Security Camera

Mohit Deshpande

February 27, 2017

1 Summation Notation

Before we get started, I want to discuss a new mathematical notation. And to motivate that notation, let's suppose that you wanted to sum up the numbers between 1 and 10. We can write the sum to be $1+2+\cdots+10$. To find the value of this sum, we could easily do this with a for loop in source code $\mathit{sum} \leftarrow 0;$ for $i \leftarrow 1$ to 10 do $\mathit{sum} \leftarrow \mathit{sum} + i$ end. But we also have a mathematical shorthand for taking the sum of a list of numbers.

$$s = \sum_{i=1}^{10} i = 1+2+\cdots+10$$

That symbol is a capital sigma and is called a summation. The summation is kind of like a for loop where we keep adding or accumulating a value. For summations, we always accumulate a value or sum. The index variable i is declared at the bottom along with the lower bound. On the top of the sigma, we declare the upper bound. Both bounds are inclusive the index variable is both of those values and every integer in between. In the example above, we're adding the numbers from 1 to 10 using a summation.

Here's another example with summation notation where we have a different quantity. Only the index variable changes (i.e., the i); everything else stays the same (i.e., the 1)

$$s = \sum_{i=1}^{10} i+1 = (1+1)+(2+1)+\cdots+(10+1)$$

Specifically when dealing with images, we can take the sum of pixels across rows, columns, or both. But we're only going to consider the sum of all pixels. Let's see this with an example. Here's a 3×3 image A:

$$A = \begin{bmatrix} 5 & 100 & 50 \\ 0 & 60 & 60 \\ 1 & 200 & 50 \end{bmatrix}$$

For all of the pixels in the image, we can write that as $5+100+50+0+60+60+1+200+50 = I_{1,1}+I_{1,2}+I_{1,3}+I_{2,1}+I_{2,2}+I_{2,3}+I_{3,1}+I_{3,2}+I_{3,3} = \sum_{i.j} I_{i,j}$. This is the notational shorthand I'll be using to denote taking the sum of all of the pixels in an image.

This summation notation is going to come in handy when we start discussing image similarity because we can concisely represent different measures of similarity using them.

2 Image Similarity

As humans, we can easily differentiate between images. We can look at two images and say "these are different" or "these are the same" almost instantly. For a computer however, remember that we only have access to the raw pixel values. So how can we differentiate two images using just pixel values? Let's see an example.

$$A = \begin{bmatrix} 0 & 230 & 75 \\ 0 & 210 & 60 \\ 0 & 200 & 50 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 225 & 70 \\ 0 & 210 & 65 \\ 0 & 200 & 55 \end{bmatrix}$$

These images aren't exactly the same, but they're also not completely different. First of all, how do we know that these images aren't identical? If the two images were the same, then they would have the same pixel values in the corresponding coordinates, meaning there would be no *difference* in the pixel values! In other words, if I subtracted the two matrices, I would be left with a matrix with all 0's. It seems that we can use difference between the two images to determine if they're the same or not. This difference has to be computed for each pixel over the entire image. Explained intuitively, one measure of image similarity says we compute the image difference by looking at each of individual pixel differences, or errors, and taking the sum of each of them. That would give us a single number that we could interpret to be the error between the images.

Let's write this out mathematically.

$$\ell_1 = \sum_{i,j} |A_{i,j} - B_{i.j}|$$

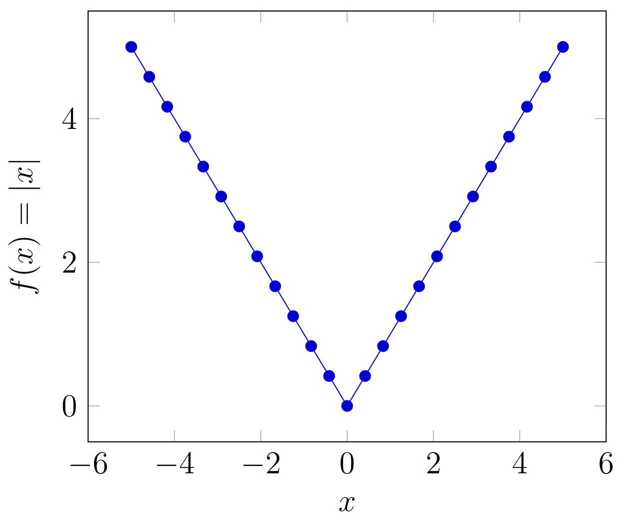
Notice that we take the absolute value of the difference. This is because an error of -2 and and error 2 are both errors, just in different directions. In fact, if we didn't have the absolute value there, then $A - B$ would actually return 0, meaning that there is no difference! We don't want there to be more or less error depending on how we subtract the images! This metric of image similarity is called ℓ_1 **norm** or ℓ_1 **error** and it's used to measure how different two images are from their pixel values.

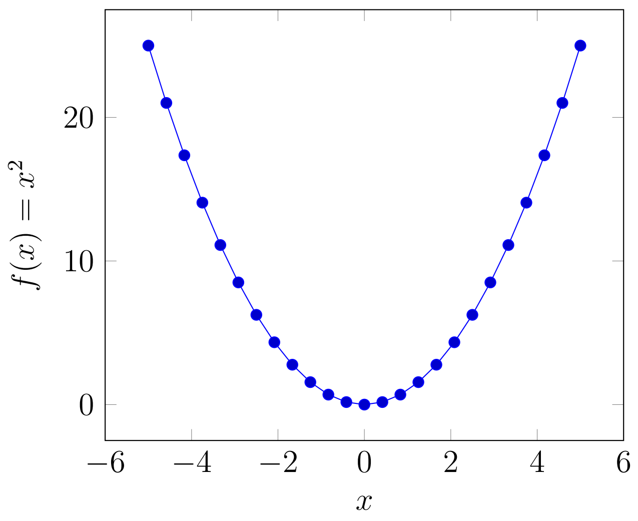
3 Sum of Squared Errors (SSE)

ℓ_1 is good, but we can improve upon it by a little bit. In particular, we're going to simply replace the absolute value of ℓ_1 with a square instead. So let's replace our absolute value with a square. And this special type of norm or error is called **Sum of Squared Errors (SSE)**

$$SSE = \sum_{i,j} (A_{i,j} - B_{i.j})^2$$

Before saying why we do this, I first want to convince you that squaring has at least all of the important properties of absolute value that we need.





For both absolute value and square, all outputs are at least 0. We square values instead of take absolute value because x^2 a smooth function. This means that it doesn't have that point that absolute value. You might think: "why does it matter if it's pointy or smooth?". This makes a big difference in terms of calculus. Smooth functions have all sorts of nice properties that pointy functions don't have in terms of calculus. And we generally use square because of that reason: the math works out easier if we have to do any calculus. We won't be doing anything with calculus, but this type of norm or error is used for so many things, particular machine learning and AI, and in that context, there's a lot of calculus and linear algebra so they prefer smooth functions.

4 Mean Squared Error

SSE is a pretty good way of measuring error but there's one major problem with it. To illustrate this example, let's consider two images.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \ B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Let's compute the SSE of these two images. Remember that SSE is this.

$$SSE = \sum_{i,j} (A_{i,j} - B_{i.j})^2$$

Intuitively, we take each pixel from corresponding coordinates, subtract them, square that difference, and add it up for all pixels. If we do this, we should get 9. Now let's take these images and make them even bigger. Suppose these matrices are both 10×10 matrices.

$$C = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, \ D = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

Now let's compute the SSE again. Basically, at each point in the image, we get a difference of 1. Then take the sum over all pixels. We get an error of 100! But this is not good! We just took the previous two matrices and expanded them by just 7 rows and 7 columns and we get more than 10 times the error! They contain exactly the same information but the latter two matrices are just a little bit bigger! We need to modify our SSE formula to account for the image size.

We can account for image size by dividing the result of SSE with the total number of pixels in the images. How can we determine this? Just multiply the rows and columns of the images together! In that sense, we're kind of "averaging" the error across all pixels. This is why we call this new metric **Mean Squared Error**. We can express it mathematically like this.

$$MSE = \frac{1}{mn} \sum_{i,j} (A_{i,j} - B_{i.j})^2$$

Now let's compute the MSE between A, B and C, D. We can start with the SSE result and divide by the image size. In both cases, we get exactly 1!

Now our metric is robust to image sizes! Although SSE is still a valid metric, MSE works a bit better for images so we'll be using that primarily.

5 Structural Similarity (SSIM)

MSE is a really good measure of error or difference. However, the issues with SSE and MSE is that they only look at each pixel individually. That's not how humans perceive images though. We don't look at each pixel of two images and compare them. We look at the images holistically. We can't see all of the small variations and noises in our image and we don't really care. We could add a small value to each pixel, and we, as humans, wouldn't be able to perceive the difference but MSE certainly will!

Instead of pixel-wise comparisons, **Structural Similarity** looks at groups of pixels as a whole to better determine if two images actually differ. Structured similarity is actually a produce of three other measures: luminance, contrast (related in a statistical sense to adding contrast to an image), and structure. Here's the equation, but I won't get into the details since it requires a good amount of stats knowledge.

$$SSIM(X,Y) = l(X,Y) \cdot c(X,Y) \cdot s(X,Y)$$

X, Y is the pixels in the $N \times N$ window that we consider. Usually $N = 8$ is a good value. This isn't like convolution where we do that computation, the window is just a window: it determines which group of pixels we consider in the image. Instead of looking at absolute pixel values, notice that we're looking at averages and spread of pixel values, like a histogram. This gives us a better representation of our image.

All of these metrics actually use statistics across a neighborhood window (similar to a kernel) to determine similarity. By looking at groups of pixels instead of individual pixels independently, we get a better idea of the big-picture, no pun intended, idea of what the image contains. To clarify, SSIM does not use any sort of image classification to determine what is actually in the image, but it uses statistics on pixels.

The good thing about SSIM however is that it can only take on a range of possible values. An image with perfect similarity will produce an SSIM value of 1 and an image with perfect dissimilarity will produce an SSIM value of -1. Unlike MSE and SSE, it is bounded above and below.