Subject: Re: fast convolving question
Posted by Chris[5] on Fri, 30 May 2008 20:06:24 GMT
View Forum Message <> Reply to Message

On May 30, 2:44 am, rog...@googlemail.com wrote:
> Dear Chris,
> thank you again for your reply and the amount of time you invested.
>
> To understand, what I mean, it seems to be better to explain it for
> very small matrices.
>
> So, let's say you have a dist(3) kernel and a dist(7) matrix.
> At first to overcome the problem with negative indices of the strict
> numerical solution of convolving matrices, I padded the matrix in each
> direction with 2 zeros, so the resulting matrix is now 9x9 (0,matrix,0
> in x- and y-direction).
>
> Then I pre-compute indices to speed up the process (main idea):
>
> 1.For the kernel: 0 - 8 + reform to vector
>
> 2.0. For the Matrix (first vector): 20-19-18-12-11-10-2-1-0(=indsmall)
> + reform to vector and insert it into matrix -
>
>> mat(20-19-18-12-11-10-2-1-0 + ind(0)) <- (ind(0) is 0)
>
> 2.1. For the Matrix (second vector): 29-28-27-20-19-18-12-11-10 +
> reform to vector and insert it into matrix -
>
>> mat(20-19-18-12-11-10-2-1-0 + ind(1)) <- (ind(1) is 9)
>
> till 2.48. 80-79-78....
>
> 3. As third step I multiply kernel-vector with the mat-vectors, so:
>
> conv(0) = kernel ## mat(indsmall+ind(0))
> conv(1) = kernel ## mat(indsmall+ind(1))
> ...
> conv(48)= kernel ## mat(indsmall+ind(48))
>
> 4. Reform conv to 7x7 and return it
>
> The trick is to only multiply the kernel as vector with the reformed
> submatrix of the matrix. I tested all types of convolving - the above
> code is only a snippet - and the fastest one were always my
> unfortunately not right indexing no-for-loop.
>
> Besides that strict convolving is a very simple scheme. Just

> multiplying the always same kernel as vector with the
> i.subarray(padded with zeros at the edges) of matrix(ixj) as vector
> (beginning from down right to upper left) and repeating this ixj
> times. Reform the given result back again to matrix.
>
> But unfortunately, only the loop-method for k=0,48 do conv(k) = ...
> works perfectly.
>
> I found several methods to convolve discrete without any loops, but
> they are always slower than fft or my one-loop-method, except the no-
> loop-method which is more than 100 times faster than fft or convol.
>
> So, please, please, please help me again and try to implement e.g.
> indgen as the for-to-loop
>
> Thanks and best regards
>
> Christian

Here's what you need to do:

You are trying to matrix multiply one vector with many different vectors. For the i'th multiplication, the second vector needs to be mat(indsmall+ind(i)). Since matrix multiplication multiplies one row of the first matrix by one column in the second, we need to make a matrix where the ith column is mat(indsmall+ind(i)). Adding the following lines of code after the else begin portion of the discrete method will do this:

```
i=indarray2[0:sm-1]
i1=transpose(rebin(indsmall,sm,sm))
i2=rebin(ind(i),sm,sm)
kernel=reform(kernel)
(conv)[i]=kernel##mat(i1+i2)
endelse
```

However, this new result is much, much slower than even the loop. I think there's a lot of overhead in rebinning indsmall and ind, though I admit I don't understand why.

To stress my earlier points a bit more, however, you should not be getting excited that your incorrect method is 100x faster. A discrete convolution of 2 NxN arrays requires 2*N^4 arithmetic operations (for each of N^2 output pixels, multiply NxN numbers and add those NxN numbers together). Your earlier incorrect method only performed 2*N^2 operations (it correctly computed the first pixel's value). It was 100x faster, but performed 1/N^2 of the total work needed. For N=100,

it did 1/10,000 of the work in 1/100 of the time. That is NOT faster!
Even if you get around the time penalties in my code that come from
creating the big arrays, you're current method is doomed to lose.

Also, there is a difference between a 'simple' scheme and an
'inexpensive' scheme. Discrete convolution may be straightforward to
understand, but it scales as N^4. You will NEVER get around doing 2N^4
operations in discrete convolution, so it's going to become a slow
process if both of your arrays are huge. Convol is packaged with IDL.
It's not written in the IDL language (which I hear makes a procedure a
bit slower), and is a mature function (introduced with IDL 1). I
highly doubt that it is doing the (necessary!) 2N^4 arithmetic
operations in a way that is less optimized than how you or I would do
it. Let me again stress that convol is much faster than your fft
algorithm for modest arrays and, for these array sizes, scales better
with increasing N. It may choke with large N when the fft comes into
its own, but I would bet that at that point BLK_CON would do the
trick.

Out of curiosity, what application are you working on that requires
both the input array and the convolution kernel to be large?

Cheers,
Chris