Subject: Re: fast convolving question
Posted by Chris[5] on Thu, 29 May 2008 09:41:59 GMT
View Forum Message <> Reply to Message

On May 28, 5:16 am, rog...@googlemail.com wrote:
```
> Dear all,
>
> the following code is able to convolve 2 matrices (e.g. 100x100 with
> 100x100) very fast by using 3 different approaches.
> The first one is fft. The second one is based on pre-computing indices
> (only ~6 times slower than fft by the given size) and multiplying
> indexed vectors with 1 for-loop and the third one is without any loop
> (~100 times faster than fft by the given size).
>
> My aim i.e the task is to convolve always without the use of fft, so i
> made this script. Unfortunately something is going wrong with the
> third method and I can't find where the error is.
>
> Please, help me!
>
> Thanks and king regards
>
> Christian
>
> Here it is:
>
> Function convolve, a,b,method,loop=loop
>
> if method eq 'fft' then begin
>
>       s1=size(a)
>       s2=size(b)
>       nx1=s1[1]
>       ny1=s1[2]
>       nx2=s2[1]
>       ny2=s2[2]
>       aa=fltarr(nx1+nx2-1,ny1+ny2-1)
>       bb=fltarr(nx1+nx2-1,ny1+ny2-1)
>       aa[0,0]=a
>       bb[nx1-1,ny1-1]=b
>       conv=double(shift(fft(fft(aa,-1)*fft(bb,-1),
> 1)*n_elements(aa),nx2,ny2))
>       return, conv
> endif
>
> if method eq 'discrete' then begin
>
> kernel  =      a
> matrix  =      b
>
> siz_k   =      size(kernel,/dimensions)
> sx_k    =      siz_k[0]
> sy_k    =      siz_k[1]
> sk             =      sx_k*sy_k
>
> siz_m   =      size(matrix,/dimensions)
> sx_m    =      siz_m[0]
> sy_m    =      siz_m[1]
> sm             =      sx_m*sy_m
>
> conv    =      fltarr(sm,/nozero)
> mat            =      fltarr(sx_k+sx_m-1,sy_k+sy_m-1)
> ;padding matrix with zeros
>  mat[((sx_k-1)/2):((sx_k-1)/2+sx_m-1),((sy_k-1)/2):((sy_k-1)/ 2+sy_m-1)]
> =      matrix
> ;compute indices
> indarray2      =      make_array(sx_m+sx_k-1,sy_m+sy_k-1,/index)
> indarray       =      transpose(indarray2)
> ind                  =      reform(indarray[0:sx_m-1,0:sy_m-1],sm,/overwrite)
> indsmall       =      (indarray[0:sx_k-1,0:sy_k-1])(reverse(indarray2[0:sk-1]))
> kernel         =      reform(kernel,sk,1,/overwrite)
>
> ;convolve by multiplying vectors
> if keyword_set(loop) then $
>        for i=0,sm-1 do (conv)[i]=kernel##mat( indsmall+ind(i) ) $
>
>        else (conv)[(i=indarray2[0:sm-1])]=kernel##mat( indsmall+ind(i) )
>
> conv           =      reform(conv,sx_m,sy_m,/overwrite)
> return, conv
> endif
> end
>
> pro test_method_conv
>
> par     =      100
> a       =      dist(par)
> b       =      dist(par)
>
> t0=systime(1)
> c1=convolve(a,b,'fft')
> print,'Convolve with fft: ',systime(1)-t0,' seconds'
> Window,1,xsize=200,ysize=200,title='Convolve with fft' &
> TVSCL,congrid(c1,100,100)
>
> t0=systime(1)
> c2=convolve(a,b,'discrete')
> print,'Convolve discrete without loop: ',systime(1)-t0,' seconds'
> Window,2,xsize=200,ysize=200,title='Convolve discrete without loop' &
> TVSCL,congrid(c2,100,100)
>
> t0=systime(1)
> c3=convolve(a,b,'discrete',/loop)
> print,'Convolve discrete with loop: ',systime(1)-t0,' seconds'
> Window,3,xsize=200,ysize=200,title='Convolve discrete with loop' &
> TVSCL,congrid(c3,100,100)
> end
```

Hmm- why don't you just use the build in IDL function convol?

Subject: Re: fast convolving question
Posted by rogass on Thu, 29 May 2008 11:38:08 GMT

View Forum Message <> Reply to Message

Because of the time which the in-built routine consumes while
convolving large arrays with large arrays.

But, did you see the error, why:

else (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))

only works for the first ind(i=0) and then repeats it by using always
ind(0) instead of ind(i)?

It this a bug or error in code?

Tanks and best regards

Christian

Subject: Re: fast convolving question
Posted by Chris[5] on Thu, 29 May 2008 19:47:57 GMT

View Forum Message <> Reply to Message

> But, did you see the error, why:
>
> else (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))
>
> only works for the first ind(i=0) and then repeats it by using always
> ind(0) instead of ind(i)?
>

> It this a bug or error in code?
>

Both kernel and mat(indsmall+ind(i)) are vectors of length sm. When
you matrix multiply them, the answer is single number. So you are just
asking to fill in conv with the same number at every element given by
i.

Subject: Re: fast convolving question
Posted by rogass on Fri, 30 May 2008 07:44:18 GMT
View Forum Message <> Reply to Message

On May 29, 9:47 pm, Chris <cnb4s...@gmail.com> wrote:
>> But, did you see the error, why:
>
>> else (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))
>
>> only works for the first ind(i=0) and then repeats it by using always
>> ind(0) instead of ind(i)?
>
>> It this a bug or error in code?
>
> Both kernel and mat(indsmall+ind(i)) are vectors of length sm. When
> you matrix multiply them, the answer is single number. So you are just
> asking to fill in conv with the same number at every element given by
> i.

Ah, ok - THANKS. But how can I fix this?

If I would:

(conv)[(i=0)]=kernel##mat(indsmall+ind(i)) ->result right
(conv)[(i=1)]=kernel##mat(indsmall+ind(i)) ->result right
and so on..

But if I would:

(conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))

or

(conv)[(i=indgen(sm))])]=kernel##mat(indsmall+ind(i))

then it does what you said. That makes no sense to me. How can I do it
parallel for all indices given by i?

If the code would work, its 100 times faster than fft - and that is the aim.

Please help again.

Thanks and best regards

Christian

Subject: Re: fast convolving question
Posted by Chris[5] on Fri, 30 May 2008 10:15:44 GMT
View Forum Message <> Reply to Message

On May 29, 9:44 pm, rog...@googlemail.com wrote:
> On May 29, 9:47 pm, Chris <cnb4s...@gmail.com> wrote:
>
>>> But, did you see the error, why:
>
>>> else (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))
>
>>> only works for the first ind(i=0) and then repeats it by using always
>>> ind(0) instead of ind(i)?
>
>>> It this a bug or error in code?
>
>> Both kernel and mat(indsmall+ind(i)) are vectors of length sm. When
>> you matrix multiply them, the answer is single number. So you are just
>> asking to fill in conv with the same number at every element given by
>> i.
>
> Ah, ok - THANKS. But how can I fix this?
>
> If I would:
>
> (conv)[(i=0)]=kernel##mat(indsmall+ind(i)) ->result right
> (conv)[(i=1)]=kernel##mat(indsmall+ind(i)) ->result right
> and so on..
>
> But if I would:
>
> (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))
>
> or
>
> (conv)[(i=indgen(sm))])]=kernel##mat(indsmall+ind(i))
>
> then it does what you said. That makes no sense to me. How can I do it
> parallel for all indices given by i?

>
> If the code would work, its 100 times faster than fft - and that is
> the aim.
>
> Please help again.
>
> Thanks and best regards
>
> Christian

Let me preface this by saying I don't fully understand your code, so
I'm not sure if this is correct or helpful. But here goes:

1)matrix(indsmall)=10,000 element vector
2)matrix(indsmall+ind(0 or 1 or...)= 10,000 element vector, different
from above and different for 0,1,etc
3)matrix(indsmall+ind)=10,000 element vector different from all of
these

the loop fills in every pixel of convol by matrix multiplying by one
of the many matrices in group 2. The non-loop fills in every pixel of
convol by matrix multiplying by the SAME matrix in 3. Hence the
difference.

How do you fix it? Hmmm not sure. If you want to do this by matrix
multiplication, you need to matrix multiply kernel (10,000 elements)
by a 10,000 by 10,000 matrix. Each row (or column, not sure) of the
matrix needs to be matrix(indsmall+ind(i)), where i is different for
each row/column. I wouldn't know how to construct this larger array
without looping, though I'm sure there's a way.

Note, however, that you should NOT be thinking that a non-looping
method is really going to beat the fft method based on your current
program. The non-loop does the necessary amount of arithmetic for
filling in one conolution pixel (multiplying 10,000 elements by 10,000
elements and adding). It then copies this 10,000 times. So while its
100x faster now, if you were to do 10,000 unique pixel calculations
(which you'll have to eventually do), the method will be 100x slower.

Also, your fft and loop outputs are different. In fact, the outputs
aren't even the same dimension. Are you sure that these other two are
doing what you want? Using convol (and only convolving with
kernel[0:98,0:98], since the kernel needs to be smaller than the image
for some reason) gives yet a third answer.

Finally, I'm really skeptical that you will figure out a method
significantly faster than blk_con or convol. Convol is 70x faster than
your fft method with 400x400 arrays, and 90x faster with 600x600

arrays. Convolution is an expensive process that you may have to live
with for 2 large arrays (though I haven't seen many applications where
the kernel needs to be so large). Also remember that, if you are using
an FFT method, the input arrays really need to be powers of 2 (64 or
128, not 100) for the fft speed benefit. Padding a 100x100 image with
zeros up to a 128x128 image will make the fourier transform much more
efficient.

Anyways, sorry this doesn't directly fix the bug in your code.
Hopefully, though, you can figure out a way to deal with an
intrinsically expensive computation (or perhaps find a way to use a
smaller kernel??)

Chris

Subject: Re: fast convolving question
Posted by rogass on Fri, 30 May 2008 12:44:58 GMT
View Forum Message <> Reply to Message

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

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

Subject: Re: fast convolving question
Posted by rogass on Mon, 09 Jun 2008 15:39:00 GMT

View Forum Message <> Reply to Message

On 30 Mai, 22:06, Chris <cnb4s...@gmail.com> wrote:
> 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

Dear Chris,
I reviewd my methods and you are right. Thank you for the time you
took to understand what i was wanting to do.

I will use fft or convol in the future, because it's much easier to
handle.

Best regards

Christian