Preconditioning Kernel Matrices


Kurt Cutajar[1]


John P. Cunningham[2]


Michael A. Osborne[3]


Maurizio Filippone[1]

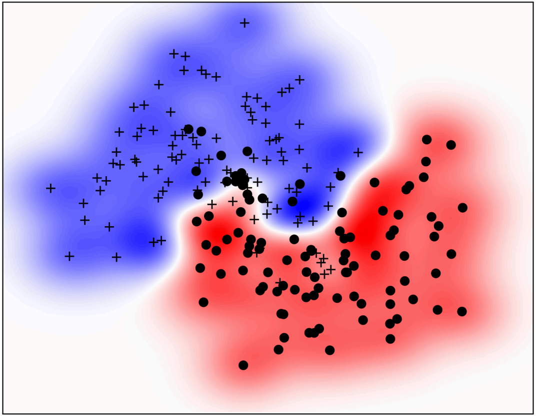
[1] EURECOM, Sophia Antipolis, France
[2] Columbia University, New York, US
[3] University of Oxford, Oxford, UK

ICML 2016, New York
June 22nd, 2016

Kernel Machines

- Operate in a high-dimensional, implicit feature space
- Rely on the construction of an $n \times n$ Gram matrix K
- E.g. RBF : $k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \exp\left(-\frac{1}{2} d^2\right)$

  where $d^2 = (\mathbf{x}_i - \mathbf{x}_j)^\top \Lambda (\mathbf{x}_i - \mathbf{x}_j)$

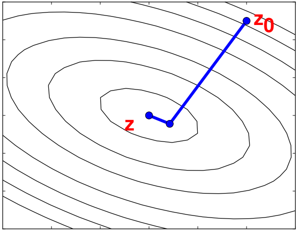


Solving Linear Systems

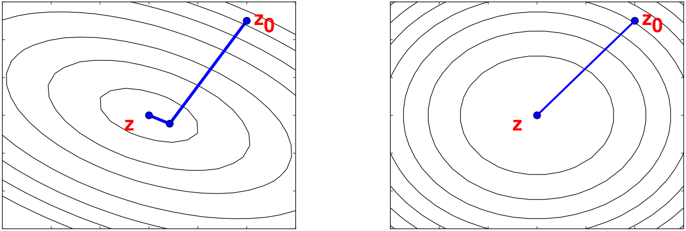
- Involve the solution of linear systems $K\mathbf{z} = \mathbf{v}$
- **Cholesky Decomposition**
  - K must be stored in memory!
  - $\mathcal{O}(n^2)$ space and $\mathcal{O}(n^3)$ time - unfeasible for large n

Solving Linear Systems

- Involve the solution of linear systems $K\mathbf{z} = \mathbf{v}$
- **Cholesky Decomposition**
  - K must be stored in memory!
  - $\mathcal{O}(n^2)$ space and $\mathcal{O}(n^3)$ time - unfeasible for large n
- **Conjugate Gradient**
  - Numerical solution of linear systems
  - $\mathcal{O}(tn^2)$ for t CG iterations - in theory $t = n$ (possibly worse!)



- **Preconditioned Conjugate Gradient** (henceforth **PCG**)
- Transforms linear system to be better conditioned, improving convergence
- Yields a new linear system of the form $P^{-1}K\mathbf{z} = P^{-1}\mathbf{v}$
- $\mathcal{O}(tn^2)$ for t PCG iterations - in practice $t \ll n$

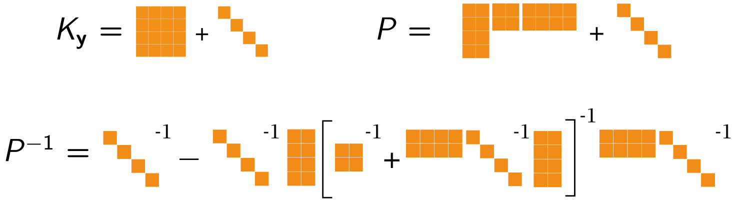


CG PCG

Preconditioning Approaches

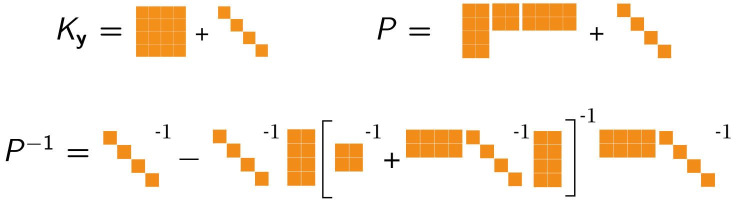
- Suppose we want to precondition $K_{\mathbf{y}} = K + \lambda I$
- Our choice of preconditioner, P, should:
  - Approximate $K_{\mathbf{y}}$ as closely as possible
  - Be easy to invert

- Suppose we want to precondition $K_{\mathbf{y}} = K + \lambda I$
- Our choice of preconditioner, P, should:
  - Approximate $K_{\mathbf{y}}$ as closely as possible
  - Be easy to invert
- For low-rank preconditioners we employ the Woodbury inversion lemma:



Preconditioning Approaches

- Suppose we want to precondition $K_{\mathbf{y}} = K + \lambda I$
- Our choice of preconditioner, P, should:
  - Approximate $K_{\mathbf{y}}$ as closely as possible
  - Be easy to invert
- For low-rank preconditioners we employ the Woodbury inversion lemma:

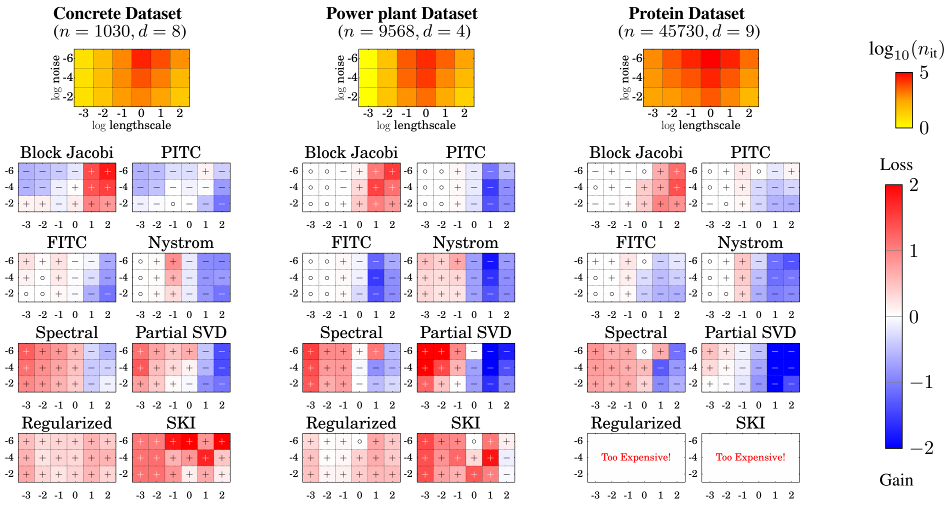


- For other preconditioners we solve inner linear systems once again using CG!

Preconditioning Approaches

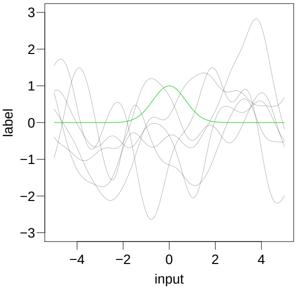
Nyström	$P = K_{XU} K_{UU}^{-1} K_{UX} + \lambda I \qquad \text{where } U \subset X$
FITC	$P = K_{XU} K_{UU}^{-1} K_{UX} + \text{diag}\left(K - K_{XU} K_{UU}^{-1} K_{UX}\right) + \lambda I$
PITC	$P = K_{XU} K_{UU}^{-1} K_{UX} + \text{bldiag}\left(K - K_{XU} K_{UU}^{-1} K_{UX}\right) + \lambda I$
Spectral	$P_{ij} = \frac{\sigma^2}{m} \sum_{r=1}^{m} \cos\left[2\pi \mathbf{s}_r^\top (\mathbf{x}_i - \mathbf{x}_j)\right] + \lambda I_{ij}$
Partial SVD	$K = A \Lambda A^\top \quad \Rightarrow \quad P = A_{[\cdot,1:m]} \Lambda_{[1:m,1:m]} A_{[1:m,\cdot]}^\top + \lambda I$
Block Jacobi	$P = \text{bldiag}(K) + \lambda I$
SKI	$P = W K_{UU} W^\top + \lambda I \qquad \text{where } K_{UU} \text{ is Kronecker}$
Regularization	$P = K + \lambda I + \delta I$

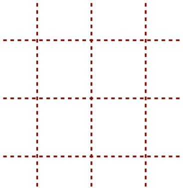
Comparison of Preconditioners vs CG



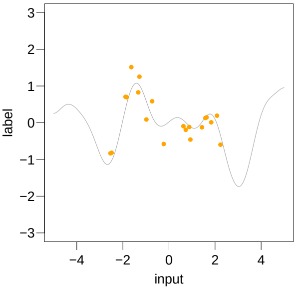
Motivating Example - Gaussian Processes

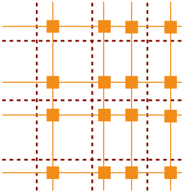
GP prior



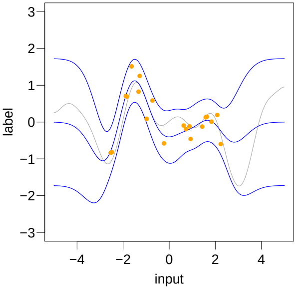
$K_\infty =$ 

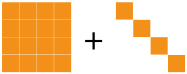
GP regression example



$K_\infty =$ 

Inference result



$K_{\mathbf{y}} =$ 

- Marginal likelihood

$$\log[p(\mathbf{y}|\mathrm{par})] = -\frac{1}{2}\log|K_{\mathbf{y}}| - \frac{1}{2}\mathbf{y}^{\mathrm{T}} K_{\mathbf{y}}^{-1}\mathbf{y} + \mathrm{const.}$$

- Derivatives wrt par

$$\frac{\partial \log[p(\mathbf{y}|\mathrm{par})]}{\partial \mathrm{par}_i} = -\frac{1}{2}\mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\right) + \frac{1}{2}\mathbf{y}^{\mathrm{T}} K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i} K_{\mathbf{y}}^{-1}\mathbf{y}$$

- Stochastic estimate of the trace - assuming $\mathrm{E}[\mathbf{r}\mathbf{r}^{\mathrm{T}}] = I$, then

$$\mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\right) = \mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathrm{E}[\mathbf{r}\mathbf{r}^{\mathrm{T}}]\right) = \mathrm{E}\left[\mathbf{r}^{\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathbf{r}\right]$$

- Stochastic gradient

$$-\frac{1}{2N_{\mathbf{r}}}\sum_{i=1}^{N_{\mathbf{r}}}\mathbf{r}^{(i)\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathbf{r}^{(i)} + \frac{1}{2}\mathbf{y}^{\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}K_{\mathbf{y}}^{-1}\mathbf{y}$$

- Stochastic estimate of the trace - assuming $\mathrm{E}[\mathbf{r}\mathbf{r}^{\mathrm{T}}] = I$, then

$$\mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\right) = \mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathrm{E}[\mathbf{r}\mathbf{r}^{\mathrm{T}}]\right) = \mathrm{E}\left[\mathbf{r}^{\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathbf{r}\right]$$

- Stochastic gradient

$$-\frac{1}{2N_{\mathbf{r}}}\sum_{i=1}^{N_{\mathbf{r}}}\mathbf{r}^{(i)\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathbf{r}^{(i)} + \frac{1}{2}\mathbf{y}^{\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}K_{\mathbf{y}}^{-1}\mathbf{y}$$

Linear systems only!

Gaussian Processes - Stochastic Gradients

- Stochastic estimate of the trace - assuming $\mathrm{E}[\mathbf{r}\mathbf{r}^{\mathrm{T}}] = I$, then

$$\mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\right) = \mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathrm{E}[\mathbf{r}\mathbf{r}^{\mathrm{T}}]\right) = \mathrm{E}\left[\mathbf{r}^{\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathbf{r}\right]$$

- Stochastic gradient

$$-\frac{1}{2N_{\mathbf{r}}}\sum_{i=1}^{N_{\mathbf{r}}}\mathbf{r}^{(i)\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}\mathbf{r}^{(i)} + \frac{1}{2}\mathbf{y}^{\mathrm{T}}K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \mathrm{par}_i}K_{\mathbf{y}}^{-1}\mathbf{y}$$

Linear systems only!

Also applicable to non-Gaussian likelihoods!

Experimental Setup

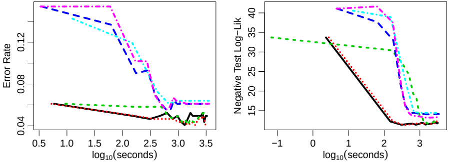
GP Kernel Parameter Optimization

- Exact gradient-based optimization using **Cholesky** decomposition (CHOL)
- Stochastic gradient-based optimization (using ADAGRAD)
  - Linear systems solved with **CG** and **PCG**
- GP Approximations
  - Variational learning of inducing variables (**VAR**)
  - Fully Independent Training Conditional (**FITC**)
  - Partially Independent Training Conditional (**PITC**)

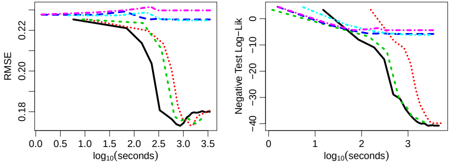
Classification

Regression

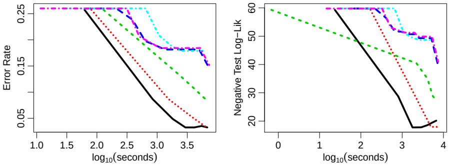
Spam ($n = 4061$, d=57)



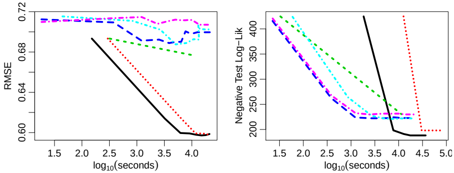
Power plant ($n = 9568$, d=4)

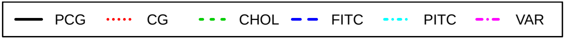


EEG ($n = 14979$, d=14)



Protein ($n = 45730$, d=9)





Conclusions

- Contributions
  - We provide an extensive discussion of preconditioning
  - We incorporate preconditioning in GP models with both Gaussian and non-Gaussian likelihoods
  - We carry out exact GP inference at a fraction of the cost

Conclusions

- Contributions
  - We provide an extensive discussion of preconditioning
  - We incorporate preconditioning in GP models with both Gaussian and non-Gaussian likelihoods
  - We carry out exact GP inference at a fraction of the cost
- Ongoing work
  - Extending this work to other kernel functions and models
  - Implementation on a distributed framework
  - Exploiting PCG in the solution of $f(K)\mathbf{z} = \mathbf{v}$

Conclusions

- Contributions
  - We provide an extensive discussion of preconditioning
  - We incorporate preconditioning in GP models with both Gaussian and non-Gaussian likelihoods
  - We carry out exact GP inference at a fraction of the cost
- Ongoing work
  - Extending this work to other kernel functions and models
  - Implementation on a distributed framework
  - Exploiting PCG in the solution of $f(K)\,\mathbf{z} = \mathbf{v}$

Thank you!