Learning Representations

Iyngkarran Kumar

Summer 2022

Contents

1 Introduction

This document contains a collection of topics that I have explored with the aim of developing an understanding of how intelligent systems form representations of their environment. In this writing, the term representation (as well as feature vector, abstraction and encoding) broadly refers to a low dimensional summary of some physical phenomenon that captures most of the information present in the raw, high dimensional description. A common example used throughout this text is the pressure, volume, temperature (p, V, T) description of an ideal gas, which, while low dimensional, contains sufficient information to determine all macroscopic properties of the gas. That's much easier than specifying the positions and velocities of trillions upon trillions of molecules!

The following questions have motivated this study:

1. By what process do animal brains come to represent knowledge of their environment as a complex series of electrical impulses? How, in animal brains, are complex objects such as trees or mountains represented, given the high dimensional raw input that the brain receives when seeing such objects?

2. Some concepts are more complex than others (a tree is clearly harder to define than simple two and three dimensional shapes). Can we speak of this complexity more precisely? Highly abstract concepts are usually described *recursively*, i.e we describe them in terms of slightly less abstract objects, which are in turn given in terms of even less abstract objects, all the way down to the atomic level. Might there exist more objective measures of concept complexity?

3. To what extent can we abstract a neural network into interactions between sub networks? Do different subnetworks of the network learn representations of different concepts? If so, how can we identify these networks?

This document seeks to lay out the basic concepts that may provide frameworks for addressing the questions above. Definitions of abstractions often refer to notions of summarising the *information* that is present in some physical phenomenon; as a result, in Section 2 we introduce some key ideas of classical information theory. Section 3 follows by detailing representation learning in present day machine learning, with Section 4 moving the discussion to the field of computational neuroscience; specifically, methods in which animal brains encode information about the surrounding environment.

Note - All of the following sections assume a knowledge of linear algebra, calculus and probability theory up to and including a first undergraduate course in the subjects.

2 Information theory

Let us begin with the basic foundations of classical information theory. We'll be looking at Claude Shannon's 1948 formulation, rather than later developments such as *algorithmic* information theory.

The world is full of uncertainty; we may be unsure whether it will rain tomorrow, we may be unsure whether we switched the kitchen light off, or we may be unsure how we will perform in midterm exams. Even at the smallest physical scales, modern physics tells us that nature is *inherently uncertain* - no scientist in the world, if given the location of a proton, can tell you exactly where that proton will be in 10 seconds. The best they can do is provide a probability distribution over possible locations. God really does play dice!

The mathematics of probability theory gives us a precise way to talk about uncertainty. Let us define an ensemble $X = (x, A_X, P_X)$, where x is the value of a random variable, A_X is the set of possible values that this variable can take, referred to as the *alphabet*, and P_X is the set of probabilities assigned to each of these possible outcomes. A simple example of an ensemble W that captures our uncertainty about the weather tomorrow could be: $W = (w, \{sunny, raining, snowing\}, \{0.26, 0.59, 0.15\})$, where $A_W = \{sunny, raining, snowing\}$ and $P_W = \{0.26, 0.59, 0.15\}$. Of course, A_W could be a continuous space, in which case P_W would be a probability density function (PDF), rather than a probability mass function (PMF).

2.1 Measuring information

Once we have defined an ensemble, we can begin thinking about how to quantify the information gained when observing a particular event x occurring. Our formalisation should satisfy the following intuitive desiderata:

- **Finding out that an event with probability *one* occurred should yield *zero* information**. Imagine that we have Alice and Bob playing a game, in which Bob has chosen a certain animal (say, a horse), and Alice is trying to guess the animal with hints given by Bob. It would not be helpful for Bob to tell Alice: 'this animal evolved on planet Earth', because this is true for *all* animals that we know of. There is zero information in this message.

- **Events that are less likely yield more information**. Clearly, Bob telling Alice that 'the animal has four legs, hooves, has been domesticated by humans and was utilised in the military until the 20th century' is more helpful than Bob telling Alice 'the animal has four legs', because there are *far fewer* animals that fit the first description. The less likely an event is, the more information it yields.

- **The information measure should *not* be bounded above**. Let's change the guessing game that Alice and Bob are playing. Instead, say that Bob knows the position of an electron[1] in the observable universe (with respect to the Earth), and Alice is trying to guess this position. Now, whatever the description that Bob gives for the position of the electron, we can always imagine

[1]Ignore quantum effects here - treat the electron as a point particle.

a *slightly more informative* description. For example, if Bob gave the position of the electron to n decimal places, Alice could ask him to give it to $n+1$ decimal places instead. Therefore, given that there are instances in which there is no upper limit to informativeness, our measure should have no upper bound.

- **Additivity axiom for independent events -** Consider two random variables that are clearly dependent, such as the average annual temperature in a city (denoted X), and the latitude of the city (denoted Y). Evidently, receiving some information about the state of X will also carry some information about Y (if we find out the annual temperature is in the range of $30-40°C$, we quickly rule out many northern and southern latitudes). Conversely, if X and Y are completely independent, finding out that $X = x$ carries no information about Y, and observing that $Y = y$ provides no indication about the value of X. Therefore, for independent events, we need our information measure, $I()$, to satisfy $I(X = x, Y = y) = I(X = x) + I(Y = y)$.

Enforcing the four conditions above, we see that the family of functions $-log_n(x), n > 1$ can serve as a measure of information. For simplicity, we choose $n = 2$, and arrive at the following definition: **The information content of an event x, that has probability $p(x)$ of occurring, is $I(x) = -log_2(p(x))$.**

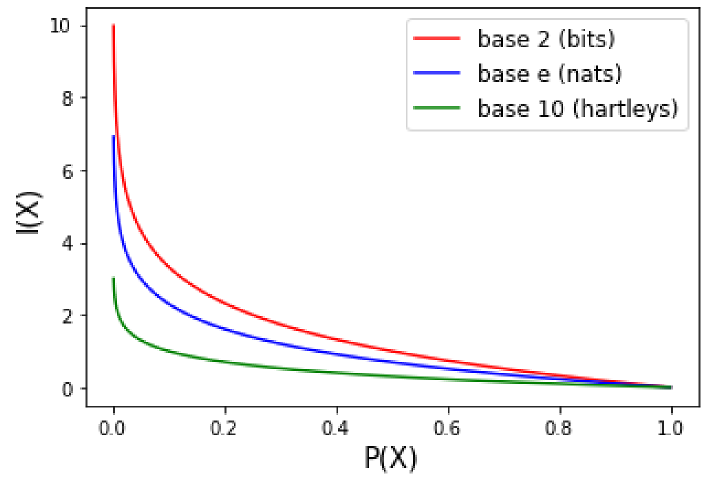


Figure 1: $-log_n()$ graphs for n= $(2, e, 10)$. The choice of log_2 is not required, and one may sometimes see information content measured in *nats* or *hartleys*, though this is rather uncommon.

This definition gives us a simple interpretation of what 1 unit (or *bit*) of information is, in a special case. Let's return to the animal guessing game. Imagine now that the animal that Alice is guessing could be one of 64 possible species - we call this set of 64 possible outcomes Alice's *hypothesis space*. Say that 32 of the animals in the hypothesis space are mammals, the other 32 are reptiles and Alice is *equally unsure* about which animal Bob could be hinting at (formally, Alice has a *uniform prior*). Now, if Bob was to tell Alice: 'the animal is a mammal', which has 1/2 a chance of being the case, this would amount to giving Alice $-log_2(1/2) = 1$ bit of information. Therefore, *assuming a uniform prior*, one bit of information allows Alice to cut down her hypothesis space by a factor of 1/2 and n bits of information would allow Alice to cut down her hypothesis space by a factor of $(1/2)^n$.

2.2 Entropy, relative entropy and mutual information

Entropy: Equipped with our definition of information, we can proceed to introduce the concept of *entropy*. We'll introduce it here as a means of quantifying the *uncertainty* of an ensemble, but it does have other interpretations, such as in coding theory (where it is related to the fewest number of bits required to encode symbols in the ensemble alphabet).

Although two ensembles might have the same alphabets, it is clear that some ensembles are more predictable than others. Consider the fair coin which lands on heads 50% of the time and tails 50% of the time and the biased coin which has a head-to-tails ratio of 9 : 1; we'll call these hypotheses $\mathcal{H}_1$

and $\mathcal{H}_2$ respectively. The first 10 outcomes of $\mathcal{H}_1$ and $\mathcal{H}_2$ may look like:
$\mathcal{H}_1$ = HTTHHHTHTH
$\mathcal{H}_2$ = HHHHHTHHHT
One is easily able to see that $\mathcal{H}_1$ is the more unpredictable of the hypotheses.

As before, we now look to formalise this notion of ensemble unpredictability (or uncertainty), which we'll call entropy. This was done by A.Khinchin in 1957 ([8]), who searched for a mathematical function that would take an ensemble X and return a scalar value that quantified ensemble uncertainty. The function in question was required to satisfy the following four desiderata ([3]):

1. **Uniform distributions have maximum uncertainty**: In the animal guessing game above, we know that if Alice assigns 99% credence to the animal being a mammal, and 1% to the animal being a reptile, she is highly *certain* about her beliefs. Conversely, a 50% credence on either possibility represents maximum *uncertainty* about the class of the animal.

2. **Additivity of uncertainty for independent events**: If one has h_1 uncertainty in whether or not it will rain tomorrow, and h_2 uncertainty in whether or not they will pass their midterm exams, assuming that the events are independent, their total uncertainty should be $h_1 + h_2$. This is because reducing uncertainty in one of these should have no effect on the uncertainty in the other.

3. **Impossible events should not increase or decrease uncertainty** - Consider two people playing the animal guessing game; Alice, who has uncertainty about whether the animal is a mammal or reptile, and Charlie, who has uncertainty about whether the animal is a mammal, reptile, or an organism from the planet Mercury. If Charlie assigns zero credence to the belief that the organism is from Mercury (and assigns identical credences to Alice for the mammal and reptile possibilities) then Charlie's has the same uncertainty as Alice does

4. **Continuity of uncertainty H w.r.t arguments (input probabilities)** - We expect that small changes in the input probabilities $P(X)$, should not lead to any 'sudden jumps' in the uncertainty $H(X)$. In other words, for any change in uncertainty $\Delta H(X)$ (no matter how small), there should be some change that can be made to the probability distribution $\Delta P(X)$ that leads to this change.

Khinchin found that the only family of functions satisfying the above were the $H(X)$ such that:

$$H(X) = \lambda \underset{x \in A_X}{\mathbb{E}}[I(x)] = -\lambda \sum_{x \in A_X} p(x) log_2(p(x)$$

where $I(x)$ is the information content of event x (note that the entropy of an ensemble can also be viewed as its expected information). This entropy is technically known as the Shannon entropy due to the base 2 used in the logarithm of $I(x)$. We see that this formalism matches our intuition that $\mathcal{H}_1$ is more unpredictable than $\mathcal{H}_2$ - respective entropies (uncertainties) are 1 bit and 0.469 bits.

Relative entropy / KL-divergence: The next addition to our information-theoretic toolbox will be the *relative entropy* or *Kullback-Leibler (KL) divergence*, defined with respect to two ensembles, X and Y, which must share the same alphabet ($A_X = A_Y$). The KL-divergence serves as a quantification of the difference between two probability distributions and is given as:

$$D_{KL}(X,Y) = \sum_{a \in A_X = A_Y} p(X=a) log(\frac{P(X=a)}{P(Y=a)})$$

.

Observe that this metric is *not symmetric*, i.e $D_{KL}(X,Y) \neq D_{KL}(Y,X)$. Like entropy, the KL divergence has a nicer interpretation in coding theory; see [2] for a detailed explanation of this.

Mutual information We finish with a discussion of the *mutual information* shared between two random variables. To motivate this concept, consider the following example:

Example: Let X be the ensemble representing the average *inflation rate* in the United Kingdom during the month of June, in a particular year. Let Y be the ensemble representing the national interest rate in the same month. For concreteness, say $X = (x, A_X = \{2.0, 3.5, 4.5, 6.0\}, P_X = \{0.5, 0.2, 0.2, 0.1\})$ and $Y = (y, A_Y = \{0.25, 0.50, 2.00, 5.00\}, P_Y = \{0.3, 0.4, 0.2, 0.1\})$.
Our question is then as follows. How much can one reduce their uncertainty in Y, the interest rate,

given that they are told the value of X, the inflation rate? Intuitively, it seems that such a concept should exist if X and Y are correlated. Whilst our distribution over possible interest rates might initially be P_Y, if one were to find out that inflation was relatively high, say 6.0%, then they would be sensible to revise P_Y, transferring probability mass to higher interest rates, as it is likely that the central bank would be employing some form of monetary policy.

The mutual information, $I(X;Y)$ allows us to precisely answer the question above. Let's give its definition first, and then discuss. The mutual information $I(X;Y)$ between two ensembles X, Y is:

$$I(X;Y) = H(Y) - \underset{x \in A_X}{\mathbb{E}}[H(Y|X = x)]$$

Let's understand this by first considering the second term - this is the expectation over $x \in A_X$ of the *conditional entropy* $H(Y|X = x)$. The conditional entropy is a quantity that is new to us, but it follows naturally from the 'regular' notion of entropy. Consider our probability distribution over interest rates (red bars, Figure 2). If we were then to learn that the inflation rate was x, we may revise this distribution (blue bars, Figure 2. Comparing both distributions, we see that our uncertainty in interest rates has reduced - we can quantify this new uncertainty with the *conditional entropy.*

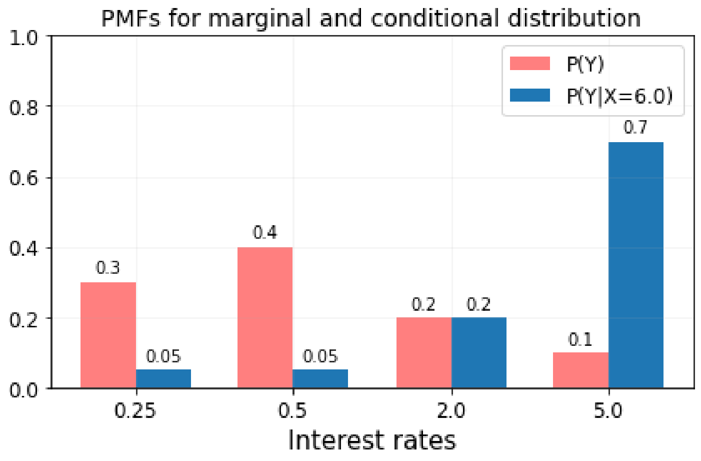


Figure 2: Probability mass functions for the marginal and conditional distribution. Computing the entropy for both, we find $H(Y) = 1.846$ bits, $H(Y|X = 6.0) = 1.257$ bits - thus, our uncertainty in interest rates have reduced after finding out the inflation rate.

To further develop intuition, let's consider the extreme values of $I(X;Y)$. If $I(X;Y) = 0$, we conclude that $\underset{x \in A_X}{\mathbb{E}}[H(Y|X = x)] = H(Y)$ - i.e, our average uncertainty in Y (interest rate) upon finding out $X = x$ (inflation rate) is the *exact same* as our regular uncertainty in Y; in other words, $X = x$ has given us no additional information about X. Conversely, consider the case in which $I(X;Y)$ attains its maximum value, $H(Y)$, i.e $\underset{x \in A_X}{\mathbb{E}}[H(Y|X = x)] = 0$. This informs us that, upon finding the value of X, our uncertainty in Y reduces to *zero*. From this, we can conclude that there is a deterministic relationship between x and y; one is directly fixed by the other.

This brings to an end the overview of some key concepts in information theory. In the next section we'll look at Representation Learning in modern day machine learning (ML) - methods by which ML systems can best compress raw input data.

3 Representation learning

The following section is based on [5].

Consider trying to describe the location of N points in three dimensional space. Three common ways to do so are to give the Cartesian representation of the locations, (x, y, z), the spherical coordinate representation (r, ϕ, θ) or the cylindrical coordinate representation, (r, ϕ, z)[2].

As another example, how might one describe a data point containing the current state of one mole of an ideal gas? One way would be to give the position and velocity of *each molecule* that composes the gas - given that each molecule can be fully represented with six numbers[3], the ideal gas could be fully described with a $6 \times N_A$ dimensional vector, where N_A is Avogadro's number, or 6.02×10^{23}. However, we know of another way in which the gas could be represented, involving just three numbers: *pressure, volume, temperature* (p, V, T). As a result, if we were given a dataset containing data for N one mole ideal gases, we could choose to plot this dataset as N points in three dimensional space, or N points in $6N_A$ dimensional space. One is clearly more practical than the other.

As a final example, consider a dataset of greyscale images of human faces, with pixel dimensions of 100×100. Each pixel can take a value from 0 to 256. Now, if asked to give a description of a single image, one *could* give a $10,000$ pixel values; however, one might succinctly summarise the image with reference to a few abstract qualities, such as: skin colour, eye colour, hair colour, size of nose, distance between eyes, shape of head etc. These qualities could easily be encoded numerically[4], allowing the dataset to be represented geometrically as N points in a relatively low dimensional space, rather than N points in $100,000$ dimensional space.

From the examples above, one sees that a *single dataset can admit multiple representations*, with some being much simpler than others. A low dimensional representation makes dataset structure more apparent - in the second example, one is likely to find it difficult to visualise a dataset projected in $6 \times N_A$ dimensions, but upon viewing the same dataset in its (p, V, T) representation, it quickly becomes apparent that our data points have some underlying structure (see Figure 3). Finding different representations, determining which are 'better' than others in a particular use case (Section 3.1), and discerning underlying dataset structure are just a few of the questions pursued by the field of representation learning.

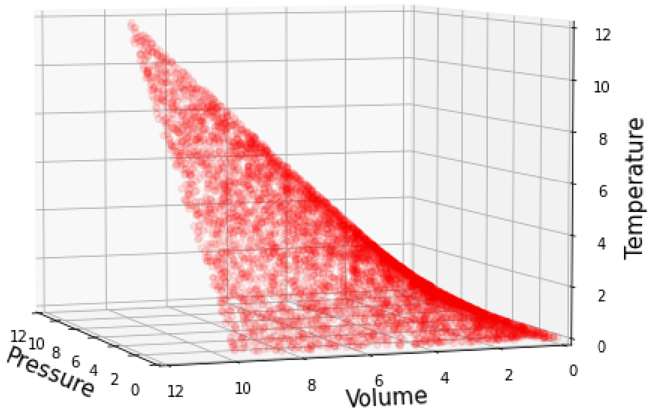


Figure 3: A sample plot of ideal gases in their (p, V, T) coordinates. It is clear that all points are embedded in some two dimensional structure, or 'manifold'.

[2]Please see [4] if unfamiliar with these coordinate systems.

[3]Three numbers to describe position, (x, yz) and three numbers to describe the velocity, (v_x, v_y, v_z).

[4]For example, the skin colours white, black and brown could be encoded as 0,1 and 2, respectively.

3.1 What makes a good representation?

In this subsection, we characterise representations and discuss how one can think about the quality of representations.

3.1.1 Low dimensionality

Where possible, we want our representation of the data to have as few dimensions as possible, whilst capturing most of the variance in the distribution. The key motivation behind this is to avoid the *curse of dimensionality.*
A common task that a system might need to perform is *classification* - as a simple example, imagine a neural network trying to classify images of cats and dogs, where these images are represented in $\mathbb{R}^{100,000}$ as in the example above. Assuming that the images of cats and dogs form distinct clusters in the so-called 'input space', classification is usually done by finding some subspace that clearly separates the two clusters, after which one just needs to determine which side a data point lies on with respect to the subspace in order to perform classification. Figure 4 shows two clusters of data points separated by a hyperplane in $\mathbb{R}^3$; if we imagine that the red points are images of dogs, and the blue points are images of cats, then a dog-cat classifier need only determine which side of the hyperplane a data point lies on.
Note - By no means must it be a hyperplane that separates clusters of data points; present day machine learning algorithms also use non-linear structures, referred to as manifolds. From hereon we'll use the term manifolds, of which hyperplanes are just a subset of.

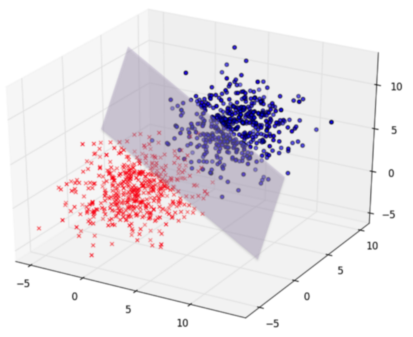


Figure 4: A manifold (in this case, a hyperplane) separating two clusters of data in input space. Manifolds in lower dimensions are generally less complex than those in high dimensions, making them easier to learn.

High-dimensional manifolds are generally more complex than low dimensional manifolds, in the sense that they are more 'wrinkled' (have a greater number of local optima per unit volume). As a result, underlying structure in a dataset that is represented in a high-dimensional space is much harder to learn via interpolation and extrapolation. This is the *curse of dimensionality.*

Therefore we want to represent our dataset in a low dimension, with the hope that the manifolds separating clusters of data points will be relatively smooth (and thus easier to learn), relative to the manifolds learned from the dataset represented in a higher dimension. This should allow for better performance in generalising to unseen data.

3.1.2 Distributed representations

Creating distributed representations is about ensuring the efficient storage of knowledge, and leveraging underlying structure in a dataset. To see this, let's look at two ways in which the following group of eleven words, {bat, dog, cow, tractor, car, cruiseliner, bacterium, monkey, lion, yacht, fish}, can be assigned a machine-interpretable representation

Localist representations - Localist representations represent each concept with a single information processing unit, or neuron, in the case of neural networks. That is to say that eleven neurons are needed to fully represent the group of words above, with neuron 1 firing to represent a bat, neuron 2 firing to represent a dog, and so on. This encoding scheme can be written numerically using an eleven bit vector (One Hot Encoding):

bat	[10000000000]
dog	[01000000000]
cow	[00100000000]
tractor	[00010000000]
car	[00001000000]
cruiseliner	[00000100000]
bacterium	[00000010000]
monkey	[00000001000]
lion	[00000000100]
yacht	[00000000010]
fish	[00000000001]

Table 1: A One Hot Encoding (OHE) scheme for the eleven words above

Whilst localist representations are intuitive and easy to implement, there are a number of drawbacks that lead to them being impractical in many ML tasks. For example:

1. For very large groups of words, the encoding vector is computationally expensive. WikiText-103, a popular natural language dataset, has a vocabulary size of $267,735$, meaning a vector with $267,735$ elements is required to represent each word using a OHE scheme. More generally, to distinguish between N different things, a localist representation requires N bits, which is rather inefficient, as we shall see.

2. Localist representations fail to incorporate dataset structure. The words 'bat' and 'dog' clearly share a relationship that 'bat' and 'cruiseliner' do not - namely, bats and dogs are both types of animals. However, the encodings for 'bat' and 'dog' are no more similar than the encodings for 'bat' and cruiseliner in the OHE scheme above.

Distributed representations - Distributed representations seek to overcome these problems by making the assumption that all concepts to be represented share some 'underlying explanatory factors' [5]. Consider the eleven words above - we could think of scoring each word by it's 'aliveness' (the degree to which the word represents a living organism). Cruiseliners are not living things, so this word receives a score of 0. Biologists consider bacteria to be living organisms, though they clearly do not exhibit the complexity that mammals do, so we assign it a score of 1. The animals in our group of words all receive an 'aliveness' score of 2. A similar thing can be done for scoring the words along a 'size' scale, where $0, 1$ and 2 represent a small, medium-sized, and large thing respectively, as well as for more abstract qualities, such as the degree to which a concept is related to the ocean, which, for the lack of a better name, we'll call 'marine-ness'. Fish and cruiseliner score 2 here, and things such as monkeys and dogs score 0.

Therefore, by acknowledging that the concepts to be represented *share some underlying explanatory factors*, a new representation scheme emerges, in which each concept is scored by its 'aliveness', size and 'marine-ness':

bat	[210]
dog	[210]
cow	[210]
tractor	[020]
car	[020]
cruiseliner	[022]
bacterium	[101]
monkey	[210]
lion	[210]
yacht	[021]
fish	[212]

Table 2: A distributed representation scheme for the group of eleven words.

One may immediately notice a problem here - many concepts have the same representation in this distributed scheme (see monkey and dog). This however, requires a very simple fix - simply increase the granularity of the scoring or add more explanatory factors. We could introduce more size levels that differentiate between monkey or dog (increasing scoring granularity); alternatively, one could score these concepts on 'domesticity' - the degree to which the thing is part of home/family life (a dog will score higher than a monkey here). In fact, modern day distributed representation schemes in NLP (known as word embeddings) project each concept into vector spaces that have hundreds or thousands of dimensions, rather than the three we have used in this example. Words are also assigned a score *from the real line,* $\mathbb{R}$, rather than a $0, 1$ or 2 as in the example above.

The representation scheme in Table 2 is more efficient in storing concepts - if each explanatory factor can receive a score from 0 to $k-1$, a vector of size n can represent k^n concepts (k=3 in the example above); compare this to the localist representation, which could only represent n concepts. Furthermore, dataset structure is also incorporated - notice the similarity between the 'bat' and 'dog' representations, compared to the 'bat' and 'cruiseliner' representations.

3.1.3 Hierarchy of abstraction and Depth

Objects in the natural world exist at various scales of complexity, and we want our representation format to account for this. To present this idea, let us consider how the retina, thalamus and visual cortex in the human brain work together to identify objects. Figure 5 shows a simplified schematic of the key components of visual processing.

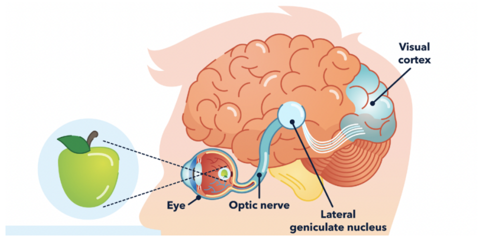
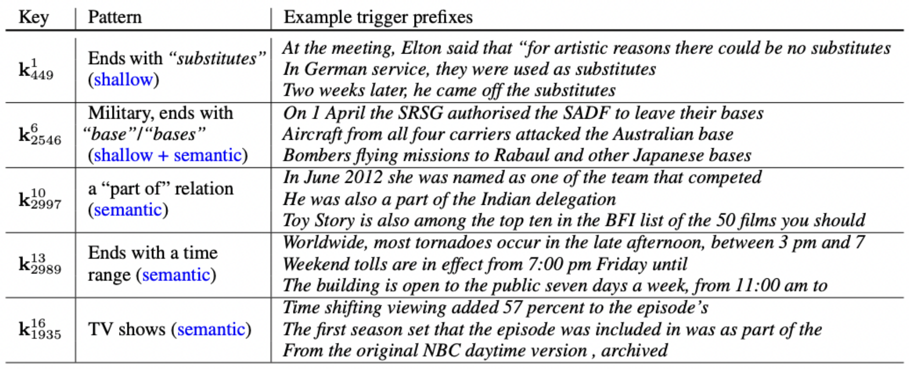


Figure 5: A simple schematic of the visual processing system. The lateral geniculate nucleas (LGN) is a cluster of neurons located in the thalamus. As discussed below, the visual cortex can be split into six subcortices.

At a very high level, regions of the brain that are involved earlier in the visual processing loop (retina, LGN) are responsible for identifying simple patterns in the input, acting as curve, edge and colour detectors, which can be mathematically modelled by applying convolution masks to the high dimensional raw data. Intermediate stages of the visual system, such as the first and second of six sub-cortices of the visual cortex, known as V1 and V2, take the results of this processing, and identify slightly more abstract objects such as simple 2D shapes, in a process that *may* mirror the convolutional mask method, but is still not well understood. The higher cortices (V3,V4,V5 and V6) then execute an analogous process, taking lower complexity objects identified by previous stages of the processing loop, and finding more sophisticated patterns in the input. For example, V3 and V4 may identify simple 3D shapes and more complex 3D shapes respectively, with neurons in V5 and V6 firing in response to highly abstract concepts that are familiar to humans in everyday experience, such as the concept of a 'tree', or 'beach resort'. Studies of artificial neural networks corroborate the idea that neural systems engage in this serial processing to build up more complex concepts from simpler patterns. Results from [10] show that lower layers in a number of convolutions neural networks can detect line orientation and act as Gabor filters[5], whereas higher layers engage in more complex tasks, such as identifying dogs or cats. It is interesting to see the 'simple patterns in lower layers, more complex patterns in higher layers' principle in the field of natural language processing too. It has been shown that the feed-forward layers in lower levels of transformer models identify syntactic patterns in input text, whereas higher level feed-forward layers respond strongest to semantic patterns in the input text [7] - see Figure 6.

The multilayer architecture of deep neural networks is well suited to this notion of more abstract concepts being built from less complex ones and may be the driving force behind the success of deep learning in the past decade. The field of deep transfer learning seeks to leverage this property by 'freezing' trainable parameters in lower level layers of the network (which are responsible for identification of simple, more universal features), whilst continuously retraining the weights in higher levels to fit a neural network to specific tasks. This method has shown to be effective in recent years by the pretraining-finetuning paradigm used to develop transformer-based language models.



Key	Pattern	Example trigger prefixes
$\mathbf{k}^{1}_{449}$	Ends with *"substitutes"* (shallow)	*At the meeting, Elton said that "for artistic reasons there could be no substitutes* *In German service, they were used as substitutes* *Two weeks later, he came off the substitutes*
$\mathbf{k}^{6}_{2546}$	Military, ends with *"base"/"bases"* (shallow + semantic)	*On 1 April the SRSG authorised the SADF to leave their bases* *Aircraft from all four carriers attacked the Australian base* *Bombers flying missions to Rabaul and other Japanese bases*
$\mathbf{k}^{10}_{2997}$	a "part of" relation (semantic)	*In June 2012 she was named as one of the team that competed* *He was also a part of the Indian delegation* *Toy Story is also among the top ten in the BFI list of the 50 films you should*
$\mathbf{k}^{13}_{2989}$	Ends with a time range (semantic)	*Worldwide, most tornadoes occur in the late afternoon, between 3 pm and 7* *Weekend tolls are in effect from 7:00 pm Friday until* *The building is open to the public seven days a week, from 11:00 am to*
$\mathbf{k}^{16}_{1935}$	TV shows (semantic)	*Time shifting viewing added 57 percent to the episode's* *The first season set that the episode was included in was as part of the* *From the original NBC daytime version , archived*

Figure 6: Results from [7] showing that higher level feedforward layers capture semantic patterns whereas lower levels capture syntactic patterns. One can interpret k^i_j as the jth neuron in layer i - the table then shows the general pattern that the neuron identifies (in the second column), and examples of textual input that led to a high activation the neuron (third column).

3.2 Three paradigms

One can view the problem of learning representations through three lenses: probabilistic methods, auto-encoder methods, and learning manifolds. Inspired by [5], we'll look at principal component analysis (PCA) through each of these lenses. In the following discussions, $\mathbf{x}, \mathbf{h}$ will denote the raw input data and the learned representation (feature vector) respectively, with the elements of $\mathbf{h}$ referred to as *latent variables*.

3.2.1 Probabilistic methods

Probabilistic methods incorporate uncertainty over latent variables by treating the learned representation as a random variable. Thus, given input data $\mathbf{x}$, the aim is to find a posterior distribution $p(h|x)$, from which a feature vector can be computed by either finding the expectation of $h|x$, or the modal value of $h|x$. Given that $p(h|x) \propto p(x,h) = p(x|h)p(h)$, the probabilistic method of representation learning can be broken down into two further sub-paradigms: directed graphical models which parameterise the *likelihood*, $p(x|h)$ and *prior* $p(h)$ separately, and undirected graphical models that just parameterise the joint distribution $p(x,h)$. Classic 'd-dimensional' PCA (PCA using the d principal

[5]Filters that identify a certain range of spatial frequncies in the image.

components of the raw data) makes a direct mapping between a data point in raw input space and its representation in the subspace spanned by the d principal components. Probabilistic PCA instead aims to produce an *posterior* distribution over this subspace, $p(h|x)$, by characterising the prior $p(h)$ and likelihood $p(x|h)$ as multivariate Gaussian distributions. The prior distribution is centred about the origin ($\mu_h = 0$), and the mean of $p(x|h)$ is an affine function of the latent variable that the distribution is conditioned on.
Once the posterior distribution has been computed ($p(h|x)$), one can then find a feature vector for $\mathbf{x}$ by computing the *expected feature vector* $\mathbb{E}[h|x] = \int_{-\infty}^{\infty} p(h|x)\, h\, dh$ or the modal feature vector (the h that maximises $p(h|x)$).

3.2.2 Auto-encoder methods

It is sometimes preferable to learn a non-probabilistic representation of some raw data $\mathbf{x}$, for which auto-encoder methods can be used. The underlying idea behind the auto-encoder is to learn *two* functions, an encoder $f_\theta()$ that yields a feature vector $\mathbf{h}$ given $\mathbf{x}$, and a decoder $g_\theta()$ which takes $\mathbf{h}$ and produces a reconstruction of the input, denoted $\widehat{\mathbf{x}}$. The aim of the auto-encoder is to minimise the difference between input $\mathbf{x}$ and $\widehat{\mathbf{x}}$, or the reconstruction error. If the encoder and decoder functions are linear transformations, the reconstruction error is the Mean Squared Error (MSE) and the dimension of the encoded representation h is d, the auto-encoder projects a data point $\mathbf{x}$ onto the first d principal components of the dataset (PCA).
It has been observed that auto-encoder solely trained solely to minimise reconstruction error might suffer from *collapse* - a phenomenon in which the identity function is learned, i.e the auto-encoder just outputs its input. To prevent this from occurring, and to enforce further properties on the encoding $\mathbf{h}$, one can add regularisation terms to the loss function. We discuss three:

Sparse auto-encoder: - Enforcing sparsity onto the representation may assist in *identifying underlying explanatory factors* (see section 3.1). A simple loss function here could be the $L1$ penalty (the sum of the absolute values of the latent variables, h_j).
Denoising auto-encoder: - In this case, artificial noise is first added to the input $\mathbf{x}$ to give a corrupted input $\tilde{\mathbf{x}}$ which is then fed into the auto-encoder - this forces the architecture to learn an encoding $\mathbf{h}$ that captures the underlying input distribution. Collapse is prevented as an auto-encoder that learns the identity map will also incorporate the artificial noise applied to the input, leading to large reconstruction error.
Contrastive auto-encoder: As mentioned in 3.1, a good low dimensional representation will identify abstract features in the raw input $\mathbf{x}$ (e.g: human faces are better described in terms of abstract features such as 'nose size' and 'face width' rather than a $100,000$ dimensional vector containing pixel intensities). Furthermore, these abstract features are highly *invariant* to small perturbations in the original raw input. Making this observation, we can add a loss term to the reconstruction error to force the auto-encoder to learn abstract representations that are robust to small perturbations in the input. Defining the Jacobian matrix as $J \in \mathbb{R}^{d_h \times d_x}$, $J_{ij} = \frac{dh_i}{dx_j}$, where d_h, d_x are dimensions of feature vector $\mathbf{h}$ and the raw input respectively, we can add the *Frobenius norm* (simply the square root of the sum of the squared elements in J) as a regularisation term[6].

3.2.3 Method of learning manifolds

The final perspective we will take on representation learning is a geometric one - that of learning a manifold[7] that a dataset approximates. We will make the assumption that, although the dataset is given in an 'ambient' dimension, say $\mathbb{R}^n$, it is well modelled by a d dimensional manifold, where $d < n$[8] - our task, then, is to find this manifold. Figure 7 provides intuition for the problem of manifold learning.

The relationship between learning a low dimensional manifold and representation learning is as follows: rather than describing a dataset in n dimensions, we could instead find a coordinate system

[6]Formally, the regularisation term is $sqrt(\sum_j^{d_x} \sum_i^{d_h} J_{ij})$

[7]A manifold is a very general concept, capturing the notion of a surface. This surface need not be globally 'flat'; curvature is permitted.

[8]Whether this assumption holds for real-word datasets is unknown; the proposition that this *is* the case is known as the *Manifold Hypothesis*

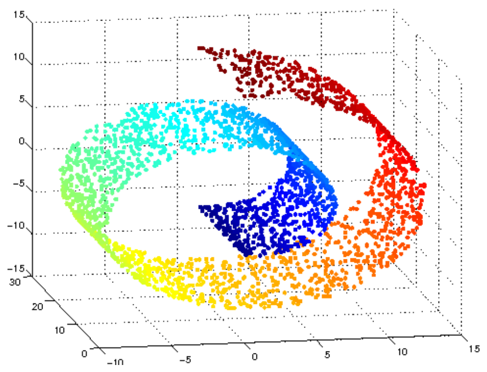


Figure 7: A dataset embedded in 3 dimensional space, that is well modelled by a 'rolled up' 2D plane. *Source: [6]*

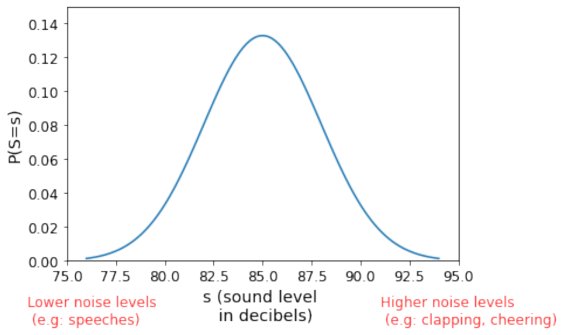
intrinsic to the low dimensional manifold, thus producing a d dimensional representation. Figure 7 is a simple example, but easy to visualise. A more concrete example is that of images - consider our image representation in $\mathbb{R}^100,000$ discussed earlier. Do all images containing dogs, say, lie near some four dimensional manifold? Do all images of houses in England lie on some 23 dimensional manifold? If so, what algorithms might we employ to find them?
The interpretation of PCA here is simple - expressing a data point $\mathbf{x}$ as a linear sum of its d principal components $\mathbf{p_i}$ ($\mathbf{x} = \sum_i^d \alpha_i \mathbf{p_i}$), is equivalent to finding the d dimensional *hyperplane* that best approximates the dataset, and projecting $\mathbf{x}$ into that hyperplane.
We end by introducing three popular manifold learning methods, as well as a high-level summary of the intuition behind them. These all follow the same underlying principle of finding a low dimensional embedding whilst preserving some key properties of the original dataset.
Multidimensional scaling (MDS) - This class of algorithms takes a dataset X, comprised of N points projected in n dimensions, and first computes an NN pairwise distance matrix M (M_{ij} is the Euclidean between $x_i, x_j \in \mathbb{X}$). The dataset is then projected into dimension d, and pairwise distances are recomputed - call this matrix M'. The optimisation objective is thus the norm of the pairwise difference between M and M', ensuring that distances in M are preserved.
Local Linear Embeddings (LLE) - This is a small modification on MDS - if we expect the low dimensional manifold to be non-linear, it is impractical to demand that all the pairwise distances between *all* points be preserved; LLE aims to preserve the distance between x_i and it's k nearest neighbours, for all i.
Isomap - This method can be viewed as an extension of MDS. Like MDS, it aims to preserve a distance between points in the dataset, but rather than preserve the Euclidean distance, Isomap preserves a quantity known as the *geodesic distance.* The geodesic distance between two points x_i, x_j, denoted as $g_{\mathcal{M}}(x_i, x_j)$, is the shortest distance between x_i and x_j when one is constrained to move on manifold $\mathcal{M}$. For example, let x_i, x_j be the points $(1, 0), (0, 1)$ in $\mathbb{R}^2$ respectively, and let $\mathcal{M}$ be the unit circle (a one dimensional manifold). Whilst the Euclidean distance between these two point is $\sqrt{2}$, the geodesic distance $g_{\mathcal{M}}(x_i, x_j) = 2\pi/2 = \pi$. Of course, when executing the Isomap algorithm, the manifold $\mathcal{M}$ that the dataset lies on is unknown - this is what we are interested in finding! Instead, the geodesic distance is approximated by finding the distance from x_i to x_j on a k-nearest neighbours graph [9], with the data points of the dataset forming the graph vertices.

4 Some Computational Neuroscience

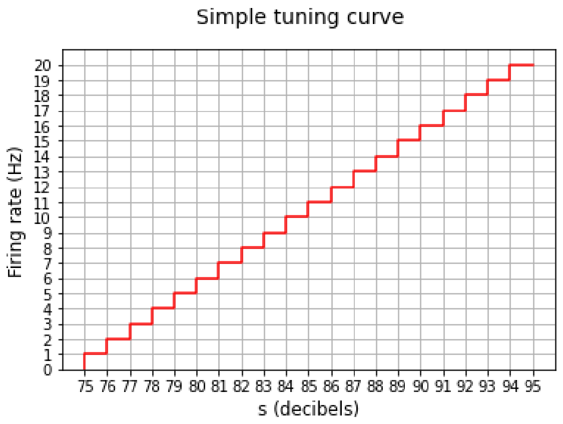
The following section is heavily based on [1]. Please see the original website for a more comprehensive summary of the ideas discussed here.

In this section, we'll discuss *neural encoding* - the process by which biological brains map their external environment to a complex series of electrical impulses, also known as neural spike trains. In particular, after having introduced some mathematical formalism, we'll derive the sigmoid activation

[9] It seems that the choice of k will be dependent on the manifold dimension d that the data is being fitted to.



(a) An example of the stimulus parameter distribution.



(b) A simple mapping from stimulus parameter to firing rate (a tuning curve). This is how neurons encode external stimuli as series of electrical impulses. As we'll see in the next section, few (if any) tuning curves take this linear shape.

Figure 8

function (seen widely in both biological and artificial neural networks) from an information-theoretic perspective.

4.1 Basic formalism

There are two important quantities in neural encoding: the stimulus parameter S and the neuron firing rate R. The stimulus parameter S characterises an external stimulus; for example, it could represent the light intensity (measured in candelas), or the sound level (decibels) in a room. Given that S will vary over space and time, it is suitable to represent it with an ensemble, $S = \{s, A_S, P_S\}$, where A_S is the range of values that S can take, and P_S the corresponding probability distribution over this space[10]. For example if we let S represent the sound levels in a room during an evening dinner, P_S may take the form of a normal distribution; one would expect some average noise level in the room, but regular deviations from this level, such as moments of clapping and cheering (heightened noise levels) and quieter periods, such as when a single member of the crowd gives a speech. This is shown in Figure 8a. 85.0 decibels is a reasonable mean noise level for this example, so we set A_S as the continuous interval $[75.0, 95.0]$, and the standard deviation to 3 decibels.[11]

The firing rate R characterises the response of a neuron to a given stimulus; we'll view it as the number of times that a neuron fires in a small time interval, in response to a stimulus S taking value s. Given that R is a function of the S, we must also represent it with an ensemble (r, A_R, P_R). Neurons can generally fire anywhere in between one and some r_{max} number of times[12], giving our neuron a total of r_{max} 'symbols' with which to encode information about the stimulus. For example, consider a neuron with the role of conveying the sound level of the dinner party above - it may fire once if $S = 75.0$ decibels, ten times if $S = 85.0$ decibels and twenty times if $S = 95.0$ decibels, with the other firing frequencies assigned to sound levels in between (thus, in this case $r_{max} = 20$). This mapping between stimulus parameter and firing rate is known as the tuning curve/neuron encoding, and is shown for this instance, in Figure 8b.

[10]We will use the notation P_S and $P(S)$ interchangeably, as will also be the case for P_R and $P(R)$ introduced in the next paragraph.

[11]Recall that the decibel scale is logarithmic, so 75.0 decibels is ten times quieter than 85.0 decibels.

[12]A neuron can also not fire, which itself can convey information about the stimulus, but we ignore this case for now to make the numbers in the discussion easier to work with.

4.2 Sigmoids (usually) Maximise Code Entropy

Let's imagine designing the human brain from scratch. Sticking with the dinner party example, how does one design a neural code R to optimally represent the environment, characterised by stimulus parameter S? In other words, what tuning curve do we choose to map the stimulus parameter onto firing rates?

At first, it may seem sensible to divide the stimulus range A_S up into r_{max} bins of equal size, and assign a firing frequency to each of these bins, which is shown in Figure 8b. In this case, the probability distribution over firing rates, P_R will look like a discretised version of 8a (with the space of possible values being A_R rather than A_S). The reason for this should be fairly evident - if $s = 85.0$ decibels is the most commonly recorded sound level, then one should expect to observe its associated firing rate (11 Hz), most often, with the probability distribution over firing rates falling away symmetrically on either side.

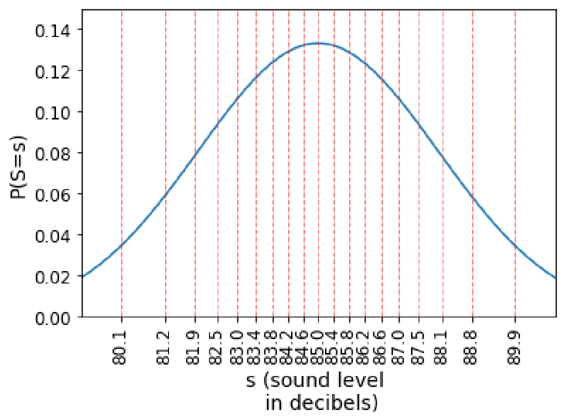
However, what properties do we want of our neural code? Recall that in section 2, we introduced the notion of entropy, which could be interpreted as the expected information when sampling from an ensemble. Now, when designing a neural code to transmit information, we'd like the code **to transmit the maximum (expected) information**, or in other words, **maximise the entropy of** $P(R)$. Under this guiding principle however, we see that the coding scheme presented in Figure 8b is not optimal; instead, to maximise code entropy, we must seek a code that produces a *uniform* $P(R)$.

Demanding that the neural code produces a uniform $P(R)$ dictates how the tuning curve should be constructed. In order to see 20 symbols (firing frequencies) equally often, one should assign the first symbol to the range of S that account for the first 5% of area underneath $P(S)$, the second symbol to the s values that account for the next 5%, and so on. For $P(S)$ above, this is shown in Figure 9a, where sound levels less than 80.1 decibels are assigned to a firing frequency of 1 Hz, the range $80.1 - 81.2$ decibels is assigned to 2 Hz, all the way up to sound levels exceeding 88.9 decibels, which are mapped to a firing frequency of 20 Hz. The resulting tuning curve is presented in Figure 9b.

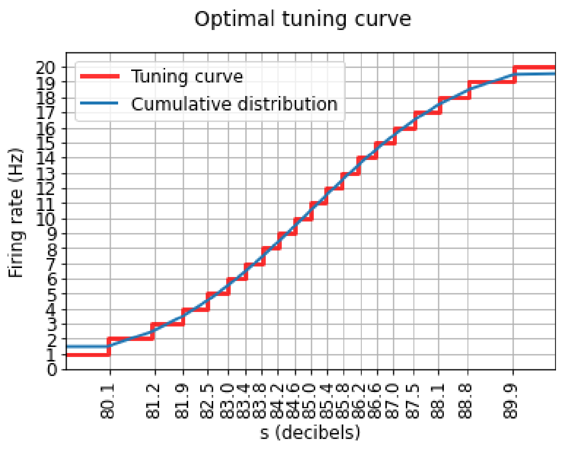
Thus, **the following general principle holds when designing tuning curves**: In order to design a maximally informative code (maximum entropy), the tuning curve should take the shape of the *cumulative distribution* of $P(S)$. Mathematically:

$$r(s) = r_{max} \int_{s_{min}}^{s} P(S)ds$$

For a stimulus S following a normal distribution, the resulting tuning curve is the 'S-curve', or sigmoid function. Given that many naturally occurring stimuli distributions are well modelled by normal distributions, the 'maximise code entropy' principle serves as a hypothesis for why the sigmoid function has been observed in biological neural networks, and are suitable activation functions in artificial neural networks. However, the general principle stated above also applies to stimuli that are modelled by other distributions, as discussed in [9].



(a)



(b)

References

[1] Computational neuroscience. https://www.coursera.org/learn/computational-neuroscience. Accessed: 2022-08-19.

[2] Count bayesian: Kullback-leibler divergence explained. https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained. Accessed: 2022-08-25.

[3] Entropy is a measure of uncertainty. https://towardsdatascience.com/entropy-is-a-measure-of-uncertainty-e2c000301c2c. Accessed: 2022-09-21.

[4] Spherical and cylindrical coordinates. https://math.libretexts.org/Bookshelves/Calculus/Book%3A_Calculus_(OpenStax)/12%3A_Vectors_in_Space/12.7%3A_Cylindrical_and_Spherical_Coordinates. Accessed: 2022-08-19.

[5] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives, 2012.

[6] L. Cayton. Algorithms for manifold learning. 2005.

[7] M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories, 2020.

[8] A. Khinchin. Mathematical foundation of information theory, 1957.

[9] S. Laughlin. A simple coding procedure enhances aneuron's information capacity. *Z. Naturforsch.*, 1981. https://www.degruyter.com/document/doi/10.1515/znc-1981-9-1040/html?lang=en.

[10] C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. An overview of early vision in inceptionv1. *Distill*, 2020. https://distill.pub/2020/circuits/early-vision.