L1 Norm Regularization and Sparsity Explained for Dummies

Shi Yan Follow
Aug 28, 2016 · 12 min read

Well, I think I'm just dumb. When understanding an abstract/mathematical idea, I have to really put it into images, I have to see and touch it in my head. I need the geometry, the object, the intuition behind the idea and better with vivid metaphors in real life.

Sometimes when I found people don't think or at least don't explain things this way, pointing me to equations and papers, saying there are no simple explanations, I got angry. And often after I thought stuff through, I could find silly intuitive explanations to those ideas. One such an experience was yesterday when I tried to understand L1 norm regularization applied to machine learning. Thus, I'd like to make this silly but intuitive piece to explain this idea to fellow dummies like myself.

When performing a machine learning task on a small dataset, one often suffers from the over-fitting problem, where the model accurately remembers all training data, including noise and unrelated features. Such a model often performs badly on new test or real data that have not been seen before. Because the model treats the training data too seriously, it failed to learn any meaningful pattern out of it, but simply memorizing everything it has seen.

Now, one solution to solve this issue is called regularization. The idea is applying an L1 norm to the solution vector of your machine learning problem (In case of deep learning, it's the neural network weights.), and trying to make it as small as possible. So if your initial goal is finding the best vector x to minimize a loss function f(x), your new task should incorporate the L1 norm of x into the formula, finding the minimum (f(x) + L1norm(x)). The big claim they often throw at you is this: An x with small L1 norm tends to be a sparse solution. Being sparse means that the majority of x's components (weights) are zeros, only few are non-zeros. And a sparse solution could avoid over-fitting.

That's it, that's how they explain it in most of the articles, textbooks, materials. Giving an idea without explanation feels like pushing a spear through the back of my head.

Not sure about you guys, but the reason for using an L1 norm to ensure sparsity and therefore avoid over-fitting wasn't so obvious to me. It took me some time to figure out why. Essentially, I had these questions:

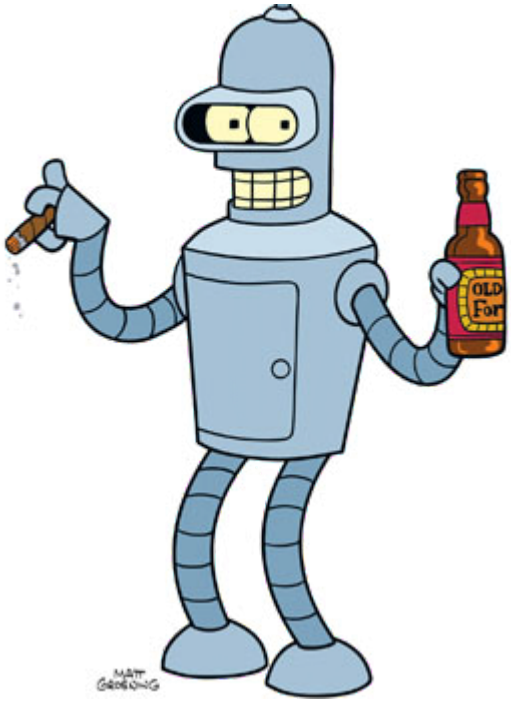
1. why does a small L1 norm give a sparse solution?
2. why does a sparse solution avoid over-fitting?
3. what does regularization do really?

My initial confusion came from the fact that I only looked at the L1 norm and only thought about what it means for L1 norm to be small. What I should really do, however, is thinking the loss function and the L1 norm penalty as a whole.

Let me explain it from the beginning, the over-fitting problem. I'd like to use a concrete example. Suppose you purchased a robot and you want to teach him to classify the Chinese characters by looking at the following example:

把打扒捕拉 被劈盐派喔

The first 5 characters belong to the first category, the last 5 are the second category. And these 10 characters are the only training data you have.



Now, unfortunately, the robot is too smart for the task. It has large enough memory to remember 5 characters. After seeing all the 10 characters, the robot learned a way to categorize them: It remembers all the first 5 characters exactly. As long as a character is not one of those 5, the robot will put the character into the second category.

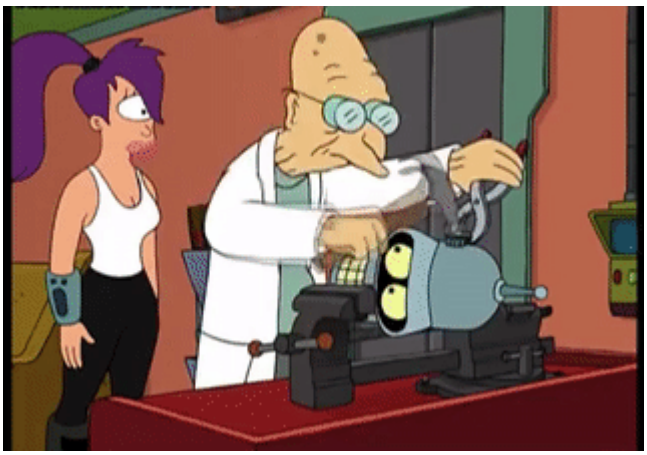
Of course, this method will work very well on the 10 training characters, as the robot can achieve 100% accuracy. However, you provide a new character:

This character should belong to the first category. But because it never appeared in the training data, the robot hasn't seen it before. Based on its algorithm, the robot will put this character into the second category, which is wrong.

It should be pretty obvious for us human to see the pattern here. All characters that belong to the first category have a common part. The robot failed the task because it's

too smart and the training data is too small.

This is the problem of over-fitting. But what is regularization and why can sparsity avoid over-fitting?



Now suppose you got angry at your robot. You banged the head of the robot with a hammer, and while doing it, you shook some of its memory chips off his head. You essentially have made the robot dumber. Now, instead of being able to memorize 5 characters, the robot can only remember a character part.

You let the robot do the training again by looking at all 10 characters and still force him to achieve the same accuracy. Because he can't remember all 5 characters this time, you essentially force him to look for a simpler pattern. Now he discovers the common part of all the category A characters!

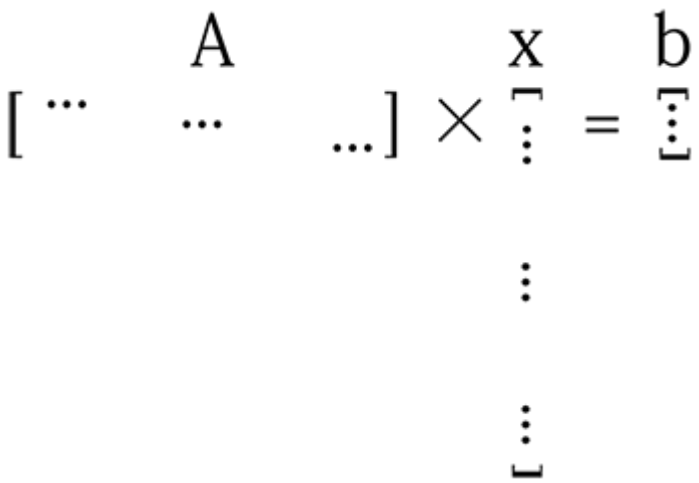
This is exactly what L1 norm regularization does. It bangs on your machine (model) to make it "dumber". So instead of simply memorizing stuff, it has to look for simpler patterns from the data. In the case of the robot, when he could remember 5 characters, his "brain" has a vector of size 5: [把, 打, 扒, 捕, 拉]. Now after regularization (banging), 4 slots of his memory became unusable. Therefore the newly learned vector is: [扌, 0, 0, 0, 0] and clearly, this is a sparse vector.

More formally, when you are solving a large vector x with less training data. The solutions to x could be a lot.

$$Ax = b$$

Here A is a matrix that contains all the training data. x is the solution vector you are looking for. b is the label vector.

When data is not enough and your model's parameter size is large, your matrix A will not be "tall" enough and your x is very long. So the above equation will look like this:



For a system like this, the solutions to x could be infinite. To find a good one out of those solutions, you want to make sure each component of your selected solution x captures a useful feature of your data. By L1 regularization, you essentially make the vector x smaller (sparse), as most of its components are useless (zeros), and at the same time, the remaining non-zero components are very "useful".

Another metaphor I can think of is this: Suppose you are the king of a kingdom that has a large population and an OK overall GDP, but the per capita is very low. Each one of your citizens is lazy and unproductive and you are mad. Therefore you command "be productive, strong and hard working, or you die!" And you enforce the same GDP as before. As a result, many people died due to your harshness, those who survived your tyranny became really capable and productive. You can think the population here is the size of your solution vector x, and commanding people to be productive or die is essentially regularization. In the regularized sparse solution, you ensure that each component of the vector x is very capable. Each component must capture some useful feature or pattern of the data.

Another way of regularization in deep learning is dropout. The idea is simple, removing some random neural connections from the neural network while training and adding them back after a while. Essentially this is still trying to make your model "dumber" by reducing the size of the neural network and put more responsibilities and pressure on the remaining weights to learn something useful. Once those weights have learned

good features, then adding back other connections to embrace new data. I'd like to think this adding back connection thing as "introducing immigrants to your kingdom when your are in short hands" in the above metaphor.

Based on this "making model dumber" idea, I guess we can come up with other similar ways to avoid over-fitting, such as starting with a small network and gradually adding new neurons and connections to the network when more data is available. Or performing a pruning while training to get rid of connections that are close to zero.

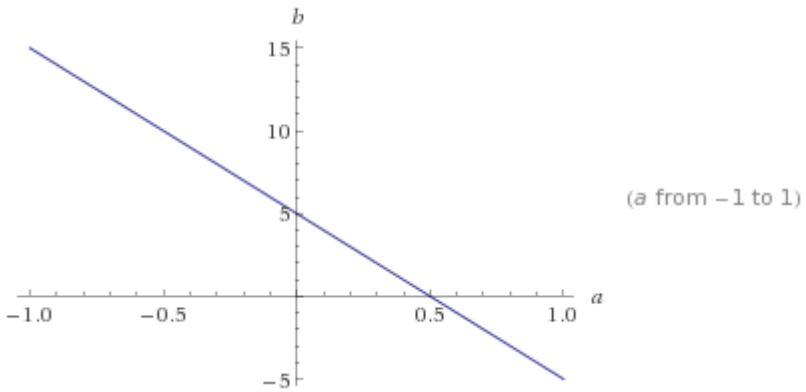
So far we have demonstrated why sparsity can avoid over-fitting. But why adding an L1 norm to the loss function and forcing the L1 norm of the solution to be small can produce sparsity?

Yesterday when I first thought about this, I used two example vectors [0.1, 0.1] and [1000, 0]. The first vector is obviously not sparse, but it has the smaller L1 norm. That's why I was confused, because looking at the L1 norm alone won't make this idea understandable. I have to consider the entire loss function as a whole.

Let's go back to the problem of Ax = b, with a simple and concrete example. Suppose we want to find a line that matches a set of points in 2D space. We all know that you need at least 2 points to fix a line. But what if the training data has only one point? Then you will have infinite solutions: every line that passes through the point is a solution. Suppose the point is at [10, 5], and a line is defined as a function y = a * x + b. Then the problem is finding a solution to this equation:

$$\begin{bmatrix} 10 & 1 \end{bmatrix} \times \begin{pmatrix} a \\ b \end{pmatrix} = (5)$$

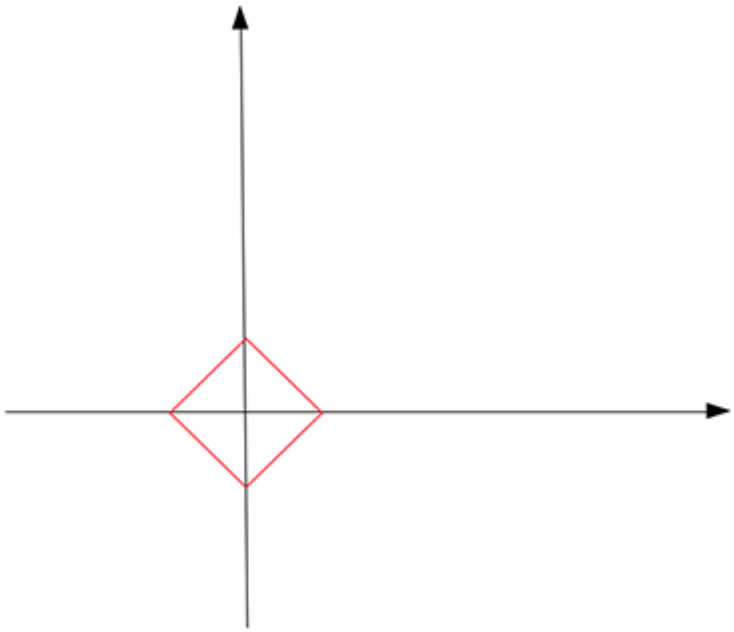
Since b = 5 – 10 * a, all points on this following line b = 5 – 10 * a should be a solution:



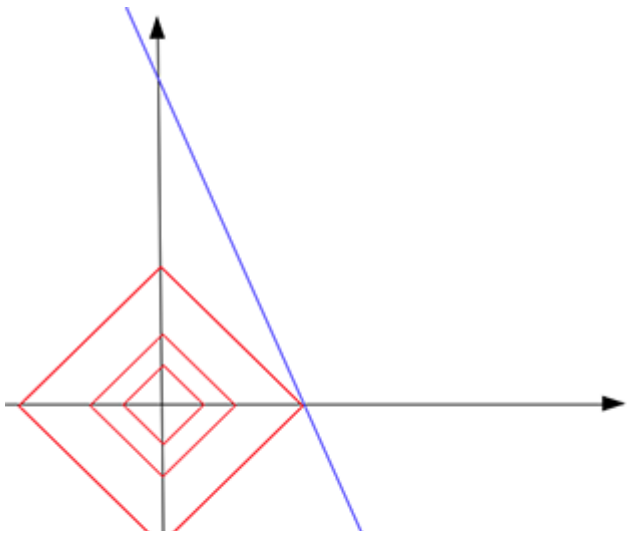
But how to find the sparse one with L1 norm?

L1 norm is defined as the summation of absolute values of a vector's all components. For example, if a vector is [x, y], its L1 norm is $|x| + |y|$.

Now if we draw all points that have a L1 norm equals to a constant c, those points should form something (in red) like this:



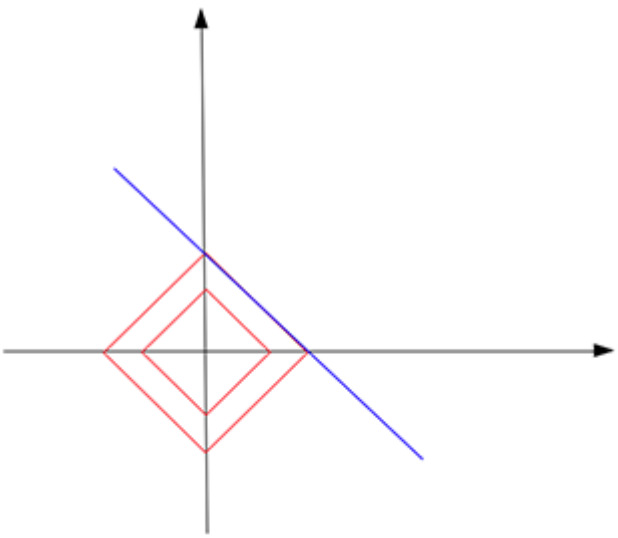
This shape looks like a tilted square. In high dimension space, it will be an octahedron. Notice that on this red shape, not all points are sparse. Only on the tips, points are sparse. That is, either x or y component of a point is zero. Now the way to find a sparse solution is enlarging this red shape from the origin by giving an ever-growing c to "touch" the blue solution line. The intuition is that the touch point is most likely at a tip of the shape. Since the tip is a sparse point, the solution defined by the touch point is also a sparse solution.



As an example, in this graph, the red shape grows 3 times till it touches the blue line b = 5–10 * a. The touch point, as you can see, is at a tip of the red shape. The touch point [0.5, 0] is a sparse vector. Therefore we say, by finding the solution point with the smallest L1 norm (0.5) out of all possible solutions (points on the blue line), we find a sparse solution [0.5, 0] to our problem. At the touch point, the constant c is the smallest L1 norm you could find within all possible solutions.

The intuition of using L1 norm is that the shape formed by all points whose L1 norm equals to a constant c has many tips (spikes) that happen to be sparse (lays on one of the axises of the coordinate system). Now we grow this shape to touch the solutions we find for our problem (usually a surface or a cross-section in high dimension). The probability that the touch point of the 2 shapes is at one of the “tips” or “spikes” of the L1 norm shape is very high. That’s why you want to put L1 norm into your loss function formula so that you can keep looking for a solution with a smaller c (at the “sparse” tip of the L1 norm). (So in the real loss function case, you are essentially shrinking the red shape to find a touch point, not enlarging it from the origin.)

Does L1 norm always touch the solution at a tip and find us a sparse solution? Not necessarily. Suppose we still want to find a line out of 2D points, but this time, the only training data is a point [1, 1000]. In this case, the solution line b = 1000 -a is in parallel to one of the edges of the L1 norm shape:



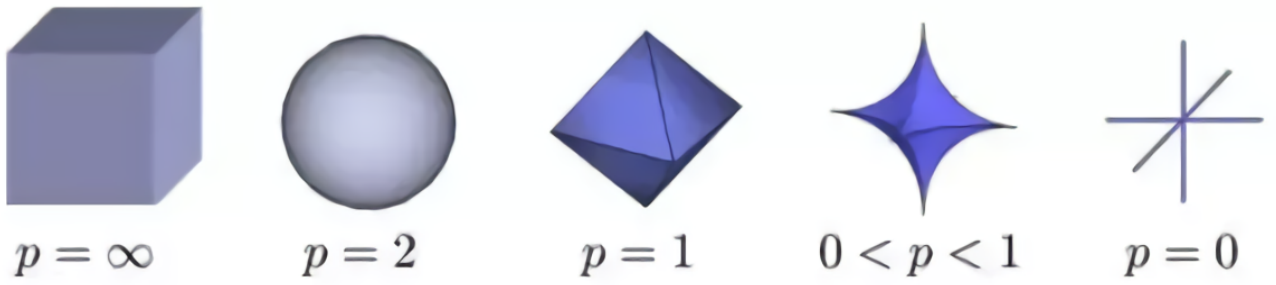
Eventually, they touch on an edge, not by a tip. Not only you can’t have a unique solution this time, most of your regularized solutions are still not sparse (other than the

two tip points.)

But again, the probability of touching a tip is very high. I guess this is even more true for high dimension, real-world problems. As when your coordinate system has more axises, your L1 norm shape should have more spikes or tips. It must look like a cactus or a hedgehog! I can't imagine.

If you push a person towards a cactus, the probability of he being pricked by the needles is pretty high. That's also why they invented this pervert weapon and that's why they want to use L1 norm.

But is the L1 norm the best kind of norm to find a sparse solution? Well, it turns out that the Lp norm when 0 <= p < 1 gives the best result. This can be explained by looking at the shapes of different norms:



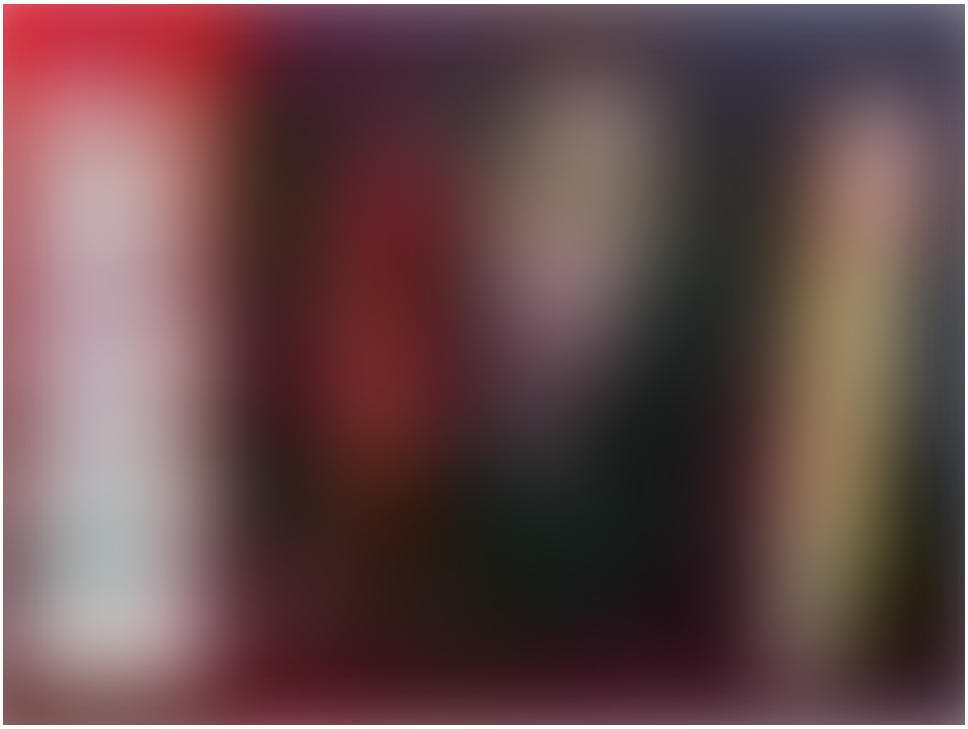
As you can see, when p < 1, the shape is more “scary”, with more sharpen, outbreaking spikes. Whereas when p = 2, the shape becomes a smooth, non-threatening ball. Then why not letting p < 1? That’s because when p < 1, there are calculation difficulties.

In conclusion, over-fitting is a problem you see when your machine learning model is too large (has too many parameters) comparing to your available training data. In this case, the model tends to remember all training cases including noisy to achieve better training score. To avoid this, regularization is applied to the model to (essentially) reduce its size. One way of regularization is making sure the trained model is sparse so that the majority of it’s components are zeros. Those zeros are essentially useless, and your model size is in fact reduced.

The reason for using L1 norm to find a sparse solution is due to its special shape. It has spikes that happen to be at sparse points. Using it to touch the solution surface will very likely to find a touch point on a spike tip and thus a sparse solution.

Just think about this:

When there is a zombie outbreak, which one should be the weapon of choice?



Machine Learning Artificial Intelligence Deep Learning Neural Networks