Linear Algebra and Machine Learning: A Simple Example

your russkie friend

October 23, 2013

1 A sum of products

Consider the following series:

$$a_1x_1 + a_2x_2 + ... + a_{n-1}x_{n-1} + a_nx_n = \sum_{i=1}^{n} a_ix_i$$

It so happens that, using a tiny bit of linear algebra notation – an *transpose* and *a dot product* (aka *inner product* for our purposes) – the following is also true:

$$\sum_{i=1}^{n} a_ix_i = A^T X$$

where A is a vector (i.e. a one-column matrix) that contains all the a values, where X is a vector with all the x values, and where A^T means taking the *transpose* of A (which in this case is simply writing it out as a row vector instead of a column vector).

2 Easier to read code (no `for` loops) and faster run time (optimized libraries)

In the example above, "$A^T X$" is arguably tighter (i.e. more concise) notation than "$\sum_{i=1}^{n} a_ix_i$", which matters when you have a lot of these sums, perhaps even nested inside one another. But more importantly and less arguably, your code will be easier to read and will run faster too.

Here's what a sum of products might look like once implemented in C++ for example:

```
double sum = 0;
for (int i = 0;  i < n;  i++)
{
   sum += a[i] * x[i];
}
```

But with a bit of linear algebra notation and the right library, this code could be simply (re)written as follows:

```
A.transpose() * X;
```

You could use the "`*`" for matrix multiplication because in C++ it's possible to overload operators. And Matlab code would read even simpler since the transpose of `A` would simply be `A'`. But the main thing is: no `for` loops! Pretty cool :). And your code will run faster too if you use linear algebra libraries since they are highly optimized these days. When you have a lot of sums and things, sometimes all nested, this can really simplify things and speed them up too. Or so I get the impression.