Rutgers University
Summer Research Program

Preparation for GpyTorch Implementation

Han Liu

August 7, 2019

1 Introduction

In this week, I have read the second paper "Exact Gaussian Processes on a Million Data Points" and have finished all the preparation work for the final GpyTorch Implementation by Matlab. Specifically, firstly I have fixed the error of Pivoting Cholesky Decomposition, and try to implement it with Matlab. Secondly, I use the result of Pivoting Cholesky Decomposition as a preconditioner to the conjugate gradient algorithm, the result shows the preconditioning accelerate the convergence significantly. Thirdly, I have figured out the Stochastic Trace Estimation, and the MATLAB results shows the fast convergence of the algorithm with the increasing of sampling times. Finally, I have figured out the innovative algorithm proposed by the second paper.

In the following week, I will implement the whole Gaussian Process (GP) using GPU acceleration algorithm proposed by this paper, and I will use Adam optimizer to train GP, on which I am currently working.

2 The Pivoting Cholesky Decomposition

In the process of implementing the pivoting cholesky decomposition (PCC), I found the algorithm I derived last week had some problems, so I have implemented it again. Specifically, we need to find $P^T K P = R^T R$ for symmetric positive matrix $K \in R^{n \times n}$, where P is permutation matrix. Following the derivation of the pivoting cholesky decomposition in the paper, we need to use one propertiy of permutation matrix:

$$P^{-1} = P^T \tag{1}$$

The mathematical details are given in appendix. Specifically, the whole algorithm should be as following [1]: We use a random matrix D to be the matrix waiting for decomposition, and a simple provement can be seen as following: The maximum diagonal element of original matrix is at the bottom right corner, after the decomposition, the reconstructed result is exchanged to the top left corner, and after exchanging the pivoting row and column of original matrix, they are the same.
The main objective of using PCC is to use it as a preconditioner for conjugate gradient algorithm (CG). In this process, we need to satisfy:

$$(M + \sigma^2 I)^{-1} \hat{K} \approx I \tag{2}$$

$\mathbf{R} = \mathbf{0}$ {define a $n \times n$ zero matrix}
$piv = 1 : n$
for $k = 1$ to n **do**
  $q = \{i : \mathbf{A}(i,i) = \max(diag(\mathbf{A}(k:n, k:n)))\} + k - 1$ {Finding the pivot}
  if Stopping criterion **then**
    stop {rank of $\mathbf{A}$ is $(k-1)$}
  end if
  $\mathbf{A}(:,k) \leftrightarrows \mathbf{A}(:,q)$ {Swap columns}
  $\mathbf{R}(:,k) \leftrightarrows \mathbf{R}(:,q)$ {Swap columns}
  $\mathbf{A}(k,:) \leftrightarrows \mathbf{A}(q,:)$ {Swap rows}
  $piv(k) \leftrightarrows piv(q)$ {Swap pivoting position}
  $\mathbf{R}(k,k) = \sqrt{\mathbf{A}(k,k)}$
  $\mathbf{R}(k, k+1:n) = \mathbf{R}(k,k)^{-1}\mathbf{A}(k, k+1:n)$
  $\mathbf{A}(k+1:n, k+1:n) = \mathbf{A}(k+1:n, k+1:n) - \mathbf{R}(k, k+1:n)^T\mathbf{R}(k, k+1:n)$
end for

Figure 1: Pseudo-code for Pivoting Cholesky Decomposition

Where M is the result of PCC of K, $\hat{K} = K + \sigma^2 I$. In order to satisfy the requirement of Eq. 2, we need to rearrange the results of PCC, this can be done by:

$$K = (P^T)^{-1} R^T R P^{-1} = P R^T R P^T \tag{3}$$

The Matlab code is as following:

```
%Implementation of Pivoted Cholesky Composition
function [ G ] = Pivited_Cholesky_Composition( A )
n=size(A,1);
v=randperm(n);
Pi = sort(v);
l=zeros(n,n);   %triangular matrix we want
P=zeros(n,n);
for k=1:n
[¬, p]=max(diag(A(k:n,k:n)));
i=p+k-1;

a=A(:,k);   %exchange column
A(:,k)= A(:,i);
A(:,i)=a;

b=l(:,k);       %exchange L
l(:,k)=l(:,i);
l(:,i)=b;

c=A(k,:);       %exchange row
A(k,:)=A(i,:);
A(i,:)=c;

a=Pi(k);        %exchange pi
Pi(k)=Pi(i);
Pi(i)=a;

l(k,k)=sqrt(A(k,k));
l(k,k+1:n)=l(k,k)\A(k,k+1:n);
A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-l(k,k+1:n)'*l(k,k+1:n);
```

```
D =

    2.6957   -3.5093    0.8201    0.6326    1.8275
   -3.5093    5.2841   -1.5381   -0.1579   -1.5306
    0.8201   -1.5381    3.2358   -0.8851   -2.9301
    0.6326   -0.1579   -0.8851    1.4629    2.7039
    1.8275   -1.5306   -2.9301    2.7039    6.7265


D_PCC =

    6.7265   -1.5306   -2.9301    1.8275    2.7039
   -1.5306    5.2841   -1.5381   -3.5093   -0.1579
   -2.9301   -1.5381    3.2358    0.8201   -0.8851
    1.8275   -3.5093    0.8201    2.6957    0.6326
    2.7039   -0.1579   -0.8851    0.6326    1.4629
```

Figure 2: Pseudo-code for Pivoting Cholesky Decomposition

```
31  P(Pi(k),k)=1;
32  end
33  Am=l'*l;
34  %exchange back the pivoted position
35  G=P*Am*P';
```

3 Preconditioning

The basic idea of preconditioning is to introduce a matrix P to solve the linear system:

$$P^{-1}\hat{K}u = P^{-1}y \tag{4}$$

The preconditioning will guarantee to have the same solution with the original linear system, and the system's convergence is depend on the conditioning of $P^{-1}\hat{K}$ rather than $\hat{K}$. Furthermore, if $P^{-1}\hat{K} \approx I$, the system will converge fastly, thus we will use the decomposition result of PCC as preconditioner. As for the implementation of preconditioned conjugate gradients (PCG), there are some mistakes of the pseudo-code given in the paper, the correct version is given as following:

Algorithm 1: Standard preconditioned conjugate gradients (PCG).

Input : mvm_A() – function for matrix-vector multiplication (MVM) with matrix A
$\mathbf{b}$ – vector to solve against
P^{-1}() – function for preconditioner

Output : $A^{-1}\mathbf{b}$.

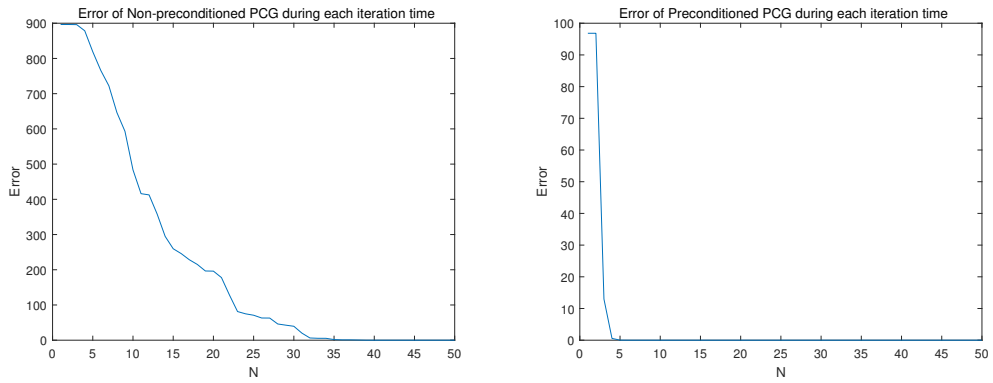
$\mathbf{u}_0 \leftarrow \mathbf{0}$ // Current solution
$\mathbf{r}_0 \leftarrow$ mvm_$A(\mathbf{u}_0)$ - $\mathbf{b}$// Current error $r_0 \leftarrow b - \text{mvm\_}A(u_0)$
$\mathbf{z}_0 \leftarrow P^{-1}(\mathbf{r}_0)$ // Preconditioned error
$\mathbf{d}_0 \leftarrow \mathbf{z}_0$ // "Search" direction for next solution

for $j \leftarrow 0$ **to** T **do**
  $\mathbf{v}_j \leftarrow$ mvm_$A(\mathbf{d}_{j-1})$
  $\alpha_j \leftarrow (\mathbf{r}_{j-1}^\top \mathbf{z}_{j-1})/(\mathbf{d}_{j-1}^\top \mathbf{v}_j)$
  $\mathbf{u}_j \leftarrow \mathbf{u}_{j-1} + \alpha_j \mathbf{d}_{j-1}$
  $\mathbf{r}_j \leftarrow \mathbf{r}_{j-1} - \alpha_j \mathbf{v}_j$
  if $\|\mathbf{r}_j\|_2 <$ tolerance **then return** $\mathbf{u}_j$;
  $\mathbf{z}_j \leftarrow P^{-1}(\mathbf{r}_j)$
  $\beta_j \leftarrow (\mathbf{z}_j^\top \mathbf{z}_j)/(\mathbf{z}_{j-1}^\top \mathbf{z}_{j-1})$ $\beta_j \leftarrow (z_j^T r_j)/(z_{j-1}^T r_{j-1})$
  $\mathbf{d}_j \leftarrow \mathbf{z}_j - \beta_j \mathbf{d}_{j-1}$ $d_j \leftarrow (z_j + \beta_j d_{j-1})$
end

return $\mathbf{u}_{j+1}$

Figure 3: Algorithm of Preconditioned Conjugate Gradients

I have also compared the results of Non-preconditioning PCG with Preconditioning PCG, their results are as following:



(a) Non-preconditioning (b) Preconditioning

Figure 4: Comparison of Two Implementations

The results show the preconditioning accelerates the speed of convergence significantly. The Matlab code is as following:

```
%Standard preconditioned conjugate gradients
function [ c ] = Standard_PCG( mvm_A,b, M )
n=size(mvm_A,1);
T=50;         %number of iteration
e=0.0001;    %tolerance


%initialization
u=zeros(n,T);
r=zeros(n,T);
z=zeros(n,T);
```

```
d=zeros(n,T);
v=zeros(n,T);
alpha=zeros(1,T);
beta=zeros(1,T);
truth=inv(mvm_A)*b;

%main program
r(:,1)=b-mvm_A*u(:,1);          %error fixed
z(:,1)=M*r(:,1);            %changes to PCG
d(:,1)=z(:,1);
E=zeros(1,T);

for j=2:T
v(:,j)=mvm_A*d(:,j-1);
alpha(j)=(z(:,j-1)'*r(:,j-1))./(d(:,j-1)'*v(:,j));
u(:,j)=u(:,j-1)+alpha(j)*d(:,j-1);
r(:,j)=r(:,j-1)-alpha(j)*v(:,j);
if norm(r(:,j),2)<e
break
end
z(:,j)=M*r(:,j);        %changes to PCG
beta(j)=(z(:,j)'*r(:,j))./(z(:,j-1)'*r(:,j-1));     %error fixed
d(:,j)=z(:,j)+beta(j)*d(:,j-1);                 %error fixed
E(j)=norm(truth-u(:,j),1);
end
E(1)=E(2);
n=1:T;
figure(1);
plot(n,E);xlabel('N');ylabel('Error');title('Error of Preconditioned PCG during ...
    each iteration time');
c=E;
```

4 Stochastic Trace Estimation

Stochastic Trace Estimation (STE) [2] is useful when determining $log|\hat{K}|$, $tr(\hat{K}^{-1}\frac{\partial \hat{K}}{\partial \theta})$, specifically, we have:

$$Tr(\hat{K}^{-1}\frac{\partial \hat{K}}{\partial \theta}) = E[z_i^T \hat{K}^{-1}\frac{\partial \hat{K}}{\partial \theta} z_i] \tag{5}$$

$$log|\hat{K}| = Tr(logT) = E[z_i^T (logT) z_i] \tag{6}$$

Where the mean can be estimated by arithmetic mean:

$$E[z_i^T (logT) z_i] = \frac{1}{N}\sum_{i=1}^{N} z_i^T (logT) z_i \tag{7}$$

The arithmetic mean converges to the mean with the increasing of N, which can be illustrated by the following graph:

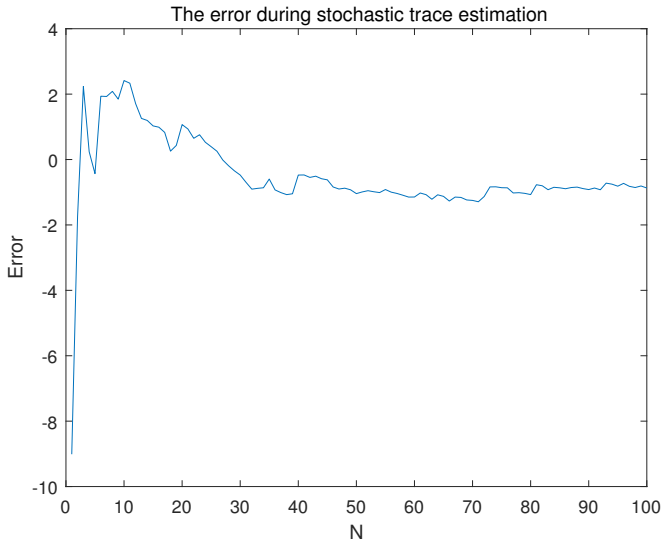


Figure 5: Convergence of Stochastic Trace Estimation

5 Partitioned kernel MVMs

The second paper (Exact Gaussian Processes on a Million Data Points) propose a new intuitive method of calculating MVM. In the process of PCG, the computation which takes the most memory usage is $v = \hat{K} * d$, which is costs $O(n^2)$ memory. We can divide data matrix X into p partitions:

$$X = [X^{(1)}; ...; X^{(p)}] \tag{8}$$

where we use " ;" to denote row-wise concatenation. Then the kernel matrix can be concatenated as:

$$\hat{K} = [\hat{K}_{X^{(1)}X}; ...; \hat{K}_{X^{(p)}X}] \tag{9}$$

Then we can rewrite the matrix-vector product $\hat{K}v$ as:

$$\hat{K}v = [\hat{K}_{X^{(1)}X}v; ...; \hat{K}_{X^{(p)}X}v] \tag{10}$$

This process is very intuitive, when the iteration time $p \to n$, the PCG only needs $O(n)$ memory.

References

[1] L. S. Bastos, A. O. Hagan, "Pivoting Cholesky Decomposition applied to Emulation", June, 2007.

[2] Fika, Paraskevi, and Christos Koukouvinos. "Stochastic estimates for the trace of functions of matrices via Hadamard matrices." Communications in Statistics-Simulation and Computation 46.5 (2017): 3491-3503.

A Derivation of PCD

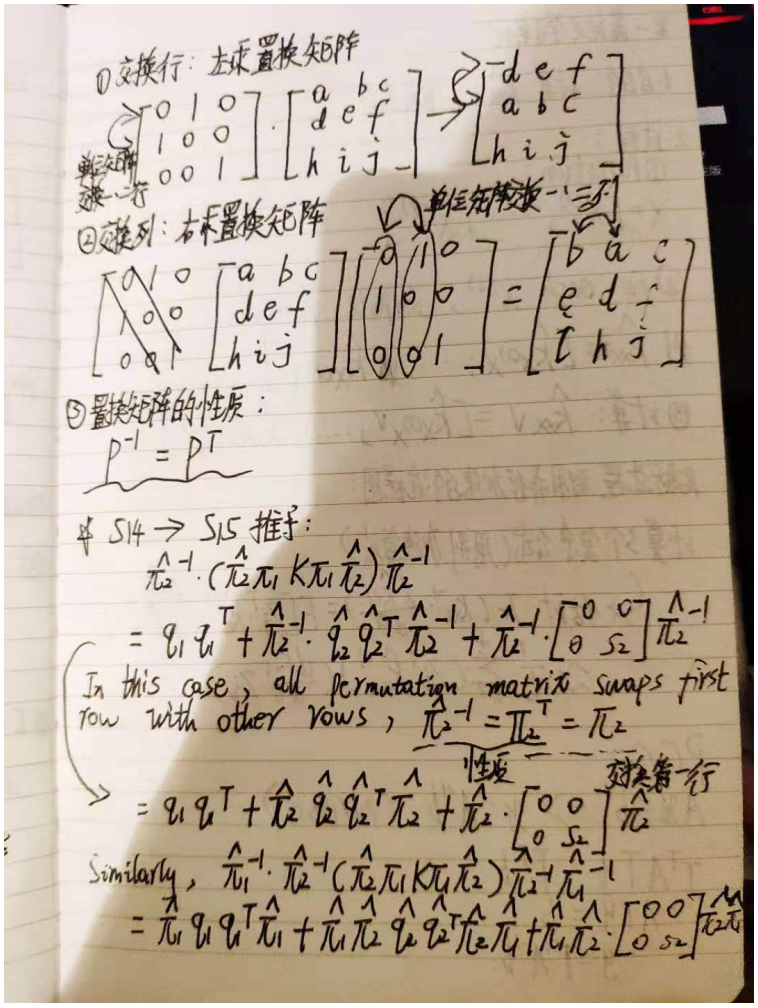


Figure 6: Derivation of PCD