the output, incurring positive and negative weight changes with nearly equal probability. Furthermore, the small magnitude of the color value will restrict the size of those weight changes. Hence, $w_{color}$ will change in small increments, both positive and negative, whereas $w_{size}$ will change in larger increments, all positive.

| Input (Size, Color) | Output | $\delta w_{size}$ | $\delta w_{color}$ |
|---|---|---|---|
| (-0.27, -0.04) | -0.031 | +0.0017 | +0.0002 |
| (-0.27, -0.02) | -0.029 | +0.0016 | +0.0001 |
| (-0.27, 0.04) | -0.023 | +0.0012 | -0.0002 |
| (0.27, -0.03) | +0.024 | +0.0013 | -0.0001 |
| (0.26, 0.03) | +0.029 | +0.0015 | +0.0002 |
| (0.26, 0.05) | +0.031 | +0.0016 | +0.0003 |
| Sum weight change: | | +0.0089 | +0.0005 |

Table 3: The result of training a 2-input, 1-output ANN on the scaled, normalized data of Table 2, using a simple Hebbian learning rule. It is assumed that the training is done in batch mode such that none of the weight changes are applied to the network until each data point has been processed.

The general implications of this rather specific technical detail are very significant:

> If the detectors of a network modify their input-weight vectors according to basic Hebbian principles, then, after training, the activation levels of detectors can be used to differentiate the input patterns **along the dimensions of highest variance**. Hence, those detectors will differentiate between objects (or situations) that are **most distinct** relative to the space of feature values observed in the training data.

As an example, if an ANN is trained on human and cat faces, then it would probably learn to fire (an output) on only one or the other species, not both - certainly a sensible way to partition the data set. Conversely, if the ANN were only trained on human faces, then the separation would occur along another dimension (or combination thereof) of high variance, such as male versus female, black versus caucasian, or maybe happy versus angry. The beauty lies in the fact that the ANN figures out the proper combination of discriminating factors, even if that combination evades concise formulation in human language.

# 4 Neuroarchitectures to Realize Effective Distributed Coding

As described above, the often-touted memory-capacity advantages of distributed coding are frequently exaggerated in the context of associative neural networks, due to content-addressable functionality typically desired of such systems.

However, organisms derive substantial benefits from distributed codes in terms of generalization (derived from pattern sparsification).

Figure 16 abstractly illustrates this fundamental advantage of a distributed code. Assume that an organism's *flight response* is driven by a single motor neuron, X, at the bottom of the figure. Tuned to trigger on the firing pattern 101110 of its upstream neighbors, X will probably also trigger on similar patterns, such as 101100 and 001110, which involve many of the same active neighbors. This makes great, life-preserving, sense. A gazelle that only flees from tigers with exactly 7 stripes cannot expect to live very long. The ability to generalize/sparsify sensory patterns such that many map to the same successful behavior(s) is paramount to adaptation, intelligence, and, ultimately, survival.

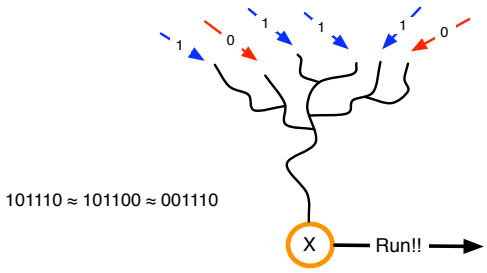


Figure 16: Illustration of a distributed code (and several similar codes) that stimulates a motor neuron.

Distributed coding in networks of simple neurons provides this flexibility as a by-product. Via synaptic change, neurons become tuned to fire on particular combinations of afferent stimuli, but not necessarily those *exact* combinations. As long as the net input to the neuron exceeds a threshold, it fires. So as long as the network uses distributed coding, and the detector neuron maintains a threshold that prevents it from firing on all-too-sparse input patterns, the detector should easily generalize over many dense distributed codes and thus fire in many (similar) situations. This is a huge advantage of distributed memory systems composed of simple nodes joined by plastic links.

So the use of distributed codes involves tradeoffs, and a key question is whether a neural system can somehow avoid their pattern-corrupting dangers while still taking advantage of their generalizing power? The brain appears to have found a way.

First of all, note that the distributed afferent patterns on the dendrites of Figure 16 do not necessarily originate from a pattern-associating layer. These afferents neurons may have very little influence upon one another. Although advantages of incorporating them in such a layer would include the ability to complete partial patterns, a) the disadvantages of interference and memory corruption could quickly override the positive effects, and b) neuron X may not require a completed afferent pattern to fire; anything reasonably close to the original pattern, that it has learned to detect , may work fine.

In short, the brain may exploit dense, distributed codes at various sites without having to store (and thus retrieve from partial patterns) many of them within a common sub-population of neurons. The brain may reserve associative memories for only relatively sparsely coded information.

But clearly humans (and other animals) have complex memories that a) involve thousands, if not millions, of neurons, and b) are retrievable from partial instantiations of themselves. How might this work?

Three brain regions, cerebellum, basal ganglia and hippocampus, all give some hints of a solution, as recognized by Marr [6, 7] and later elaborated by Rolls and Treves [11]. Each of these areas contains an entry-level layer of neurons that exhibit very competitive interactions: when active, they either directly, or via interneurons, inhibit their neighbors. This leads to very sparse coding at these entry layers, since only a relatively small subset of neurons can be active at the same time. However, fan-in to these layers can still be high (as it is in the basal ganglia and hippocampus). Thus,

a dense distributed code in the space of afferents compresses to a sparse code within the competitive layer.

If this competitive layer then feeds into an associative network, the latter will only be expected to store and recall sparse codes, which, as we have seen above, can be achieved with little interference. This is precisely the case in the hippocampus, where the dentate gyrus serves as a competitive layer which then sends afferents to CA3, which, with its high degree of internal (recurrent) connections, appears optimally designed for associative pattern processing.

Figure 17 illustrates this basic combination of layers. Dense distributed codes are detected by neurons in the competitive layer, with ample flexibility to generalize across similar dense codes - just as the neurons of artificial competitive networks often fire on all vectors in the neighborhood of their prototype. Competition leads to sparse coding, since only a few neurons are active at any one time, and these sparse patterns are sent to the associative layer, which uses Hebbian learning to strengthen the synapses between co-active cells.

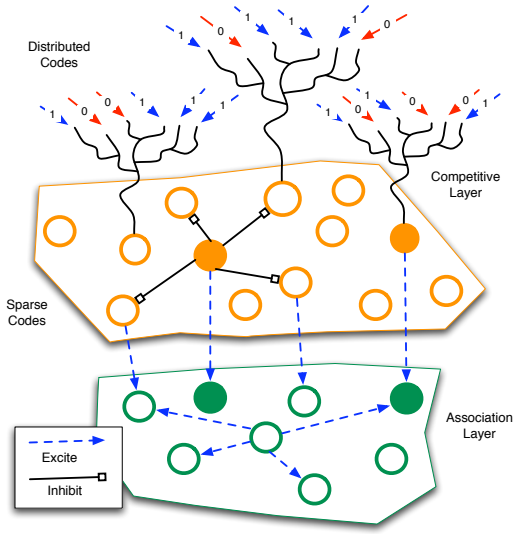


Figure 17: Using a competitive layer to sparsify dense distributed codes, which are then sent further to an associative layer.

Thus, the sparsification of one layer by another normally involves an *expansion* in layer size, a concept known as *expansion recoding*. As detailed in Appendix ??, a sparse layer needs many orders of magnitude more neurons than an upstream dense-coding layer, if the former is to faithfully detect patterns in the latter. Figure 18 portrays this general topology, which is evident in both the hippocampus and cerebellum [11].

## 4.1 Pattern Processing with Dense Distributed Codes

Figure 17 shows how a sparsely-coding competitive layer can serve as an interface between a densely-coded and a sparsely-coded region, the latter of which can then perform associations with reduced interference. However, as mentioned above, densely-coded regions may also require pattern-completing functionality. Since interference becomes a major problem with densely-coded layer coding, equipping that layer with a thick network of recurrent excitatory connections seems impractical. Alternatively, if the dense layer can transfer its partial patterns to a sparse layer, which performs relatively interference-free pattern completion, then recurrent links from the sparse to the dense layer could reinstate the complete dense pattern. This basic architecture appears in Figure 19, where dense coding occurs in a *weakly associative* layer, i.e. one containing a relatively low number of intra-layer excitatory connections [2] .

[2] A justification that sparse coding reduces inference appears in Appendix (appendix reference)

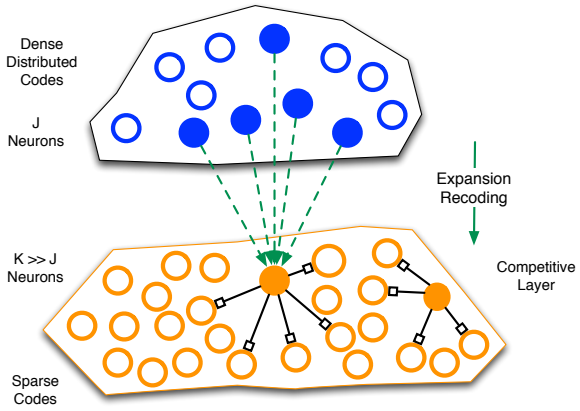


Figure 18: Basic illustration of expansion recoding between a densely-coded layer and sparsely-coded, competitive layer, with considerably more neurons.

Consider a dense pattern, $P_d$, presented to the upper layer of Figure 19. Assume that $P_d$ consists of two component patterns, $C_1$ (the 4 active neurons on the left) and $C_2$ (the 3 active neurons on the right). Competitive layers often learn to detect recurring subpatterns, so if $C_1$ and $C_2$ are frequent components of dense patterns, then such dedicated detectors (the two filled orange circles in the competitive layer) can easily emerge. These, in turn, send low-fanout afferents to the lower associative layer, which processes sparse codes produced by the competitive layer. In the figure, low fanout is represented as a 1-1 correspondence between the two layers.

In the lower associative layer, a high level of internal recurrent connections facilitates the synaptic coupling of coactive neurons via standard Hebbian learning. Thus, the two active nodes (dark green nodes in the bottom layer), which essentially represent $C_1$ and $C_2$, become linked such that either can stimulate the other, where strong linkage is denoted by the thick, dotted blue lines between two neurons in Figure 19.

To complete this circuit, neurons of the strong association layer must link back to neurons representing the individual parts of $C_1$ and $C_2$. Thus, when either of these fire, they can activate $C_1$ or $C_2$ in its entirety. These inter-layer feedback links are drawn as dotted red lines in Figure 19.

With all of these layers and links in place, completion of a densely-coded pattern proceeds as follows. Assume that a sizeable portion of $C_1$ enters the upper layer. This can fire the detector for $C_1$ in the competitive layer, which then sends strong excitation to its corresponding neuron (which also represents $C_1$) in the strong association layer. There, the neuron for $C_1$ activates the neuron for $C_2$, and now both neurons send feedback excitation to the upper-layer neurons that constitute the complete versions of $C_1$ and $C_2$.

Figure 21 depicts this recall of a densely-coded pattern, while Figure 20 displays the circuitry need to store the pattern $P_d$ in the 3-layered network. The important additional feature in this figure is the tuning of recurrent connections from the sparse lower ($A_s$) to the dense upper ($A_d$) associative layer. This involves Hebbian learning driven by coactivity between the $A_d$ neurons constituting $C_1$ and $C_2$; and the $A_s$ neurons that represent each of those components.

This relatively elegant solution (believed to be employed by the brain, as discussed below) appears too good to be true. After all, why should pattern-processing problems at one level be solved by a pattern processor at a different level? How can all of the patterns in $A_d$ be stored (and thus remembered) in $A_s$? This latter question is particularly difficult to answer in the brain, since the region that most strongly corresponds to $A_s$, CA3 of the Hippocampus, has many orders of magnitude fewer neurons than the standard $A_d$, which is the neocortex.

Clearly, $A_s$ cannot store all of $A_d$'s possible patterns. However, if the patterns of $A_d$ are biased such that they often

consist of common subpatterns, then a) a competitive layer can detect these subpatterns, and b) a smaller, but more densely interconnected associative layer can store links between these subpatterns. The key lies in the biased, strictly non-random, nature of the pattern set; and living organisms inhabit a world chock full of these biasing regularities (a.k.a. invariants). Most humans are not capable of remembering a large corpus of random patterns, but gifted individuals or those with some training in memory techniques, can easily remember large collections of *meaningful* patterns, in the sense that they consist of subpatterns that occur in everyday life.

Interestingly enough, patterns with common subpatterns are those most likely to cause interference in a distributed memory. Hence, the basic repetitive nature of our sensory world may have forced evolution to design network memory systems involving multiple layers of well-segregated competitive and associative functionalities.

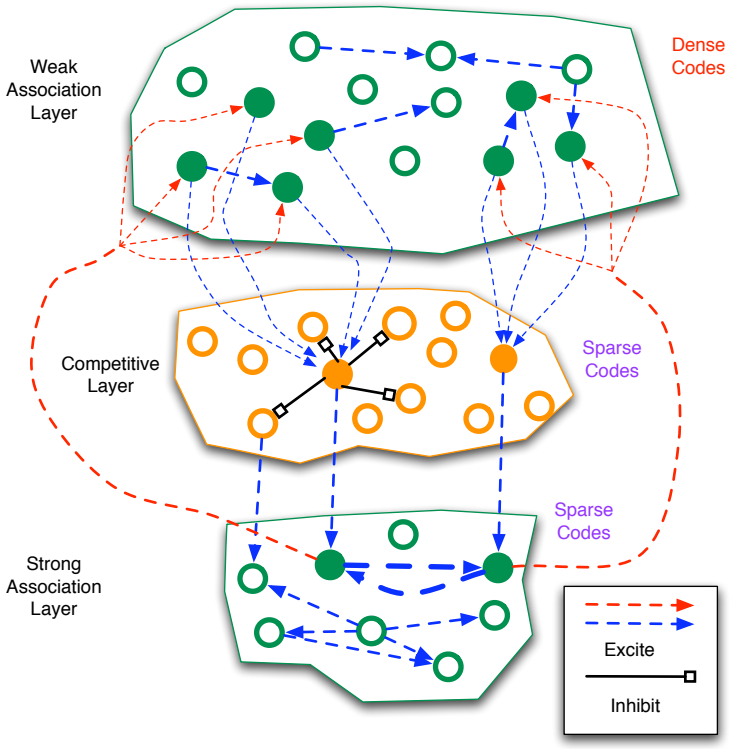


Figure 19: Illustration of the use of stacked competitive and associative layers to support pattern completion in distributed layers that use dense codes. The upper associative layer is weak due to the low number of internal excitatory connections, while the lower associative layer is much stronger. Subpatterns in the dense codes of the upper layer are detected in the competitive layer, which sends its sparse patterns to the lower associative layer, which can then complete sparse partial patterns. Recurrent links from the lower to the upper associative layer enable single neurons in the former to activate neuron assemblies in the latter.

## 4.2 Cortical-Hippocampal Interactions

In the mammalian brain, the interaction between cortex and hippocampus (see Figure 22) is believed to involve a *re-presentation* of densely-coded cortical patterns as sparsely-coded sequences in CA3, a hippocampal region with a very high level of intra-layer recurrence. The dentate gyrus (DG) serves as the competitive-layer entry point to the hippocampus. This, along with the general pattern of convergent collaterals from the primary cortical areas to the entorhinal cortex (EC) - the cortical gateway to the hippocampus - indicates that dense distributed codes in the neocortex are greatly sparsified within the hippocampus.

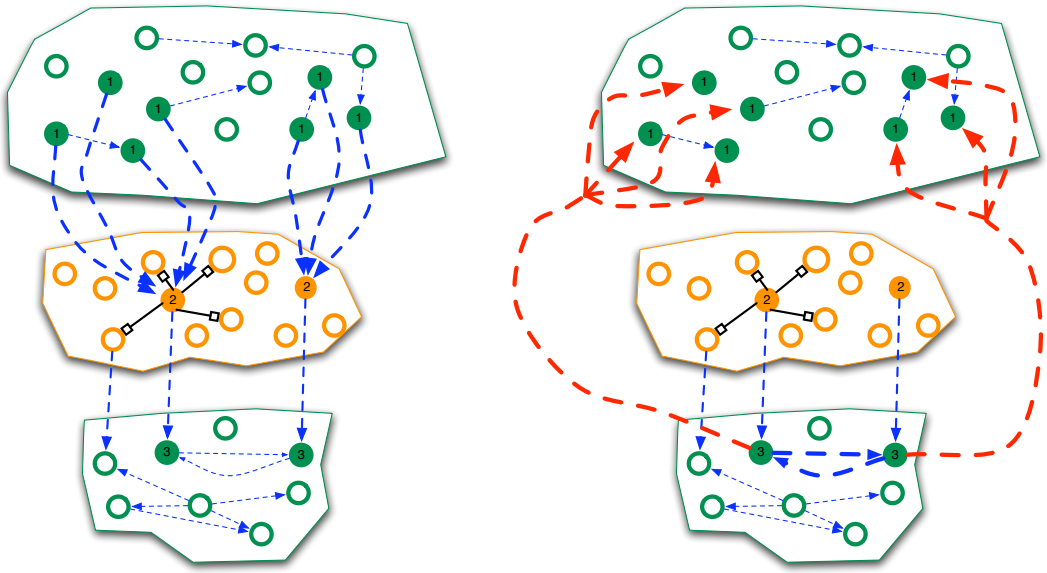

Figure 20: Storage (i.e. learning) of a dense distributed pattern ($P_d$) in a weakly associative layer (top green) with the help of a competitive layer (middle orange) and a strong, sparsely-coded associative layer (bottom green). Filled circles denote active neurons, and the numbers in each filled neuron indicate the time step at which they become active. (Left) After the complete pattern is entered into the weaker associative layer, $A_d$, signals feed forward to the competitive layer, where winner nodes have their active afferent synapses strengthened (thick dotted blue lines) by Hebbian learning. The winners send their signals to corresponding neurons in the lower associative layer, $A_s$. (Right) The synapses between coactive neurons in $A_s$ become strengthened by Hebbian mechanisms (thick dotted blue lines). In addition, feedback collaterals from $A_s$ to $A_d$ (thick red dotted lines) are strengthened when their presynaptic and postsynaptic neurons are active, thus binding $P_s$, the sparse representation of $P_d$ in $A_s$ to $P_d$ in $A_d$. On the right, some connections from the left are removed for ease of readability.

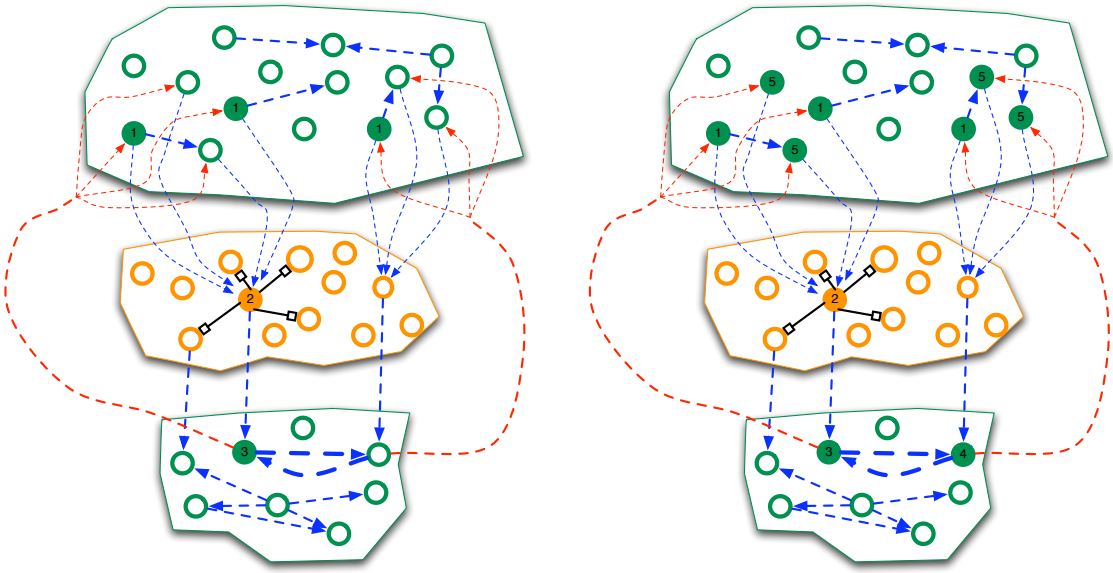


Figure 21: Pattern completion in a densely coded distributed layer (top green) with the help of a competitive layer (middle orange), combined with a sparsely coded associative layer (bottom green), as in Figure 19. Filled circles denote active neurons, and the numbers in each filled neuron indicate the time step at which they become active. (Left) The partial pattern (P*) enters the top layer, $A_d$, at time 1. This stimulates a detector for the leftmost partial pattern in the competitive layer at time 2. The competitive node then directly stimulates its counterpart in the lower association network , $A_s$ at time 3. (Right) At time 4, the simple two-node pattern is completed in $A_s$ due to previously-learned connections between these 2 neurons, which then send excitatory signals back to those neurons of $A_d$ that consitute the complete pattern (P). These feedback signals combined with the few recurrent synapses in $A_d$ stimulate the remainder of P's neurons to activation at time 5.

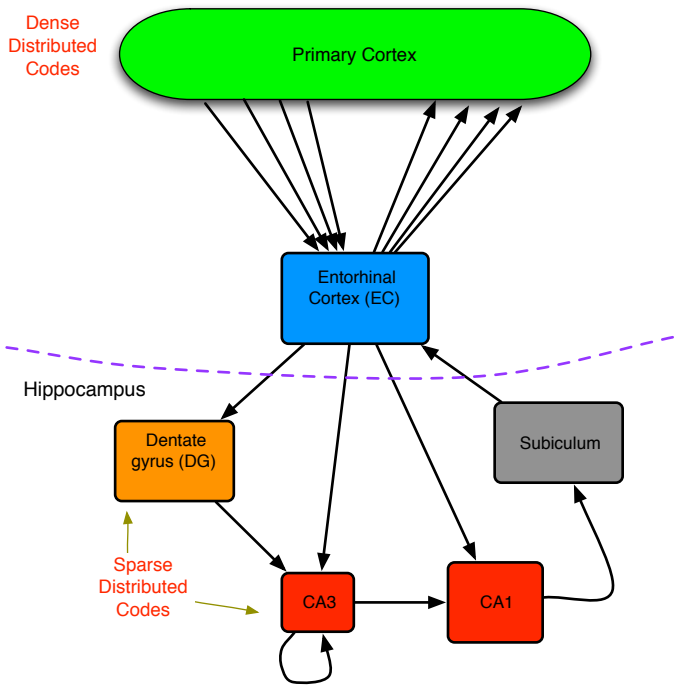


Figure 22: Overview of the hippocampus and its relationship to the cortex, which, for the purposes of this section, is divided into the primary cortex and EC, which serves as the gateway to (and from) the hippocampus for the rest of the cortex. The sizes of each area give a very rough indication of relative sizes between the different brain regions. For example, CA3 contains fewer neurons than DG and CA1 and is much smaller than the primary cortex.

Return signals from CA3 to the cortex - through CA1, the subiculum and EC - allow the completion of dense codes in the primary cortex after their corresponding sparse codes have been completed in CA3. After many instances of CA3-assisted retrieval, a cortical pattern may become self-completing as the weaker associative capabilities of the cortex eventually permit Hebbian learning between the various detailed components of the dense pattern.

One often-cited theory [8] posits that CA3 utilizes high recurrence and a high learning rate to quickly learn abstract (hence sparse) patterns, while the cortex must rely on a much slower learning rate and lower recurrence. Thus, CA3 learns a sparse pattern quickly and then gradually *offloads* a denser version to the cortex via repeated recall, some of which may occur during sleep and dreaming.

### 4.3 3rd-Order Emergence in the Brain

To review, the brain has evolved a topological structure - present in several areas - that performs expansion recoding. In general, this helps reduce interference between activation patterns. When the expansion recoding level pipes its outputs to a second, sparse-coding, associative layer, ($A_s$), and when that layer can feed back to the densely-coding entry level ($A_d$), the resulting circuit enables $A_d$ to gradually tune its synapses to become an effective pattern-processing machine. For the patterns that it has learned, $A_d$ no longer needs help carrying out the duties of a content-addressable memory: it can complete partial patterns.

Essentially, an associational network relies upon 2nd-order emergence to learn and reproduce patterns, but doing so accurately becomes difficult when patterns begin to interfere with one another. One can argue that 3rd-order emergence, as supported by a multi-leveled topology such as the cortico-hippocampal loop, solves the problem: the sparse patterns learned in CA3 have the power to recreate their corresponding dense patterns in the cortex. A key role of the hippocampus is producing these correlations between sparse CA3 patterns and denser cortical patterns. In other words, the hippocampus builds representations of cortical patterns, and these CA3 patterns generate their cortical counterparts by priming a 2nd-order emergent process. During waking hours, this re-creation is aided by sensory input, but during dreaming, the sparse patterns mange quite well on their own, though their cortical correlates tend to be noisy.

The hippocampus directly nurtures 3rd-order emergence by combining two 2nd-order emergent layers into a complex circuit. Many brain regions build correlations between neural populations, but CA3 appears to be one of the most prolific pattern-learning areas. A layer with so many recurrent connections should produce very strong internal correlations, and these are a much better basis for high inter-layer correlation. Consequently, the hippocampus is the queen of the neural match makers.

## 5 The Vital Context of Neural Representation

Analogies between computers and brains frequent the science literature, although less so now than in AI's adolescence. Some significant differences between the two, such as the parallel-versus serial nature of their computations, have blurred (with the ubiquity of parallel computing). But others cannot be ignored. As discussed in an earlier chapter, the hardware-software distinction appears to have no direct counterpart in the brain, since the *code* run by neurons is fully distributed across the brain and intimately tied to the synapses, which, as physical components of neural circuitry, would seem to mirror hardware (although their dynamic nature might earn them the label of *softwire*). So you cannot just pull the code out of a brain and transfer it to another. But if not the code, what about the data? What about the knowledge, the representations?

As this chapter shows, the data is no less engrained in the neural hardware than the *program* is. A neural activation pattern make little sense outside of the brain, but it gets even worse. In a GOFAI system, a chunk of information, such

as a logical axiom, can be passed among different modules via an internal operation that typically involves copying its bit-representation into different locations; and each module interprets those bits the same way. In a neural system, transfer by copying is rare. Instead, one k-bit pattern in region R1 may correlate with an m-bit pattern in region R2 such that the 2 patterns essentially represent one another. The presence of one will, in some contexts, lead to the emergence of the other. However, any attempt at directly imposing the k-bit pattern into R2 and expecting it to *mean the same thing* (or anything) there seems as likely as tossing vials of DNA onto a floor and expecting them to square dance together. Even internally, the meanings of activation patterns are entirely location dependent; they require re-interpretation at every level. Neural networks don't store knowledge; they embody it.

Science programs (and science fiction movies) often tease the imagination with projections of future technologies that can copy all thoughts from a brain and load them into another brain, whether natural or artificial, thus producing a mental clone. Though the process of recoding a hundred trillion synapses is mind-boggling enough, it would not, in all likelihood, yield anything close to a cognitive copy. The patterns (and attractors in synaptic space that promote them) derive little meaning from their basic structure, but from their relationships to one another and (most relevant to cloning claims) the body and world. Brain, body and world are so tightly intertwined that isolated cerebral activation patterns reveal only an inkling of the actual knowledge and intelligence of the agent itself.

# References


[1] P. CHURCHLAND, *The Engine of Reason, the Seat of the Soul*, The MIT Press, Cambridge, MA, 1999.

[2] A. CLARK, *Mindware: An Introduction to the Philosophy of Cognitive Science*, The MIT Press, Cambridge, MA, 2001.

[3] T. DEACON, *The Symbolic Species: The Co-evolution of Language and the Brain*, W.W. Norton and Company, New York, 1998.

[4] G. EDELMAN AND G. TONONI, *A Universe of Consciousness*, Basic Books, New York, NY, 2000.

[5] D. HEBB, *The Organization of Behavior*, John Wiley and Sons, New York, NY, 1949.

[6] D. MARR, *A theory of cerebellar cortex*, Journal of Physiology, 202 (1969), pp. 437–470.

[7] ——, *Simple memory: A theory for archicortex*, Proceedings of The Royal Society of London, Series B, 262 (1971), pp. 23–81.

[8] J. MCCLELLAND, B. MCNAUGHTON, AND R. O'REILLY, *Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory*, Tech. Rep. PDP.CNS.94.1, Carnegia Mellon University, Mar. 1994.

[9] M. MEHTA, A. LEE, AND M. WILSON, *Role of experience and oscillations in transforming a rate code into a temporal code*, Nature, 417 (2002), pp. 741–746.

[10] F. RIEKE, D. WARLAND, R. DE RUYTER VAN STEVENINCK, AND W. BIALEK, *Spikes: exploring the neural code*, MIT Press, Cambridge, MA, USA, 1999.

[11] E. ROLLS AND A. TREVES, *Neural Networks and Brain Function*, Oxford University Press, New York, 1998.