## Chapter Revision History

The table notes major changes between revisions. Minor changes such as small clarifications or formatting changes are not noted.

| Version | Date | Changes | Principal Author(s) |
|---|---|---|---|
| 0.4 | | Initial release | A. Lavin, S. Ahmad, J. Hawkins |

## Sparse Distributed Representations

In this chapter we introduce Sparse Distributed Representations (SDRs), the fundamental form of information representation in the brain, and in HTM systems. We talk about several interesting and useful mathematical properties of SDRs and then discuss how SDRs are used in the brain.

## What is a Sparse Distributed Representation?

One of the most interesting challenges in AI is the problem of knowledge representation. Representing everyday facts and relationships in a form that computers can work with has proven to be difficult with traditional computer science methods. The basic problem is that our knowledge of the world is not divided into discrete facts with well-defined relationships. Almost everything we know has exceptions, and the relationships between concepts are too numerous and ill-defined to map onto traditional computer data structures.

Brains do not have this problem. They represent information using a method called Sparse Distributed Representations, or SDRs. SDRs and their mathematical properties are essential for biological intelligence. Everything the brain does and every principle described in this book is based on SDRs. SDRs are the language of the brain. The flexibility and creativity of human intelligence is inseparable from this representation method. Therefore, if we want intelligent machines to be similarly flexible and creative, they need to be based on the same representation method, SDRs.

An SDR consists of thousands of bits where at any point in time a small percentage of the bits are 1's and the rest are 0's. The bits in an SDR correspond to neurons in the brain, a 1 being a relatively active neuron and a 0 being a relatively inactive neuron. The most important property of SDRs is that each bit has meaning. Therefore, the set of active bits in any particular representation encodes the set of semantic attributes of what is being represented. The bits are not labeled (that is to say, no one assigns meanings to the bits), but rather, the semantic meanings of bits are learned. If two SDRs have active bits in the same locations, they share the semantic attributes represented by those bits. By determining the overlap between two SDRs (the equivalent bits that are 1 in both SDRs) we can immediately see how two representations are semantically similar and how they are semantically different. Because of this semantic overlap property, systems based on SDRs automatically generalize based on semantic similarity.

HTM theory defines how to create, store, and recall SDRs and sequences of SDRs. SDRs are not moved around in memory, like data in computers. Instead the set of active neurons, within a fixed population of neurons, changes over time. At one moment a set of neurons represents one thing; the next moment it represents something else. Within one set of neurons, an SDR at one point in time can associatively link to the next occurring SDR. In this way, sequences of SDRs are learned. Associative linking also occurs between different populations of cells (layer to layer or region to region). The meanings of the neuron encodings in one region are different than the meanings of neuron encodings in another region. In this way, an SDR in one modality, such as a sound, can associatively invoke an SDR in another modality, such as vision.

Any type of concept can be encoded in an SDR, including different types of sensory data, words, locations, and behaviors. This is why the neocortex is a universal learning machine. The individual regions of the neocortex operate on SDRs without "knowing" what the SDRs represent in the real world. HTM systems work the same way. As long as the inputs are in a proper SDR format, the HTM algorithms will work. In an HTM-based system, knowledge is inherent in the data, not in the algorithms.

To better understand the properties of SDRs it can be helpful to think about how information is typically represented in computers and the relative merits of SDRs versus the representation scheme used by computers. In computers, we represent information with bytes and words. For example, to represent information about a medical patient, a computer program might use one byte to store the patient's age and another byte to store the patient's gender. Data structures such as lists and trees are used to organize pieces of information that are related. This type of representation works well when the information we need to represent is well defined and limited in extent. However, AI researchers discovered that to make a computer intelligent, it needs to know a huge amount of knowledge, the structure of which is not well defined.

For example, what if we want our intelligent computer to know about cars? Think of all the things you know about cars. You know what they do, how to drive them, how to get in and out of them, how to clean them, ways they can fail, what the different controls do, what is found under the hood, how to change tires, etc. We know the shapes of cars and the sounds they make. If you just think about tires, you might recall different types of tires, different brands of tires, how they wear unevenly, the best way to rotate them, etc. The list of all the things you know about cars goes on and on.

Each piece of knowledge leads to other pieces of knowledge in an ever-expanding web of associations. For example, cars have doors, but other objects have doors too, such as houses, planes, mailboxes, and elevators. We intuitively know what is similar and different about all these doors and we can make predictions about new types of doors we have never seen before based on previous experience. We (our brains) find it easy to recall a vast number of facts and relationships via association. But when AI scientists try to encode this type of knowledge into a computer, they find it difficult.

In computers information is represented using words of 8, 32, or 64 bits. Every combination of 1's and 0's is used, from all 1's to all 0's, which is sometimes called a "dense" representation. An example of a dense representation is the ASCII code for letters of the alphabet. In ASCII the letter "m" is represented by:

01101101

Notice it is the combination of all eight bits that encodes "m", and the individual bits in this representation don't mean anything on their own. You can't say what the third bit means; the combination of all eight bits is required. Notice also that dense representations are brittle. For example, if you change just one bit in the ASCII code for "m" as follows:

01100101

you get the representation for an entirely different letter, "e". One wrong bit and the meaning changes completely.

There is nothing about 01101101 that tells you what it represents or what attributes it has. It is a purely arbitrary and abstract representation. The meaning of a dense representation must be stored elsewhere.

In contrast, as mentioned earlier, SDRs have thousands of bits. At every point in time a small percentage of the bits are 1's and the rest are 0's. An SDR may have 2,000 bits with 2% (or 40) being 1 and the rest 0, as visualized in Figure 1.

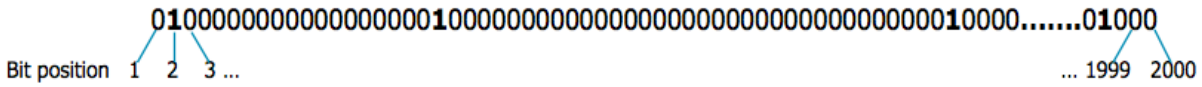


*Figure 1:* An SDR of 2000 bits, where only a few are ON (ones).

In an SDR, each bit has meaning. For example if we want to represent letters of the alphabet using SDRs, there may be bits representing whether the letter is a consonant or a vowel, bits representing how the letter sounds, bits representing where in the alphabet the letter appears, bits representing how the letter is drawn (i.e. open or closed shape, ascenders, descenders), etc. To represent a particular letter, we pick the 40 attributes that best describe that letter and make those bits 1. In practice, the meaning of each bit is learned rather than assigned; we are using letters and their attributes for illustration of the principles.

With SDRs the meaning of the thing being represented is encoded in the set of active bits. Therefore, if two different representations have a 1 in the same location we can be certain that the two representations share that attribute. Because the representations are sparse, two representations will not share an attribute by chance; a shared bit/attribute is always meaningful. As shown in Figure 2, simply comparing SDRs in this way tells us how any two objects are semantically similar and how they are different.

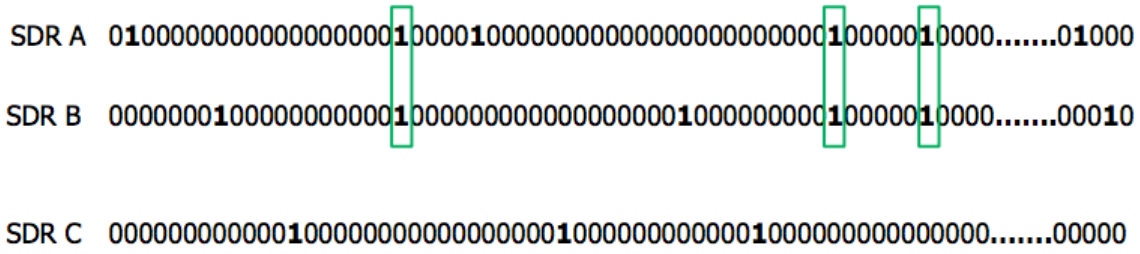


*Figure 2:* SDR A and SDR B have matching 1 bits and therefore have shared semantic meaning; SDR C has no matching bits or shared semantic meaning.

There are some surprising mathematical properties of SDRs that don't manifest in tradition computer data structures. For example, to store an SDR in memory, we don't have to store all the bits as you would with a dense representation. We only have to store the locations of the 1-bits, and surprisingly, we only have to store the locations of a small number of the 1-bits. Say we have an SDR with 10,000 bits, of which 2%, or 200, are 1's. To store this SDR, we remember the locations of the 200 1-bits. To compare a new SDR to the stored SDR, we look to see if there is a 1-bit in each of the 200 locations of the new SDR. If there is, then the new SDR matches the stored SDR. But now imagine we store the location of just 10 of the 1-bits randomly chosen from the original 200. To see if a new SDR matches the stored SDR, we look for 1-bits in the 10 locations. You might be thinking, "But wait, there are many patterns that would match the 10 bits yet be different in the other bits. We could easily have a false positive match!" This is true. However, the chance that a randomly chosen SDR would share the same 10 bits is extremely low; it won't happen by chance, so storing ten bits is sufficient. However, if two SDRs did have ten 1-bits in the same location but differed in the other bits then the two SDRs are semantically similar. Treating them as the same is a useful form of generalization. We discuss this interesting property, and derive the math behind it, later in this chapter.

Another surprising and useful property of SDRs is the union property, which is illustrated in Figure 3. We can take a set of SDRs and form a new SDR, which is the union of the original set. To form a union, we simply OR the SDRs together. The resulting union has the same number of bits as each of the original SDRs, but is less sparse. Forming a union is a one-way operation, which means that given a union SDR you can't say what SDRs were used to form the union. However, you can take a new SDR, and by comparing it to the union, determine if it is a member of the set of SDRs used to form the union. The chance of incorrectly determining membership in the union is very low due to the sparseness of the SDRs.

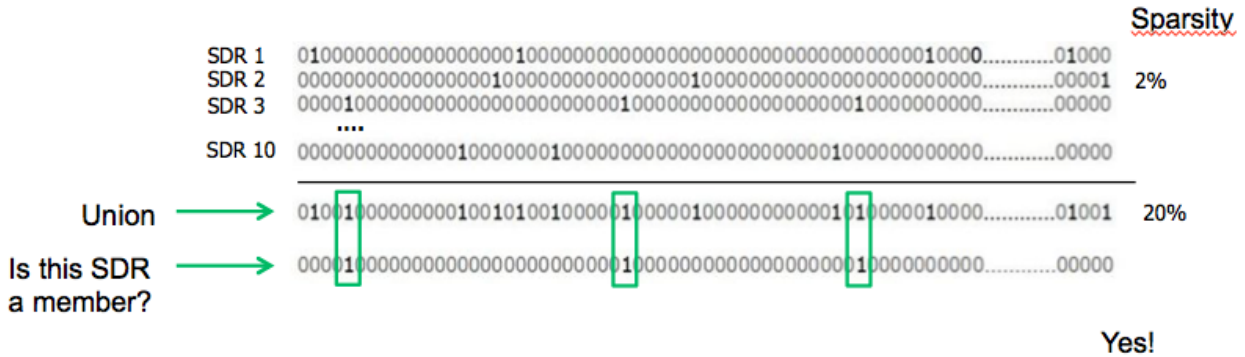


*Figure 3:* A union of 10 SDRs is formed by taking the mathematical OR of the bits. New SDR membership is checked by confirming 1 bits match. Note the union SDR is less sparse than the input SDRs.

These properties, and a few others, are incredibly useful in practice and get to the core of what makes brains different than computers. The following sections describe these properties and operations of SDRs in more detail. At the end of the chapter we discuss some of the ways SDRs are used in the brain and in HTMs.

## Mathematical Properties of SDRs

In this section, we discuss the following mathematical properties of SDRs with a focus on deriving fundamental scaling laws and error bounds:

- Capacity of SDRs and the probability of mismatches
- Robustness of SDRs and the probability of error with noise
- Reliable classification of a list of SDR vectors
- Unions of SDRs
- Robustness of unions in the presence of noise

These properties and their associated operations demonstrate the usefulness of SDRs as a memory space, which we illustrate in examples relevant to HTMs. In our analysis we lean on the intuitions provided by Kanerva (Kanerva, 1988 & 1997) as well as some of the techniques used for analyzing Bloom filters (Bloom, 1970). We start each property discussion with a summary description, and then go into the derivation of the mathematics behind the property. But first, here are some definitions of terms and notations we use in the following discussion and throughout the text. A more comprehensive list of terms can be found in the Glossary at the end of this book.

### Mathematical Definitions and Notation

**Binary vectors**: For the purposes of this discussion, we consider SDRs as **binary vectors**, using the notation $\mathbf{x} = [b_0, \ldots, b_{n-1}]$ for an SDR $x$. The values of each element are "0" or "1", for OFF and ON, respectively.

**Vector size**: In an SDR $\mathbf{x} = [b_0, \ldots, b_{n-1}]$, $n$ denotes the **size of a vector**. Equivalently, we say $n$ represents the total number of positions in the vector, or the total number of bits.

**Sparsity**: At any point in time, a fraction of the $n$ bits in vector $\mathbf{x}$ will be ON and the rest will be OFF. Let $s$ denote the percent of ON bits. Generally in sparse representations, $s$ will be substantially less than 50%.

**Vector cardinality**: Let $w$ denote the **vector cardinality**, which we define as the total number of ON bits in the vector. If the percent of ON bits in vector $\mathbf{x}$ is $s$, then $w_{\mathbf{x}} = s \times n = \|\mathbf{x}\|_0$.

**Overlap**: We determine the similarity between two SDRs using an **overlap score**. The overlap score is simply the number of ON bits in common, or in the same locations, between the vectors. If $\mathbf{x}$ and $\mathbf{y}$ are two SDRs, then the overlap can be computed as the dot product:

$$overlap(\mathbf{x}, \mathbf{y}) \equiv \mathbf{x} \cdot \mathbf{y}$$

Notice we do not use a typical distance metric, such as Hamming or Euclidean, to quantify similarity. With overlap we can derive some useful properties discussed later, which would not hold with these distance metrics.

**Matching**: We determine a **match** between two SDRs by checking if the two encodings overlap sufficiently. For two SDRs $\mathbf{x}$ and $\mathbf{y}$:

$$match(\mathbf{x}, \mathbf{y}|\theta) \equiv overlap(\mathbf{x}, \mathbf{y}) \geq \theta$$

If $\mathbf{x}$ and $\mathbf{y}$ have the same cardinality $w$, we can determine an exact match by setting $\theta = w$. In this case, if $\theta$ is less than $w$, the overlap score will indicate an **inexact match**.

Consider an example of two SDR vectors:

$$\mathbf{x} = [0100000000000000000010000000000000110000000]$$
$$\mathbf{y} = [1000000000000000000010000000000000110000000]$$

Both vectors have size $n = 40$, $s = 0.1$, and $w = 4$. The overlap between $x$ and $y$ is 3; i.e. there are three ON bits in common positions of both vectors. Thus the two vectors match when the threshold is set at $\theta = 3$, but they are not an exact match. Note that a threshold larger than either vector's cardinality – i.e., $\theta > w$ – implies a match is not possible.

## Capacity of SDRs and the Probability of Mismatches

To be useful in practice, SDRs should have a large capacity. Given a vector with fixed size $n$ and cardinality $w$, the number of unique SDR encodings this vector can represent is the combination $n$ choose $w$:

$$\binom{n}{w} = \frac{n!}{w!\,(n-w)!} \tag{1}$$

Note this is significantly smaller than the number of encodings possible with dense representations in the same size vector, which is $2^n$. This implies a potential loss of capacity, as the number of possible input patterns is much greater than the number of possible representations in the SDR encoding. Although SDRs have a much smaller capacity than dense encodings, in practice this is meaningless. With typical values such as $n = 2048$ and $w = 40$, the SDR representation space is astronomically large at $2.37\times10^{84}$ encodings; the estimated number of atoms in the observable universe is $\sim10^{80}$.

For SDRs to be useful in representing information we need to be able to reliably distinguish between encodings; i.e. SDRs should be distinct such that we don't confuse the encoded information. It is valuable then to understand the probability with which two random SDRs would be identical. Given two random SDRs with the same parameters, $x$ and $y$, the probability they are identical is

$$P(x = y) = 1/\binom{n}{w} \tag{2}$$

Consider an example with $n = 1024$ and $w = 2$. There are 523,776 possible encodings and the probability two random encodings are identical is rather high, i.e. one in 523,776. This probability decreases extremely rapidly as $w$ increases. With $w = 4$, the probability dives to less than one in 45 billion. For $n = 2048$ and $w = 40$, typical HTM values, the probability two random encodings are identical is essentially zero. Please note (2) reflects the false positive probability under exact matching conditions, not inexact matching used in most HTM models; this is discussed later in the chapter.

The equations above show that SDRs, with sufficiently large sizes and densities, have an impressive capacity for unique encodings, and there is almost no chance of different representations ending up with the same encoding.

## Overlap Set

We introduce the notion of an overlap set to help analyze the effects of matching under varying conditions. Let $\mathbf{x}$ be an SDR encoding of size $n$ with $w_x$ bits on. The overlap set of $\mathbf{x}$ with respect to $b$ is $\Omega_{\mathbf{x}}(n, w, b)$, defined as the set of vectors of size $n$ with $w$ bits on, that have exactly $b$ bits of overlap with $\mathbf{x}$. The number of such vectors is $|\Omega_{\mathbf{x}}(n, w, b)|$, where $|\cdot|$ denotes the number of elements in a set. Assuming $b \leq w_x$ and $b \leq w$,

$$|\Omega_x(n, w, b)| = \binom{w_x}{b} \times \binom{n - w_x}{w - b} \tag{3}$$

The first term in the product of (3) the number of subsets of $x$ with $b$ bits ON, and the second term is the number of other patterns containing $n - w_x$ bits, of which $w - b$ bits are ON.

The overlap set is instructive as to how we can compare SDRs reliably; i.e. not get false negatives or false positives, even in the presence of significant noise, where noise implies random fluctuations of ON/OFF bits. In the following sections we explore the robustness of SDRs in the presence of noise using two different concepts, inexact matching and subsampling.

## Inexact Matching

If we require two SDRs to have an exact match (i.e. $\theta = w$) then even a single bit of noise in either of the SDRs' ON bits would generate a false negative where we fail to recognize matching SDRs. In general we would like the system to be tolerant to changes or noise in the input. That is, rarely would we require exact matches, where $\theta = w$. Lowering $\theta$ allows us to use inexact matching, decreasing the sensitivity and increasing the overall noise robustness of the system. For example, consider SDR vectors $\mathbf{x}$ and $\mathbf{x}'$, where $\mathbf{x}'$ is $\mathbf{x}$ corrupted by random noise. With $w = 40$ and $\theta$ lowered to 20, the noise can flip 50% of the bits (ON to OFF and vice-versa) and still match $\mathbf{x}$ to $\mathbf{x}'$.

Yet increasing the robustness comes with the cost of more false positives. That is, decreasing $\theta$ also increases the probability of a false match with another random vector. There is an inherent tradeoff in these parameters, as we would like the chance of a false match to be as low as possible while retaining robustness.

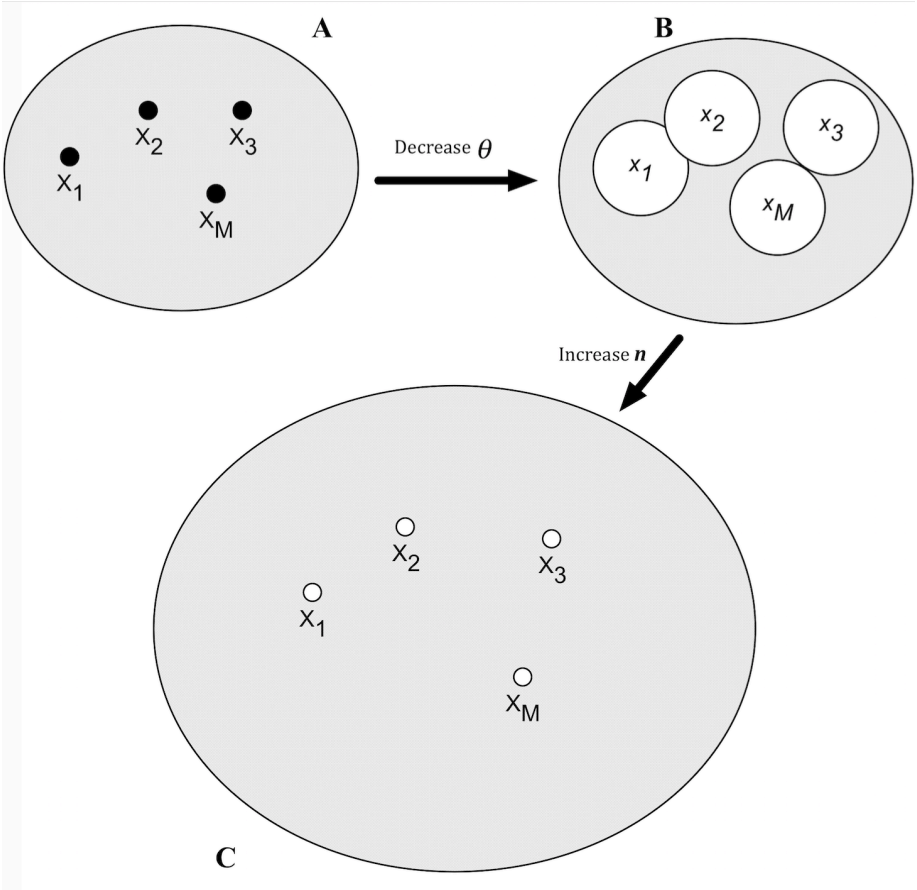


*Figure 4:* This figure illustrates the conceptual difference of lowering the match threshold θ. The large grey areas represent the space of all possible SDRs, where the elements $x_1, x_2, \ldots x_M$ are individual SDRs within the space. In space A we see the exact matching scenario, where SDRs are single points in the space. Notice $x_1, x_2, \ldots x_M$ in space B are now larger circles within the space, implying more SDRs will match to them in B than in A. That is, the ratio of white to grey becomes much larger as you decrease θ. Spaces A and B are the same size because they have a fixed n. If we increase n—i.e. increase the space of possible SDRs—the ratio of white to grey becomes smaller, as shown in space C. The transitions from A to B to C illustrate the tradeoff between the parameters θ and n: decreasing θ gives you more robustness, but also increases your susceptibility to false matches, but increasing n mitigates this effect.

With appropriate parameter values the SDRs can have a large amount of noise robustness with a very small chance of false positives. To arrive at the desired parameter values we need to calculate the false positive likelihood as a function of the matching threshold.

Given an SDR encoding $\mathbf{x}$ and another random SDR $\mathbf{y}$, both with size $n$ and cardinality $w$, what is the probability of a false match, i.e. the chance the $overlap(\mathbf{x}, \mathbf{y}) \geq \theta$? A match is defined as an overlap of $\theta$ bits or greater, up to $w$. With $\binom{n}{w}$ total patterns, the probability of a false positive is:

$$fp_w^n(\theta) = \frac{\sum_{b=\theta}^{w} |\Omega_x(n, w, b)|}{\binom{n}{w}} \tag{4}$$

What happens when $\theta = w$, or an exact match? The numerator in (4) evaluates to 1, and the equation reduces to (2).

To gain a better intuition for (4), again suppose vector parameters $n = 1024$ and $w = 4$. If the threshold is $\theta = 2$, corresponding to 50% noise, then the probability of an error is one in 14,587. That is, with 50% noise there is a significant chance of false matches. If $w$ and $\theta$ are increased to 20 and 10, respectively, the probability of a false match decreases drastically to less than one in $10^{13}$! Thus, with a relatively modest increase in w and θ, and holding n fixed, SDRs can achieve essentially perfect robustness with up to 50% noise. Figure 5 illustrates this for HTM values used in practice.

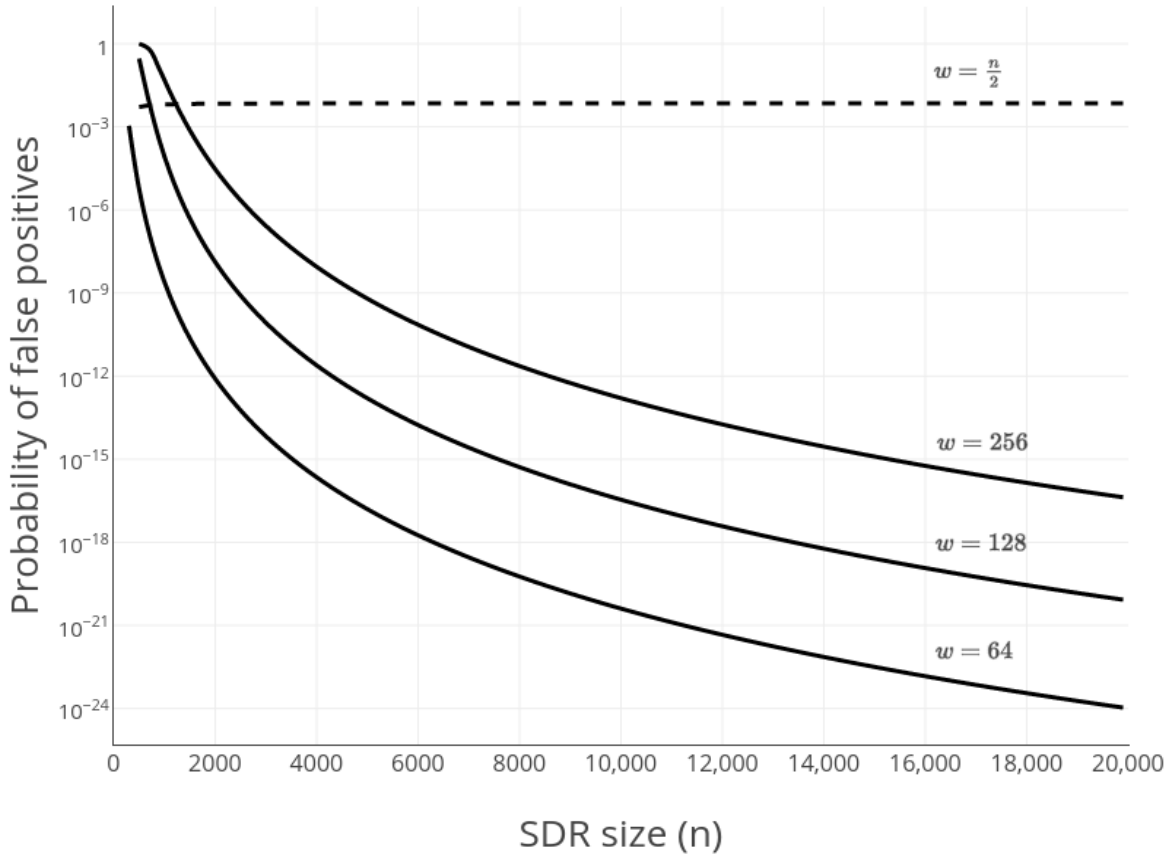


*Figure 5*: This plot illustrates the behavior of Eq. 4. The three solid curves show the rapid drop in error rates (i.e. probability of false positives) as you increase the SDR size n. Each curve represents a different number of ON bits w, and a constant 50% match threshold θ. For all three curves the error drops faster than exponentially as n increases, becoming essentially 0 once $n > 2000$. The dashed line shows the error rate when half of the bits in the SDR are ON. Notice this line maintains a relatively high error rate, implying it is not possible to get robust recognition with a non-sparse representation; both sparsity and high-dimensionality are required to achieve low error rates. Use this plot interactively online: *placeholder for plot.ly link*

## Subsampling

An interesting property of SDRs is the ability to reliably compare against a subsampled version of a vector. That is, recognizing a large distributed pattern by matching a small subset of the active bits in the large pattern. Let $\mathbf{x}$ be an SDR and let $\mathbf{x}'$ be a subsampled version of $\mathbf{x}$, such that $w_{x'} \leq w_x$. It's self-evident the subsampled vector $\boldsymbol{x}'$ will always match $\mathbf{x}$, provided $\theta \leq w_{x'}$ that is. However, as you increase the subsampling the chance of a false positive increases.

What is the probability of a false match between $\mathbf{x}'$ and a random SDR $\mathbf{y}$? Here the overlap set is computed with respect to the subsample $\mathbf{x}'$, rather than the full vector $\mathbf{x}$. If $b \leq w_{x'}$ and $w_{x'} \leq w_y$ then the number of patterns with exactly $b$ bits of overlap with $x'$ is:

$$\left|\Omega_{x'}(n, w_y, b)\right| = \binom{w_{x'}}{b} \times \binom{n - w_{x'}}{w_y - b} \tag{6}$$

Given a threshold $\theta \leq w_{x'}$, the chance of a false positive is then:

$$fp^n_{w_y}(\theta) = \frac{\sum_{b=\theta}^{w_{x'}} \left|\Omega_{x'}\left(n, w_y, b\right)\right|}{\binom{n}{w_y}} \tag{7}$$

Notice (6) and (7) differ from (3) and (4), respectively, only in the vectors being compared. That is, subsampling is simply a variant of the inexact matching properties discussed above.

For instance, suppose $n = 1024$ and $w_y = 8$. Subsampling half the bits in $x$ and setting the threshold to two (i.e. $w_{x'} = 4, \theta = 2$), we find the probability of an error is one in 3,142. However, increasing $w_y$ to 20 and the relevant parameter ratios fixed (i.e. $w_{x'} = 10, \theta = 5$) the chance of a false positive drops precipitously to one in 2.5 million. Increasing $n$ to 2048, $w_y = 40$, $w_{x'} = 20$, and $\theta = 10$, more practical HTM parameter values, the probability of a false positive plummets to better than one in $10^{12}$! This is remarkable considering that the threshold is about 25% of the original number of ON bits. Figure 6 illustrates this reliability in subsampling for varying HTM parameters.

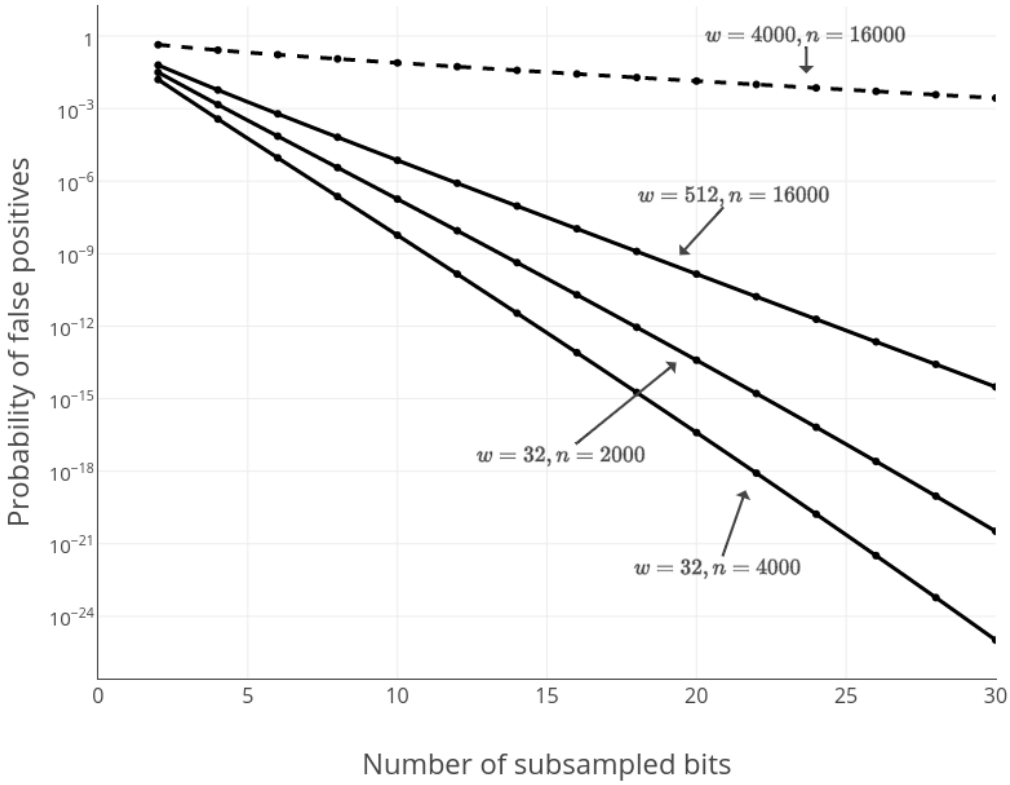


*Figure 6:* This plot illustrates the behavior of Eq. 6, where we can see the error rates (i.e. probability of false positives) as a function of the size (i.e. number of bits) of the subsampling. The three solid curves represent a few dimensionalities and sparsities, showing an exponential improvement in error rates as the number of subsampled bits increases. With sufficiently high dimensionality and sparse activity, subsampling values between 15 and 25 can lead to very low error rates. Conversely, the dashed line represents the error rates for a relatively dense representation (25% total ON bits); the error remains high, despite the high dimensionality. Use this plot interactively online: *placeholder for plot.ly link*

With practical parameters, it is clear SDRs offer minimal probabilities of false positives. The properties of subsampling and inexact matching allow us to use SDRs as reliable mechanisms for classification, discussed in the next section.

## Reliable Classification of a List of SDR Vectors

A useful operation with SDRs is the ability to classify vectors, where we can reliably tell if an SDR belongs to a set of similar SDRs. We consider a form of classification similar to nearest neighbor classification. Let $X$ be a set of $M$ vectors, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$, where each vector $\boldsymbol{x}_i$ is an SDR. Given any random SDR $\boldsymbol{y}$ we classify it as belonging to this set as follows:

$$\exists_{\boldsymbol{x}_i \in X} match(\boldsymbol{x}, \mathbf{y}) = true \tag{8}$$

How reliably can we classify a vector corrupted by noise? More specifically, if we introduce noise in a vector $\boldsymbol{x}_i$ by toggling ON/OFF a random selection $t$ of the $n$ bits, what is the likelihood of a false positive classification? Assuming $t \leq w - \theta$, there are no false negatives in this scheme, only false positives. Thus the question of interest becomes what is the probability the classification of a random vector $\boldsymbol{y}$ is a false positive? Since all vectors in $X$ are unique with respect to matching, the probability of a false positive is by:

$$fp_x(\theta) = 1 - (fp_w^n(\theta))^M \quad (9)$$

The false positive probability of an individual overlap is extremely small, so it is more practical to use the following bound:

$$fp_x(\theta) \leq Mfp_w^n(\theta) \quad (10)$$

Consider for example $n = 64$ and $w = 3$ for all vectors. If $\theta = 2$, 10 vectors can be stored in your list, and the probability of false positives is about one in 22. Increasing $w$ to 12 and $\theta$ to eight, maintaining the ratio $\frac{\theta}{w} = \frac{2}{3}$, the chance of a false positive drops to about one in 2363. Now increase the parameters to more realistic values: $n = 1024$, $w = 21$, and $\theta = 14$ (i.e. two-thirds of $w$). In this case the chance of a false positive with 10 vectors plummets to about one in $10^{20}$. In fact, with these parameters the false positive rate for storing a billion vectors is better than one in $10^{12}$!

This result illustrates a remarkable property of SDRs. Suppose a large set of patterns is encoded in SDRs, and stored in a list. A massive number of these patterns can be retrieved almost perfectly, even in the presence of a large amount of noise. The main requirement being the SDR parameters n, w, and t need to be sufficiently high. As illustrated in the above example, with low values such as $n = 64$ and $w = 3$ your SDRs are unable to take advantage of these properties.

## Unions of SDRs

One of the most fascinating properties of SDRs is the ability to reliably store a set of patterns in a single fixed representation by taking the OR of all the vectors. We call this the union property. To store a set of $M$ vectors, the union mechanism is simply the Boolean OR of all the vectors, resulting in a new vector $\boldsymbol{X}$. To determine if a new SDR $\mathbf{y}$ is a member of the set, we simply compute the $match(\boldsymbol{X}, \mathbf{y})$.

$$\boldsymbol{x}_1 = [01000000000010000000 \ldots 010]$$
$$\boldsymbol{x}_2 = [00000000000000000010 \ldots 100]$$
$$\boldsymbol{x}_3 = [10100000000000000000 \ldots 010]$$
$$\vdots$$
$$\boldsymbol{x}_{10} = [00000000000000110000 \ldots 010]$$

$$\boldsymbol{X} = \boldsymbol{x}_1 OR \boldsymbol{x}_2 OR, \ldots, \boldsymbol{x}_{10}$$

$$\boldsymbol{X} = [11100000000110110000 \ldots 110]$$

$$\boldsymbol{y} = [10000000000001000000 \ldots 001]$$

$$\therefore match(\boldsymbol{X}, \boldsymbol{y}) = 1$$

*Figure 7 (top)* Taking the OR of a set of M SDR vectors results in the union vector $\mathbf{X}$. With each individual vector having a total of 2% ON bits, and $M = 10$, it follows that the percentage of ON bits in $\mathbf{X}$ is at most 20%. The logic is straightforward: if there is no overlap within the set of vectors, each ON bit will correspond to its own ON bit in the union vector, summing the ON bit percentages. With overlap, however, ON bits will be shared in the union vector, resulting in a lower percentage of ON bits. *(bottom)* Computing the $\text{match}(\mathbf{X}, \mathbf{y})$ reveals if $\mathbf{y}$ is a member of the union set $\mathbf{X}$ – i.e. if the ON positions in $\mathbf{y}$ are ON in $\mathbf{X}$ as well.

The advantage of the union property is that a fixed-size SDR vector can store a dynamic set of elements. As such, a fixed set of cells and connections can operate on a dynamic list. It also provides an alternate way to do classification. In HTMs, unions of SDRs are used extensively to make temporal predictions, for temporal pooling, to represent invariances, and to create an effective hierarchy. However, there are limits on the number of

vectors that can be reliably stored in a set. That is, the union property has the downside of increased potential for false positives.

How reliable is the union property? There is no risk of false negatives; if a given vector is in the set, its bits will all be ON regardless of the other patterns, and the overlap will be perfect. However, the union property increases the likelihood of false positives. With the number of vectors, $M$, sufficiently large, the union set will become saturated with ON bits, and almost any other random vector will return a false positive match. It's essential to understand this relationship so we can stay within the limits of the union property.

Let us first calculate the probability of a false positive assuming exact matches, i.e. $\theta = w$. In this case, a false positive with a new random pattern $y$ occurs if all of the bits in $y$ overlap with $X$. When $M = 1$, the probability any given bit is OFF is given by $1 - s$, where $s = \frac{w}{n}$. As $M$ grows, this probability is given by:

$$p_0 = (1 - s)^M \tag{11}$$

After $M$ union operations, the probability a given bit in X is ON is $1 - p_0$. The probability of a false positive, i.e. all $w$ bits in $y$ are ON, is therefore:

$$p_{fp} = (1 - p_0)^w = (1 - s^M)^w \tag{12}$$

The technique used to arrive at (12) is similar to the derivation of the false positive rate for Bloom filters (Bloom, 1970; Broder and Mitzenmacher, 2004). The slight difference is that in Bloom filters each bit is chosen independently, i.e. with replacement. As such, a given vector could contain less than $w$ ON bits. In this analysis we guarantee that there are exactly $w$ bits ON in each vector.

The above derivation assures us, under certain conditions, we can store SDRs as unions without much worry for false positives. For instance, consider SDR parameters $n = 1024$ and $w = 2$. Storing $M = 20$ vectors, the chance of a false positive is about one in 680. However, if $w$ is increased to 20, the chance drops dramatically to about one in 5.5 billion. This is a remarkable feature of the union property. In fact, if increasing $M$ to 40, the chance of an error is still better than $10^{-5}$.

To gain an intuitive sense of the union property, the expected number of ON bits in the union vector is $n(1 - p_0)$. This grows slower than linearly, as shown in (12); additional union operations contribute fewer and fewer ON bits to the resulting SDR. Consider for instance $M = 80$, where 20% of the bits are 0. When we consider an additional vector with 40 ON bits, there is a reasonable chance it will have at least one bit among this 20, and hence it won't be a false positive. That is, only vectors with all of their w bits amongst the 80% ON are false positives. As we increase n and w, the number of patterns that can OR together reliably increases substantially. As illustrated in Figure 8, if n and w are large enough, the probably of false positives remains acceptably low, even as M increases.

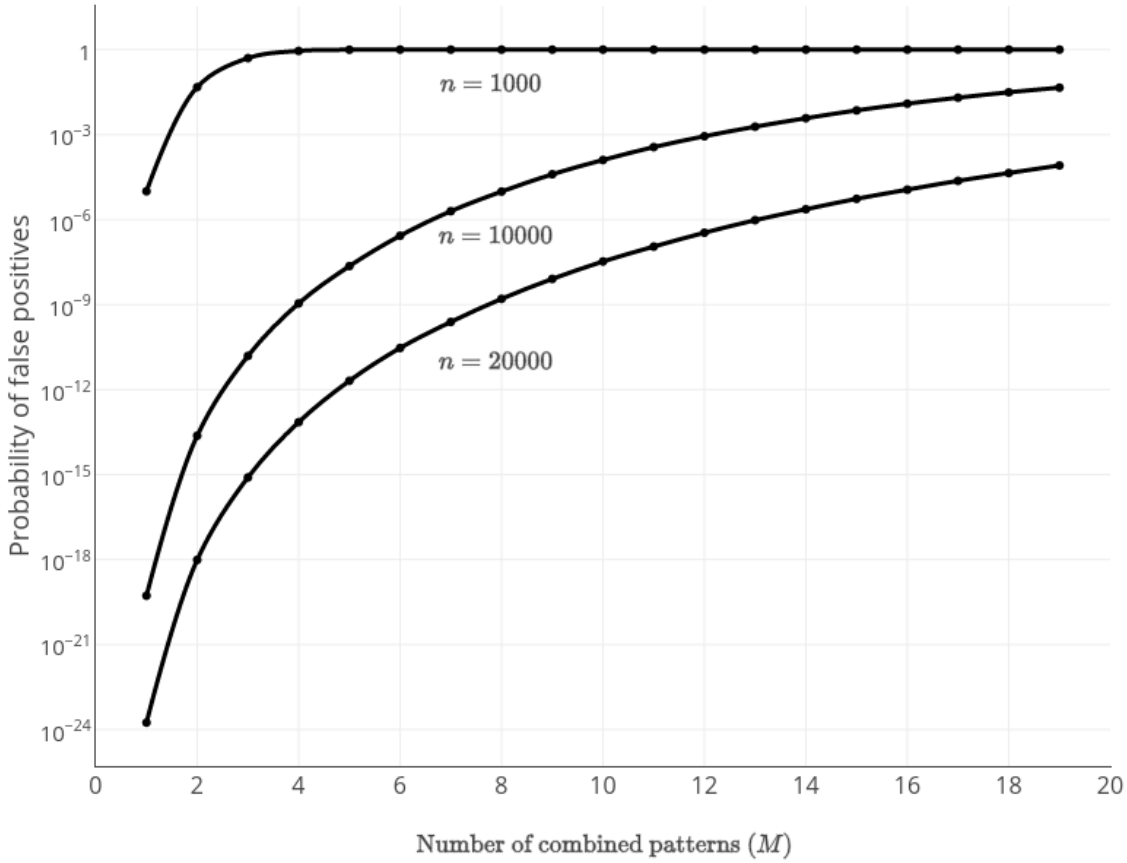


*Figure 8:* This plot shows the classification error rates (i.e. probability of false positives when matching to a union set of SDRs) of Eq. 12. The three lines show the calculated values for a few SDR dimensionalities, where w = 200. We see the error increases monotonically with the number of patterns stored. More importantly the plot shows that the size of the SDR is a critical factor: a small number of bits (1000) leads to relatively high error rates, while larger vectors (10,000+) are much more robust. Use this plot interactively online: *placeholder for plot.ly link*

## Robustness of Unions in the Presence of Noise

As mentioned above, the expected number of ON bits in the union vector $\boldsymbol{X}$ is $\widetilde{w}_X = n(1 - p_0)$, where we use the tilde notation to represent a union vector. Assuming $n \geq \widetilde{w}_X \geq w$, we can calculate the expected size of the overlap set:

$$E[|\Omega_X(n, w, b)|] = \binom{\widetilde{w_X}}{b} \times \binom{n - \widetilde{w_X}}{w - b} \tag{13}$$

For a match we need an overlap of $\theta$ or greater bits (up to $w$). The probability of a false match is therefore:

$$\varepsilon \approx \frac{\sum_{b=\theta}^{w} |\Omega_X(n, w, b)|}{\binom{n}{w}} \tag{14}$$

Notice (14) is an approximation of the error, as we're working with the expected number of ON bits in $\boldsymbol{X}$[1].

As you would expect, the chance of error increases as the threshold is lowered, but the consequences of this tradeoff can be mitigated by increasing $n$. Suppose $n = 1024$ and $w = 20$. When storing $M = 20$ vectors, the chance of a false positive when using perfect matches is about one in 5 billion. Using a threshold of 19 increases the false positive rate to about one in 123 million. When $\theta = 18$, the chance increases to one in 4 million. However, if you increase $n$ to 2048 with $\theta = 18$, the false positive rate drops dramatically to one in 223 billion! This example illustrates the union property's robustness to noise, and is yet another example of our larger theme: small linear changes in SDR numbers can cause super-exponential improvements in the error rates.

### Computational Efficiency

Although SDR vectors are large, all the operations we've discussed run in time linear with the number of ON bits[2]. That is, the computations are dependent on the number of ON bits, $w$, not on the size of the vector, $n$. There are no loops or optimization processes required, enabling fast calculations with SDR vectors. This would not be the case, however, with more standard distance metrics, which are typically $O(n)$. For HTM systems this is important since in practice $w \ll n$.

In the following section we discuss how the brain takes advantage of the mathematical properties of SDRs, and how this is manifested in HTM systems.

## SDRs in the Brain and in HTM Systems

How do sparse distributed representations and their mathematical principles relate to information storage and retrieval in the brain? In this section we will list some of the ways SDRs are used in the brain and the corresponding components of HTM theory.

### Neuron Activations are SDRs

If you look at any population of neurons in the neocortex their activity will be sparse, where a low percentage of neurons are highly active (spiking) and the remaining neurons are inactive or spiking very slowly. SDRs represent the activity of a set of neurons, with 1- and 0-bits representing active and relatively inactive neurons, respectively. All the functions of an HTM system are based on this basic correspondence between neurons and SDRs.

The activity of biological neurons is more complex than a simple 1 or 0. Neurons emit spikes, which are in some sense a binary output, but the frequency and patterns of spikes varies considerably for different types of neurons and under different conditions. There are differing views on how to interpret the output of a neuron. On one extreme are arguments that the timing of each individual spike matters; the inter-spike time encodes information. Other theorists consider the output of a neuron as a scalar value, corresponding to the rate of spiking. However, it has been shown that sometimes the neocortex can perform significant tasks so quickly that the neurons involved do not have enough time for even a second spike from each neuron to contribute to the completion of the task. In these tasks inter-spike timing and spike rate can't be responsible for encoding information. Sometimes neurons start spiking with a mini-burst of two to four spikes in quick succession before settling into a steadier rate of spiking. These mini-bursts can invoke long lasting effects in the post-synaptic cells, i.e. the cells receiving this input.

HTM theory says that a neuron can be in one of several states:

- Active (spiking)
- Inactive (not spiking or very slowly spiking)
- Predicted (depolarized but not spiking)
- Active after predicted (a mini-burst followed by spiking)

---

[1] The assumption behind this approximation being the actual number of ON bits is the same as the expected number of ON bits.

[2] The computations are O(w) and independent of the size of the vector, $n$.

These HTM neuron states differ from those of other neural network models and this deserves some explanation. First, it is well established that some biological neurons spike at different rates depending on how well their input matches their ideal "receptive field". However, individual neurons are never essential to the performance of the network; the population of active cells is what matters most, and any individual neuron can stop working with little effect to the network. It follows that variable spiking rate is non-essential to neocortical function, and thus it is property that we choose to ignore in the HTM model. We can always compensate for lack of variable encoding by using more bits in an SDR. All the HTM implementations created to date have worked well without variable rate encoding. The second reason for avoiding rate encoding is that binary cell states make software and hardware implementations much simpler. HTM systems require almost no floating point operations, which are needed in any system with rate encoding. Hardware for implementing HTM will be far simpler without the need for floating point math. There is an analogy to programmable computers. When people first started building programmable computers, some designers advocated for decimal logic. Binary logic won out because it is much simpler to build.

Although HTM neurons don't have variable rate outputs, they do incorporate two new states that don't exist in other theories. When a neuron recognizes a pattern on its distal or apical dendrites, the dendrite generates a local NMDA spike, which depolarizes the cell body without generating a somatic spike. In HTM theory, this internal depolarized state of the cell represents a prediction of future activity and plays a critical role in sequence memory. And finally, under some conditions a neuron will start firing with a mini-burst of spikes. One condition that can cause a mini-burst is when a cell starts firing from a previously depolarized state. A mini-burst activates metabotropic receptors in the post-synaptic cell, which leads to long lasting depolarization and learning effects. Although the neuroscience around mini-bursts is not settled, HTM theory has a need for the system acting differently when an input is predicted from when it isn't predicted. Mini-bursts and metabotropic receptors fill that role. By invoking metabolic effects the neuron can stay active after its input ceases and can exhibit enhanced learning effects. These two states, predicted (depolarized) and active after predicted (mini-burst) are excellent examples of how HTM theory combines top-down system-level theoretical needs with detailed biological detail to gain new insights into biology.

*Figure 9:* Visualization of an SDR in the brain *(placeholder)*

## Neural Predictions as Unions of SDRs

HTM sequence memory makes multiple simultaneous predictions of what will happen next. This capability is an example of the union property of SDRs. The biological equivalent is multiple sets of cells (SDRs) being depolarized at the same time. Because each representation is sparse, many predictions can be made simultaneously. For example, consider a layer of neurons implementing an HTM sequence memory. If 1% of the neurons in the layer are active and lead to 20 different predictions, then about 20% of the neurons would be in the depolarized/predictive state. Even with 20% of neurons depolarized, the system can reliably detect if one of the predictions occurs or not. As a human, you would not be consciously aware of these predictions because the predicted cells are not spiking. However, if an unexpected input occurs, the network detects it, and you become aware something is wrong.

## Synapses as a Means for Storing SDRs

Computer memory is often called "random access memory." A byte is stored in a memory location on a hard drive or on a memory chip. To access the byte's value you need to know its address in memory. The word "random" means that you can retrieve information in any order as long as you have the address of the item you want to retrieve. Memory in the brain is called "associative memory." In associative memory, one SDR is linked to another SDR which is linked to another, etc. SDRs are recalled through "association" with other SDRs. There is no centralized memory and no random access. Every neuron participates in both forming SDRs and in learning the associations.

Consider a neuron that we want to recognize a particular pattern of activity. To achieve this, the neuron forms synapses to the active cells in the pattern. As described above, a neuron only needs to form a small number of synapses, typically fewer than twenty, to accurately recognize a pattern in a large population of cells as long as the pattern is sparse. Forming new synapses is the basis of almost all memory in the brain.

But we don't want just one neuron to recognize a pattern; we want a set of neurons to recognize a pattern. This way one SDR will invoke another SDR. We want SDR pattern "A" to invoke SDR pattern "B." This can be achieved if each active cell in pattern "B" forms twenty synapses to a random sample of the cells in pattern "A."

If the two patterns “A” and “B” are subsequent patterns in the same population of neurons then the learned association from “A” to “B” is a transition, and forms the basis of sequence memory. If the patterns “A” and “B” are in different populations of cells then pattern “A” will concurrently activate pattern “B.” If the neurons in pattern “A” connect to the distal synapses of neurons in pattern “B,” then the “B” pattern will be predicted. If the neurons in pattern “A” connect to the proximal synapses of neurons in pattern “B,” then the “B” pattern will consist of active neurons.

All associative memory operations use the same basic memory mechanism, the formation of new synapses on the dendrite segments of neurons. Because all neurons have dozens of dendrite segments and thousands of synapses, each neuron does not just recognize one pattern but dozens of independent patterns. Each neuron participates in many different SDRs.

## Conclusion

SDRs are the language of the brain, and HTM theory defines how to create, store, and recall SDRs and sequences of SDRs. In this chapter we learned about powerful mathematical properties of SDRs, and how these properties enable the brain, and HTM, to learn sequences and to generalize.

## References


Kanerva, P. (1988). Sparse Distributed Memory. Bradford Books of MIT Press.

Kanerva, P. (1997). Fully distributed representation. Proceedings of 1997 Real World Computing Symposium (RWC ’97, Tokyo, Jan. 1997), pp. 358–365. Tsukuba-city, Japan: Real World Computing Partnership.

Bloom, B.H. (1970). Space/ Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, Volume 13, Number 7, July 1970, pp. 422-426.

Broder, A., & Mitzenmacher, M. (2004). Network Applications of Bloom Filters, A Survey. Internet Mathematics, Volume 1, No. 4, pp. 485-509.

Ahmad, S., & Hawkins, J. (2016). How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites. *arXiv*, 1601.00720. Neurons and Cognition; Artificial Intelligence. Retrieved from http://arxiv.org/abs/1601.00720