THE CURSE OF DIMENSIONALITY

A DRAMA IN FIVE ACTS

MICHELLE HUNG AND ANDREW GRITSEVSKIY

Notes by us!!![†]

W375, Brown

In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move. (Douglas Adams, The Hitchhiker's Guide to the Galaxy*)*

---

1 AN INTRODUCTION

When you were born, a lot of things quickly became axioms. You always accelerate downward at $1g$, the sun shines from above, the pressure is 1atm. We've gotten so used to this that it's terrifying to go on a roller coaster; we shine flashlights onto the bottom of our faces while telling horror stories; our ears have to do significant adjustment to the 0.78 atm experienced on airplanes. However, one fundamental axiom restricts us forever—and no matter how hard we try, we have no change of escaping it (outside our powerful imagination). This principle of the universe is that we live in three spatial dimensions.

In many ways, this is a blessing. Imagine the complexity of kinematics calculations if projectiles flew through 10-dimensional space[1]. However, in a ten-dimensional world, we'd have a much lower chance of getting into the San Francisco housing crisis. Most importantly, three dimensions are easy to understand, and work quite well for everything we know. However, our intuition fails miserably as the number of dimensions increases, which jeopardizes our ability to solve mathematical problems, discover new cures to diseases, and begs the crucial question of *what goes wrong in higher dimensions*?

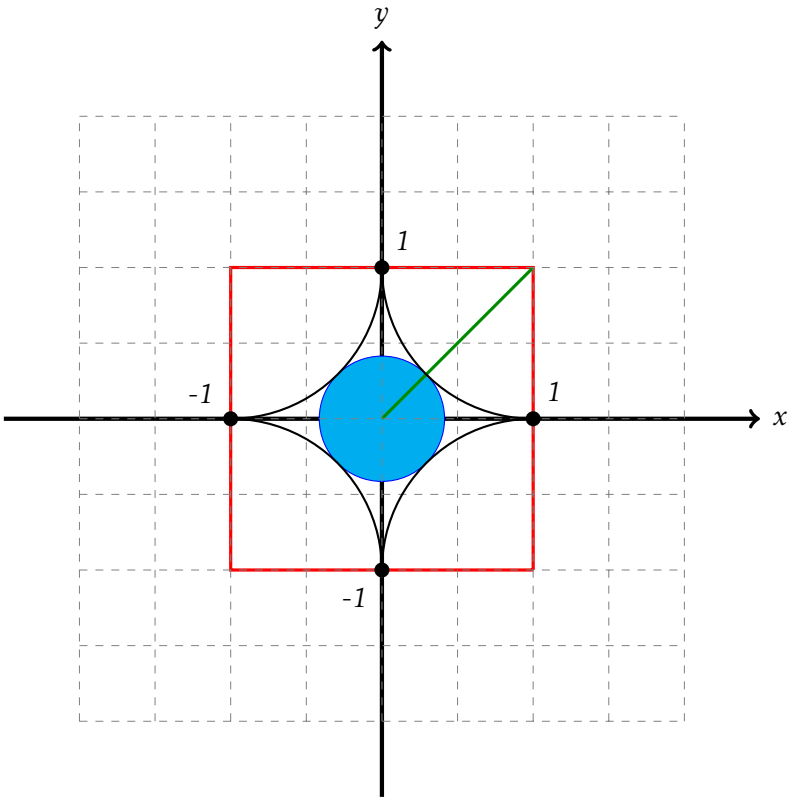
[†]Disclaimer: these notes have not been proofread and are not intended for publication.

[1]As physicist D. Kauffman pointed out, kinematics would not necessarily become a lot more difficult; however, in many other ways, life as we know it would either cease to exist or have to adapt significantly; for instance, due to the lack of solar system formations. For an interesting look at why 3 spatial + 1 time dimension may be necessary for us to exist, check out Max Tegmark's paper at http://space.mit.edu/home/tegmark/dimensions.pdf.

2 A FAILURE OF INTUITION

Consider the following simple construction in $\mathbb{R}^2$: we take a 2×2 square, centered at the origin. Place a quarter-circle in each of the corners, and finally, inscribe a circle centered at the origin tangent to all four quarter-circles, as shown:

Figure 1: The construction in $\mathbb{R}^2$



Let's find the ratio q_2 of the area of the inner circle to the area of the bounding box. Using some sick geometry, we get that the radius of the inner circle is $\sqrt{2} - 1$, and thus its area is $S_c = \pi(\sqrt{2} - 1)^2$. Therefore, we get

$$q_2 = \frac{\pi(\sqrt{2} - 1)^2}{2^2} \approx 0.135$$

Now consider the analog in $\mathbb{R}^3$. Instead of a square, take a $2 \times 2 \times 2$ cube, centered at the origin. In each corner, inscribe one-eighth of a sphere. Now, inscribe an inner sphere, centered at the origin, tangent to all eight pieces. What is q_3? Well, the radius of the inner sphere is $\sqrt{3} - 1$. Using the volume formula for a 2-sphere, we get

$$q_3 = \frac{\frac{4}{3}\pi(\sqrt{3} - 1)^3}{2^3} \approx 0.205$$

What does this ratio, q_d, approach, as the number of dimensions increases? That is, what is the value of

$$\lim_{d\to\infty} q_d \, ?$$

You may think, intuitively, that this ratio approaches 1. After all, it makes sense that the sphere gets larger and larger, and in infinite dimensions, begins to take up most of the space in the bounding box. Those of you who have had more exposure to higher-dimensional geometry may say that the ratio approaches zero[2]. Some starry-eyed mathematicians may hope that it approaches one-half. In fact,

$$\lim_{d\to\infty} q_d = \infty$$

If you're familiar with multivariable calculus, you can use d-dimensional spherical coordinates to derive the closed form

$$q_d = \frac{\pi^{\frac{d}{2}}(\sqrt{d}-1)^d}{\Gamma(\frac{d}{2}+1)2^d}$$

The gamma in the denominator is the Euler Gamma function, an extension of the factorial. To see where this comes from, try deriving this equation (Exercise) and look at the collected terms from the nested integrals.

and see the limit for yourself[3].

How does this make any sense? Well, the first place our intuition fails us is by the idea of the "bounding box". In fact, this box is not "bounding" at all—rather, it's a pretty arbitrary construction that the inner sphere can easily leave. In fact, consider the 4-dimensional case. The radius of the inner 3-sphere is $\sqrt{4}-1=1$. So, in four dimensions, our inner sphere starts pushing up against the boundary of the box!

In higher dimensions, the result is even more striking. Observe that

$$q_7 = \frac{\frac{16}{105}\pi^3(\sqrt{7}-1)^7}{2^7} \approx 1.21$$

So at this point, the volume of the inscribed sphere is straight up larger than that of the bounding box. Here's the question: where is all this extra area concentrated? Obviously, the most volume the ball can fit in the bounding box has an upper bound of 2^n. The extra area must all be concentrated in the small protrusions the ball makes out of the bounding box—in $\mathbb{R}^7$, this is a mere 17% of the volume, but as the dimensionality increases, closer and closer to 100% of the volume of the d-ball is concentrated on the surface. This is a fundamental tenet of the curse of dimensionality:

[2] After all, it is true that the volume of a constant-radius sphere in high dimensions approaches zero. In our case, however, the radius is very non-constant since it increases as $\sqrt{d}$.

[3] It's actually nontrivial to show that this goes to infinity as $d \to \infty$—there's a factorial and a 2^d term in the denominator! You can either do a bunch of fancy rearrangements and derivatives and Stirling approximations, or you can take WolframAlpha's word for it at goo.gl/GTWz1S.

In higher dimensions, almost all of the volume of a ball is concentrated arbitratily close to its surface.

Another way of seeing this is, given $V_d(r)$ is the volume of a d-ball with radius r, it is true that for any $\epsilon > 0$,

$$\lim_{d \to \infty} \frac{V_d(r)}{V_d(r - \epsilon)} = \infty$$

This leads to the following famous joke:

Never buy an orange in high dimensions. Once you peel it, there's nothing left!

3 THE K-NEAREST NEIGHBORS PROBLEM

Now we will explore an example of the curse of dimensionality at work.

3.1 *Approach 1: linear search*

The task is as follows: Given n points in $\mathbb{R}^d$ (where is the number of dimensions), find k nearest neighbors from query point $X \in \mathbb{R}^d$.

For a bit, we'll assume $k = 1$ to make things simple, and later we'll think about the general case.

The simplest approach to this problem is called linear search, and is conducted as follows. Store the current closest neighbor in variable "current best". For each point, measure its distance to X. If it is closer to X than "current best", update "current best" to be that new point. The running time of this algorithm is $O(n)$, because you need to iterate through n points.

What about if k can be anything from 1 to n? Then, you need to store a list of k current bests, and for each point, you need to compare it to each element of the list. The runtime increases to $O(kn)$, which is equivalent to $O(n^2)$, since k can go up to n. If the number of points is small, this is workable, but as n grows large linear search is much too slow. Can we do better?

Yes. Data structures called k-d trees allow us to significantly reduce the running time in low dimensions (we will see what happens in high dimensions later on).

3.2 *Approach 2: k-d trees*

A k-d tree, or k-dimensional tree, is a data structure used to organize n points in $\mathbb{R}^k$. A k-d tree for a set of n points in $\mathbb{R}^d$ is constructed as follows[4]:

– The root node of the tree corresponds to the median of the n points, taken with respect to the first dimension (rounding up if the median does not fall on one of the n points).

[4] I'm really sorry about lack of diagrams but in my defense, they're really annoying and difficult to make.

– Through that median point, a hyperplane split along the perpendicular axis divides the remaining points into left and right children of the root node and defines the *bounding box* containing itself and all its children.

– Each level down the tree, the dimension by which the median is measured cycles through the d dimensions. The algorithm iterates through all the points, creating bounding boxes until every point is on a hyperplane split.

Now consider a k-d tree representing n points in $\mathbb{R}^d$. We can use this tree to find the k nearest neighbors to a query point X $\in \mathbb{R}^d$. Again, we'll write out the algorithm for $k = 1$, which can easily be generalized to greater values of k.

– Store a running estimate of the nearest neighbor in the variable "current best". Starting with the root node, move down the tree recursively, in a depth-first fashion.

– At each node, if it is not closer to X than the current best, move on. Otherwise, do the following: store that node as the new current best. Then, draw a hypersphere with center at X and radius equal to the current best. For any bounding box that is completely disjoint from the hypersphere, remove from the tree the box's corresponding node and all of its children (because those points must be farther than the current best away from X, and thus cannot be X's closest neighbor).

– Continue until all remaining points (ones that have not been eliminated) have been checked.

What is the runtime of this new strategy? Building the k-d tree takes $O(n \log n)$ time (finding the median for each layer of the tree). Running the algorithm takes $O(\log n)$ on average and $O(n)$ at most (in the case that the k-d tree partitions cannot eliminate any bounding boxes). Since building the tree is a fixed cost and makes all future queries a lot easier, we can consider the runtime to be just $O(\log n)$.

What about when k goes up to n? The runtime becomes $O(n \log n)$. Recall that linear search took $O(n^2)$ time when k goes up to n. So, on average, this is must faster than linear search (if you get very unlucky, it is the same). But what happens if dimensions higher than 2?

Consider n data points chosen uniformly and at random inside a hypersphere sphere of radius 1 in $\mathbb{R}^d$. Note that in general, the volume of a hypersphere is $V = s_d r^d$, where s_d is some constant dependent on d, and r is the radius. Given a data set, we can calculate the distance of the center to its nearest neighbor. If we perform this experiment many times, what is the median of these distances? Let this median be m. Half of the time, the nearest neighbor will be farther than m and half of the time, it will be nearer than m. If it is farther than m, then all n data points are in the region between the

sphere of radius m and the sphere of radius 1. The chances of this occurring is 1/2. The volume between the spheres of radii m and 1 is $s_n 1^n - s_n d^n$. The probability that all N data points, generated independently, fall between both spheres is then

$$\frac{1}{2} = \left(\frac{s_d 1^d - s_d m^d}{s_d} \right)^n$$

So m is

$$m = \left(1 - \left(\frac{1}{2} \right)^{1/n} \right)^{1/d}$$

Thus, when d approaches infinity, m converges to 1. Geometrically, this means that all points tend to fall on the surface of the sphere (remember a similar result from the introductory hypersphere example).

In the context of nearest neighbor searches, when d becomes large, all points end up being a similar distance away from the query point, and it becomes hard to eliminate any bounding boxes. The runtime is just as bad as linear search.

In general, when the number of dimensions is high, Euclidean distance becomes a much worse quantifier of distance between two points.

4 CLUSTERING

Let's talk about some real-life situations where this causes some problems. Say you are looking for statistical patterns in a collection of data. One way of doing so is using a *clustering algorithm*—something that separates data into several disjoint sets, based on how "clustered" the points are. k-d trees, for instance, are often used to generate clusters by the metric of the distance between data points. However, the curse of dimensionality strikes again—normal cluster-finding strategies tend to fail in high dimensions.

There are, however, several ways to fix this problem. The first method, proposed by Robert Tibshirani, Guenther Walther, and Trevor Hastie at Stanford University in 2001, concerns what is known as *gap statistics*. Basically, there exist reference data sets with known distributions of points in several clusters. You can match the experimental data to the best reference distribution and find clusters accordingly.

Another proposal is something known as the "noise method": essentially, if you claim that you have some algorithm that clusters data, try moving around the points a bit. If the clusters remain the same, that's a good sign. If the clusters change significantly, then you're most likely clustering noise. This method allows you to check whether your clusters are reasonable; however, it does not tell us much about the underlying structure.

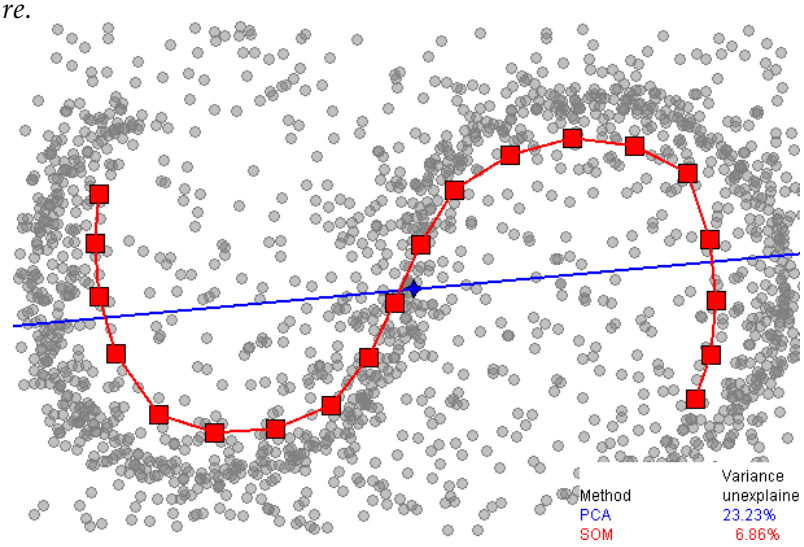
The last solution uses something called *manifold learning*, which we'll briefly define here:

4.1 *Manifold Learning*

1 DEFINITION. A **manifold** is a thing that looks like $\mathbb{R}^n$ if you zoom in far enough. More precisely, for every point on a manifold, there is a neighborhood of that point which is locally homeomorphic to n-dimensional Euclidean space.

The idea of manifold learning is simply that in most cases, the high dimensionality of data is artificial—that is, there is usually an embedding of a low-dimensional manifold in the data space that describes the data pretty well. This is shown in Figure 4.1, where there is a really simple S-shaped manifold that describes the data, which standard statistical techniques, such as principal component analysis, can't find.

Figure 2: A comparison of using principal component analysis (blue) to describe the main patterns in the data versus a type of manifold learning (red). Clearly, using the embedding of the 1-D manifold gives us a better description of the underlying structure.



5 AUTOENCODERS

Let's use this.

Autoencoders use manifold theory, along with recent technological developments, to cluster data. On a basic level, the structure of the autoencoder is simply two functions—f_1 and f_2. Given a certain input vector v, we apply f_1 to v to get some encoding e_v. Then, we apply f_2 to e_v to get back v.

$$f_2 \circ f_1 = \mathrm{id} \tag{1}$$

Now, let's say we're trying to learn something about the English language. Thus, for our input vectors, we take all the words of the English language. A lot of English words tend to be pretty long; for instance, the word "triangle" is eight letters long, which we consider to be a point in the eight-dimensional letter space. How do we reduce the dimensionality so that we can learn relationships between words?

Well, let's set some restrictions. We want the following things to be true:

1. $f_2 \circ f_1 = \text{id}$
2. e_v has length 4 (is a point in 4-dimensional space)
3. If v_1 and v_2 are similar words, then e_{v_1} and e_{v_2} are nearby points in the 4D space.

If an autoencoder follows these rules, we get a lot of pretty sweet properties. Essentially what is happening is we are taking the (high-dimensional[5]) input data and projecting it into a low dimensional space, which we call the *latent space*. Essentially, we are trying to fit all of our dataset onto this lower-dimensional manifold.

How do we actually go about *doing* this, rather than just saying this is what we want? Well, we do this using neural networks. For the purposes of this class, a neural network is simply a function that learns to minimize a certain loss in order to reach a certain objective. For instance, if we have a network that recognizes the faces of staff, the loss could be how many times the network gets a face wrong, such as labeling J-Lo's face as Jeff. We will not go into exactly how the neural network minimizes loss (something something multivariable calculus)[6].

Based on our specifications above, let's define some losses:

1. Since an autoencoder, by definition, has to lose some information, we want to make $f_2 \circ f_1$ as close as possible to the identity.
2. Let's say we also want to make related words really close along the x-dimension; that is, $\Delta_x(e_{v_1}, e_{v_2}) \propto \frac{1}{\text{relatedness}(v_1, v_2)}$
3. Similarly, let's say that $\Delta_y(e_{v_1}, e_{v_2}) \propto \frac{1}{\text{rhyming}(v_1, v_2)}$, where $\text{rhyming}(v_1, v_2)$ is how well two words rhyme[7].

[5] 8 is a big number (https://www.youtube.com/watch?v=0oDsibtPMHw)

[6] For a detailed introduction to the way neural networks learn, we'd recommend reading through the Stanford Convolutional Neural Networks for Visual Recognition class (cs231n.github.io), as well as checking out Michael Nielsen's neuralnetworksanddeeplearning.com. They're both excellent resources; in addition, feel free to talk to Dyusha. If you're into Julia-lang, check out https://github.com/derikk/nndl-julia for a sick (in-progress) implementation of Michael Nielsen's intro.

[7] So perfect rhymes are close together, slant rhymes are a bit farther, and non-rhymes are far apart.

That's probably it for now. You might have noticed that we're only imposing losses along the x and y-dimensions, which means that we're really projecting onto a two-dimensional space[8][9]. Therefore, the point e_{stool} will be close on the x-axis to the point e_{chair}, which itself will be close on the y-axis to the point e_{bear}. The point e_{mule}, for instance, might be pretty close on the x-axis to bear (since they're both animals) and a bit closer on the y-axis to stool (since it's a pretty decent rhyme).

What does this mean for our clustering algorithm? Well, recall how we had significant trouble in higher dimensions due to the curse of dimensionality. Now, since we projected all the data onto a two-dimensional manifold, we can use existing clustering algorithms to learn about our data!

This is the beauty of autoencoders—they produce a lower-dimensional representation of the data so we can learn on the manifold.

[8] In real life, we usually don't impose metrics along certain dimensions; we usually let the autoencoder figure it out for itself. However, for the sake of this example, let's say that this is the case.

[9] I just realized that this page already has a ton of footnotes. The place where this is most annoying is *War and Peace*—Tolstoy writes all his characters' five-page letters in *French*, so you have to sit with a microscope to read through the font size 6 translation of Natasha's letter at the bottom of the page. Anyways, we'd hate our readers to suffer a similar fate, which is why I'm gonna end this footnote right here.