MLSS 2019 London

Kernels
Part III: Large scale

Lorenzo Rosasco
MaLGa- Machine learning Genova Center
Universit’a di Genova
MIT
IIT

Outline

Large scale

Random features

Nyström

Optimization

Conclusion

Learning with kernels

$$\widehat{f}_\lambda(x) = \sum_{i=1}^{n} k(x, x_i) c_i, \qquad c = (\widehat{K} + \lambda n I)^{-1} \widehat{y}$$

$$(\widehat{K})_{ij} = k(x_i, x_j)$$

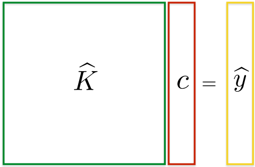
Requires[1]: time $O(c_k n^2 + n^3)$ space $O(n^2)$.

[1] c_k cost of evaluating k

Bottlenecks for kernels

$$\widehat{f}_{\lambda}(x) = \sum_{i=1}^{n} k(x, x_i) c_i$$

$$(\widehat{K} + \lambda n I) c = \widehat{y}$$



Computations: **Space** $O(n^2)$ **Kernel eval.** $O(n^2)$ **Time** $O(n^3)$

Kernel space//time

Computations: **Space** $O(n^2)$ **Kernel eval.** $O(n^2)$ **Time** $O(n^3)$

- Kernel methods require manipulating $\widehat{K}$.
- Memory is the main bottleneck.
- On the fly kernel evaluation helps but does not solve the problem.

Outline

Large scale

Random features

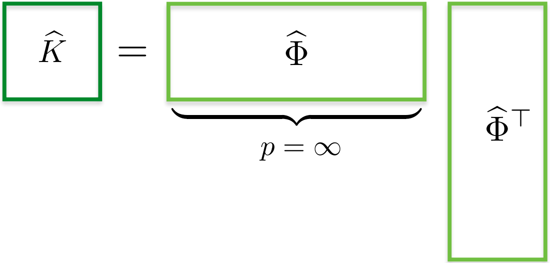
Nyström

Optimization

Conclusion

Feature maps

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^\top \Phi(\mathbf{x}')$$



$$\widehat{K} = \underbrace{\widehat{\Phi}}_{p = \infty} \widehat{\Phi}^\top$$

$$\mathrm{p} \gg \mathrm{n}$$

LS with features

Let $\widehat{\Phi} \in \mathbb{R}^{np}$, $p \leqslant \infty$ Then,

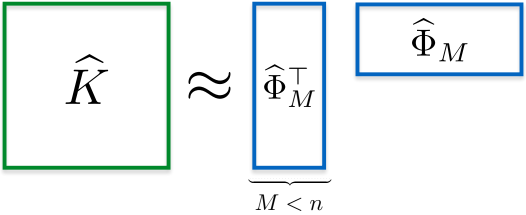
$$\widehat{w}_{\lambda} = (\widehat{\Phi}^{\top}\widehat{\Phi} + \lambda n I)^{-1}\widehat{\Phi}^{\top}\widehat{y}$$

Requires: time $O(np^2 + p^3)$ space $O(no \vee p^2)$.

$p = \infty \ldots.$

Approximate feature maps

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') \approx \Phi_{\mathrm{M}}(\mathbf{x})^\top \Phi_{\mathrm{M}}(\mathbf{x}')$$



$$\mathrm{M} \ll \mathrm{n}$$

Least squares with features

Let $\widehat{\Phi}_M \in \mathbb{R}^{nM}$ with $(\widehat{\Phi}_M)_{ij} = \Phi_M(\mathbf{x}_i)^j$

Then,

$$\widehat{w}_\lambda = (\widehat{\Phi}_M^\top \widehat{\Phi}_M + \lambda n I)^{-1} \widehat{\Phi}_M^\top \widehat{\mathbf{y}}$$

Requires: time $O(nM^2 + M^3)$ space $O(M^2)$.

Example: linear sketching

Let S be a d × M matrix and

$$\widehat{\Phi}_M = \widehat{X}S$$

Equivalenty

$$\mathbf{x} \in \mathbb{R}^d \quad \mapsto \quad \Phi_M(\mathbf{x}) = (\mathbf{s}_j^\top \mathbf{x})_{j=1}^M \in \mathbb{R}^M$$

with $s_1, \dots, s_M$ columns of S.

Random linear sketching

Let $s_j \sim \mathcal{N}(0, I)$ iid, then

$$\mathbf{x}^\top \mathbf{x}' = \frac{1}{M}\mathbb{E}[\Phi_M(\mathbf{x})^\top \Phi_M(\mathbf{x}')].$$

Random linear sketching

Let $s_j \sim \mathcal{N}(0, I)$ iid, then

$$x^\top x' = \frac{1}{M}\mathbb{E}[\Phi_M(x)^\top \Phi_M(x')].$$

Proof

$$\frac{1}{M}\mathbb{E}[\Phi_M(x)^\top \Phi_M(x')] = \frac{1}{M}\mathbb{E}[\sum_{j=1}^{M} x^\top s_j s_j^\top x'] = \frac{1}{M}\sum_{j=1}^{M} x^\top \underbrace{\mathbb{E}[s_j s_j^\top]}_{\text{Identity}} x' = x^\top x'.$$

Random linear sketching

Let $s_j \sim \mathcal{N}(0, I)$ iid, then

$$\mathbf{x}^\top \mathbf{x}' = \frac{1}{M}\mathbb{E}[\Phi_M(\mathbf{x})^\top \Phi_M(\mathbf{x}')].$$

Proof

$$\frac{1}{M}\mathbb{E}[\Phi_M(\mathbf{x})^\top \Phi_M(\mathbf{x}')] = \frac{1}{M}\mathbb{E}[\sum_{j=1}^{M} \mathbf{x}^\top s_j s_j^\top \mathbf{x}'] = \frac{1}{M}\sum_{j=1}^{M} \mathbf{x}^\top \underbrace{\mathbb{E}[s_j s_j^\top]}_{\text{Identity}} \mathbf{x}' = \mathbf{x}^\top \mathbf{x}'.$$

Note:

- Related to Johnson-Linderstrauss Lemma…

Linear random features

$$\mathbf{x}^\top \mathbf{x}' = \mathbb{E}[(\mathbf{x}^\top \mathbf{s})(\mathbf{s}^\top \mathbf{x}')]$$

Linear random features

$$\mathbf{x}^\top \mathbf{x}' = \mathbb{E}[(\mathbf{x}^\top \mathbf{s})(\mathbf{s}^\top \mathbf{x}')]$$

$$\Downarrow$$

$$\mathbf{x}^\top \mathbf{x}' \approx \frac{1}{M} \sum_{j=1}^{M} \mathbf{x}^\top \mathbf{s}_j \mathbf{s}_j^\top \mathbf{x}'$$

Linear random features

$$\mathbf{x}^\top \mathbf{x}' = \mathbb{E}[(\mathbf{x}^\top \mathsf{s})(\mathsf{s}^\top \mathbf{x}')]$$

$$\Downarrow$$

$$\mathbf{x}^\top \mathbf{x}' \approx \frac{1}{M} \sum_{j=1}^{M} \mathbf{x}^\top \mathsf{s}_j \mathsf{s}_j^\top \mathbf{x}' = \frac{1}{M} \Phi_M(\mathbf{x})^\top \Phi_M(\mathbf{x}')$$

with $\Phi_M(\mathbf{x}) = (\mathsf{s}_j^\top \mathbf{x})_{j=1}^{M} \in \mathbb{R}^M$

Random features

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[\psi(\mathbf{x}, \mathbf{s})\psi(\mathbf{x}', \mathbf{s})]$$

Random features

$$k(x, x') = \mathbb{E}[\psi(x, s)\psi(x', s)]$$

$$\Downarrow$$

$$k(x, x') \approx \frac{1}{M}\sum_{j=1}^{M} \psi(x, s_j)\psi(x', s_j)$$

Random features

$$k(x, x') = \mathbb{E}[\psi(x, s)\psi(x', s)]$$

$$\Downarrow$$

$$k(x, x') \approx \frac{1}{M}\sum_{j=1}^{M}\psi(x, s_j)\psi(x', s_j) = \frac{1}{M}\Phi_M(x)^\top\Phi_M(x')$$

with $\Phi_M(x) = (\psi(x, s_1), \psi(x, s_2), \ldots, \psi(x, s_M))$.

Random Fourier features

Let $X = \mathbb{R}$, $s_j \sim \mathcal{N}(0, 1)$ iid and

$$\psi(x, s_j) = \underbrace{e^{i s_j x}}_{\text{complex exp.}} .$$

Random Fourier features

Let $X = \mathbb{R}$, $s_j \sim \mathcal{N}(0,1)$ iid and

$$\psi(x, s_j) = \underbrace{e^{is_jx}}_{\text{complex exp.}} .$$

For $k(x,x') = e^{-|x-x'|^2\gamma}$, then

$$k(x,x') = \mathbb{E}[\psi(x,s)\psi(x',s)]$$

Random Fourier features

Let $X = \mathbb{R}$, $s_j \sim \mathcal{N}(0,1)$ iid and

$$\psi(x, s_j) = \underbrace{e^{is_jx}}_{\text{complex exp.}} .$$

For $k(x, x') = e^{-|x-x'|^2\gamma}$, then

$$k(x, x') = \mathbb{E}[\psi(x, s)\psi(x', s)]$$

Proof: from basic properties of the Fourier transform

$$e^{-|x-x'|^2\gamma} = \text{const.} \int ds \underbrace{e^{isx}}_{\text{Inv. transf. -}} \underbrace{e^{-isx'}}_{\text{Transl.}} \underbrace{e^{\frac{s^2}{\gamma}}}_{\text{- Tranf. of Gaussian}} .$$

Random Fourier features (cont.)

- The above reasoning immediately extends to $X = \mathbb{R}^d$.
- Using symmetry one can show the same result holds for

$$\psi(\mathbf{x}, (\beta_j, \mathbf{b}_j)) = \cos(\beta_j^\top \mathbf{x} + \mathbf{b}_j)$$

with b_j uniformly distributed.

Random ReLU features

Let $(\beta_j, b_j) \sim U[\mathbb{S}^{d+1}]$

$$\psi(\mathbf{x}, (\beta_j, b_j)) = |\beta_j \mathbf{x} + b_j|_+$$

If

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') = \sin\theta + (\pi - \theta)\cos\theta, \qquad \theta = \mathrm{arcos}(\mathbf{x}^\top \mathbf{x}')$$

then

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[\psi(\mathbf{x}, \mathbf{s})\psi(\mathbf{x}', \mathbf{s})]$$

Kernels and random features

- Pick a kernel and derive a random feature map, or . . .

Kernels and random features

- Pick a kernel and derive a random feature map, or …
- …pick random features and derive the limit kernel.

Kernels and random features

- Pick a kernel and derive a random feature map, or . . .
- . . . pick random features and derive the limit kernel.

Useful perspective for neural nets?

Random features and neural nets

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \sigma(\beta_j^\top \mathbf{x} + b_j)$$

- $\sigma : \mathbb{R} \to \mathbb{R}$ a non linear activation function.
- For $j = 1, \dots, M$, β_j, w_j, b_j parameters to be determined.

Random features and neural nets

$$f(x) = \sum_{j=1}^{M} w_j \sigma(\beta_j^\top x + b_j)$$

- $\sigma : \mathbb{R} \to \mathbb{R}$ a non linear activation function.
- For $j = 1, \dots, M$, β_j, w_j, b_j parameters to be determined.

Some references

- **History** [McCulloch, Pitts '43; Rosenblatt '58; Minsky, Papert '69; Y. LeCun, '85; Hinton et al. '06]
- **Deep learning** [Krizhevsky et al. '12 - 18705 Cit.!!!]
- **Theory** [Barron '92-94; Bartlett, Anthony '99; Pinkus, '99]

Learning with random features

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \sigma({\beta_j}^\top \mathbf{x} + b_j)$$

- $\sigma : \mathbb{R} \to \mathbb{R}$ a non linear activation (?) function.
- For $j = 1, \ldots, M$, w_j parameters to be determined
- For $j = 1, \ldots, M$, β_j, b_j chosen at random

Learning with random features

$$f(x) = \sum_{j=1}^{M} w_j \sigma({\beta_j}^\top x + b_j)$$

- $\sigma : \mathbb{R} \to \mathbb{R}$ a non linear activation (?) function.
- For $j = 1, \ldots, M$, w_j parameters to be determined
- For $j = 1, \ldots, M$, β_j, b_j chosen at random

Some references

- **Neural nets** [Block '62], **Extreme learning machine** [Huang et al. '06] 5196 Cit.??
- **Gaussian processes/kernel methods** [Neal '95, Rahimi, Recht '06'08'08, Acot et al. '18 (Neural tangent kernel)]

Mean field neural nets model

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \sigma({\beta_j}^\top \mathbf{x} + b_j)$$

Mean field neural nets model

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \sigma({\beta_j}^\top \mathbf{x} + b_j)$$

Infinitely wide neural nets define RKHS

$$f(\mathbf{x}) = \int d\pi(\beta, b) w(\beta, b) \sigma(\beta^\top \mathbf{x} + b)$$

Mean field neural nets model

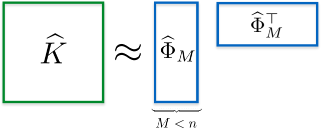
$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \sigma(\beta_j{}^\top \mathbf{x} + b_j)$$

Infinitely wide neural nets define RKHS

$$f(\mathbf{x}) = \int d\pi(\beta, b) w(\beta, b) \sigma(\beta^\top \mathbf{x} + b) = \mathbb{E}[w(\beta, b)\sigma(\beta^\top \mathbf{x} + b)].$$

...

Least squares and random features



$$\widehat{K} \approx \underbrace{\widehat{\Phi}_M}_{M<n} \widehat{\Phi}_M^\top$$

$\widehat{w}_\lambda = (\widehat{\Phi}_M^\top \widehat{\Phi}_M + \lambda n I)^{-1} \widehat{\Phi}_M^\top \widehat{y}$ Requires: time $O(nM^2 + M^3)$ space $O(nM)$.

Even better: SGD

$w_{t+1} = w_t - \gamma_t \Phi_M(x_t)(w_t^\top \Phi_M(x_t) - y_t)$ Requires: time $O(nMt)$ space $O(M)$.

How many random features?

$$\boxed{\widehat{K}} \approx \underbrace{\boxed{\widehat{\Phi}_M}}_{M<n} \boxed{\widehat{\Phi}_M^\top}$$

kernel approximation

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') \approx \Phi_{\mathrm{M}}(\mathbf{x})^\top \Phi_{\mathrm{M}}(\mathbf{x}')$$

vs

learning

$$\mathbb{E}[\ell(\mathbf{y}, \widehat{\mathrm{w}}_\lambda \Phi_{\mathrm{M}}(\mathbf{x}))]$$

The number of RF needed for learning can be much smaller than n!

But…

The number of RF needed for learning can be much smaller than n!

But…there’s no time today!

What about data dependent approximations?

Outline

Large scale

Random features

Nyström

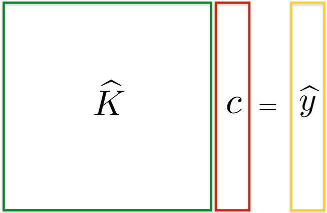
Optimization

Conclusion

Representer theorem

$$\widehat{f}_\lambda(x) = \sum_{i=1}^{n} k(x, x_i) c_i$$

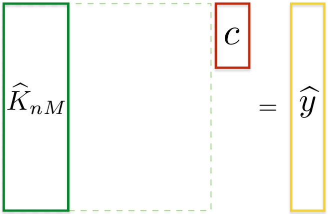
$$(\widehat{K} + \lambda n I)c = \widehat{y}$$



Nyström

$$\widehat{f}_{\lambda,M}(x) = \sum_{i=1}^{M} K(x,\widetilde{x}_i)c_i$$

$$(\widehat{K}_{nM}^{\top}\widehat{K}_{nM} + \lambda n\widehat{K}_{MM})c = \widehat{K}_{nM}^{\top}\widehat{y}$$

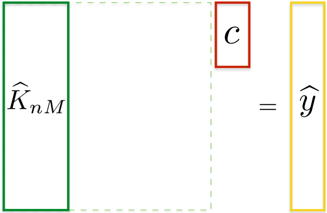


with $\widetilde{x}_1,\dots,\widetilde{x}_M \subset x_1,\dots x_n$ and corresponding columns chosen at random

Nyström

$$\widehat{f}_{\lambda,M}(x) = \sum_{i=1}^{M} K(x, \widetilde{x}_i) c_i$$

$$(\widehat{K}_{nM}^{\top} \widehat{K}_{nM} + \lambda n \widehat{K}_{MM}) c = \widehat{K}_{nM}^{\top} \widehat{y}$$



with $\widetilde{x}_1, \dots, \widetilde{x}_M \subset x_1, \dots x_n$ and corresponding columns chosen at random

Not throwing data! Just using a subset for modeling functions!

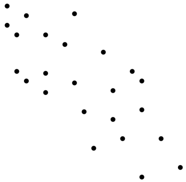
Why should it work?

Why should it work?

$$(\underbrace{\widehat{X}^\top \widehat{X}}_{\text{Cov.matrix}} + \lambda n)^{-1}$$

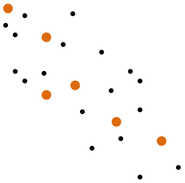
Why should it work?

$$(\underbrace{\widehat{X}^\top \widehat{X}}_{\text{Cov.matrix}} + \lambda n)^{-1}$$



Why should it work?

$$\Big(\underbrace{\widehat{X}^{\top}\widehat{X}}_{\text{Cov.matrix}} + \lambda n\Big)^{-1}$$



Why should it work?

$$\left(\underbrace{\widehat{X}^\top \widehat{X}}_{\text{Cov.matrix}} + \lambda n \right)^{-1}$$

Why should it work?

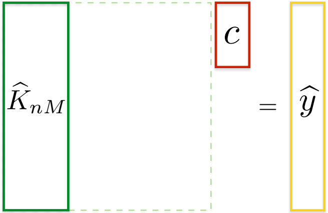
$$(\widehat{K} + \lambda n)^{-1}$$

We apply the same idea to the feature/kernel space!

Name game

$$\widehat{f}_{\lambda,M}(x) = \sum_{i=1}^{M} K(x, \widetilde{x}_i) c_i$$

$$(\widehat{K}_{nM}^{\top} \widehat{K}_{nM} + \lambda n \widehat{K}_{MM}) c = \widehat{K}_{nM}^{\top} \widehat{y}$$



Some references

- **Nyström methods** (Smola, Scholköpf '00)
- Gaussian processes: inducing inputs (Quiñonero-Candela et al '05)
- Randomized numerical linear algebra: column sampling (Halko et al. '11)

Why the name Nyström approximation?

Why the name Nyström approximation?

Discrete approximation of integral operators

For all x

$$\int k(x,x')c(x')dx' = y(x) \qquad \mapsto \qquad \sum_{j=1}^{M} k(x,\widetilde{x}_j)c(\widetilde{x}_j) = y(\widetilde{x}_j)$$

Related to to **quadrature** methods.

Why the name Nyström approximation?

Discrete approximation of integral operators

For all x

$$\int k(x,x')c(x')dx' = y(x) \qquad \mapsto \qquad \sum_{j=1}^{M} k(x,\widetilde{x}_j)c(\widetilde{x}_j) = y(\widetilde{x}_j)$$

Related to to **quadrature** methods.

From operators to (large)matrices

For all $i = 1, \ldots, n$

$$\sum_{j=1}^{n} k(x_i,x_j)c_j = y_j \qquad \mapsto \qquad \sum_{j=1}^{M} k(x_i,\widetilde{x}_j)c_i = y_j$$

Nyström approximation and subsampling

For all $i = 1, \dots, n$

$$\sum_{j=1}^{n} k(x_i, x_j) c_j = y_j \qquad \mapsto \qquad \sum_{j=1}^{M} k(x_i, \widetilde{x}_j) c_i = y_j$$

The above formulation highlights connection to columns subsampling

$$\widehat{K} c = \widehat{y} \qquad \mapsto \qquad \widehat{K}_{nM} c_M = \widehat{y}$$

Nyström as sketching

Consider the $d \times M$ matrix $S = (\widetilde{x}_1, \dots, \widetilde{x}_M)$ and

$$\widehat{\Phi}_M = \widehat{X} S$$

Equivalenty

$$\mathbf{x} \in \mathbb{R}^d \quad \mapsto \quad \Phi_M(\mathbf{x}) = (\widetilde{\mathbf{x}}_j^\top \mathbf{x})_{j=1}^M \in \mathbb{R}^M$$

with $s_1, \dots, s_M$ columns of S.

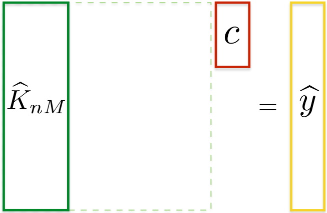
Nyström (for the linear kernels) is a form of data driven sketching.

Nyström as projection regularization

The cost of Nyström kernel ridge regression

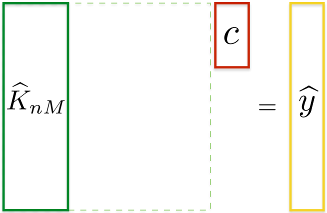
$$\widehat{f}_{\lambda,M}(x) = \sum_{i=1}^{M} K(x, \widetilde{x}_i) c_i$$

$$(\widehat{K}_{nM}^\top \widehat{K}_{nM} + \lambda n \widehat{K}_{MM}) c = \widehat{K}_{nM}^\top \widehat{y}$$



Requires: time $O(nM^2 + M^3)$ space $O(nM)$.

How many Nyström centers?



kernel approximation

$$\widehat{\mathrm{K}} \approx \widehat{\mathrm{K}}_{\mathrm{nM}}\, \widehat{\mathrm{K}}_{\mathrm{MM}}^{-1}\, \widehat{\mathrm{K}}_{\mathrm{nM}}^{\top}$$

vs

learning

$$\mathbb{E}[\ell(\mathbf{y}, \widehat{\mathbf{f}}_{\lambda,\mathrm{M}}(\mathbf{x}))]$$

The number of Nyström centers needed for learning can be much smaller than n!

But…

The number of Nyström centers needed for learning can be much smaller than n!

But. . . there’s no time today!

Can we do better?

Possible improvements

- **adaptive sampling (leverage scores)**

- optimization

Leverage scores and sampling

$$\ell(i, \lambda) = (\widehat{K}(\widehat{K} + \lambda n I)^{-1}))_{ii}$$

Sampling $J = \widetilde{x}_1, \dots, \widetilde{x}_M \subset x_1, \dots x_n$ according to $\ell(1, \lambda), \dots, \ell(n, \lambda)$.

- Can lead to smaller M than uniform sampling.
- Fast algorithms needed…

Approximate leverage scores

Basic idea: use only a subset of points to compute lev. scores.

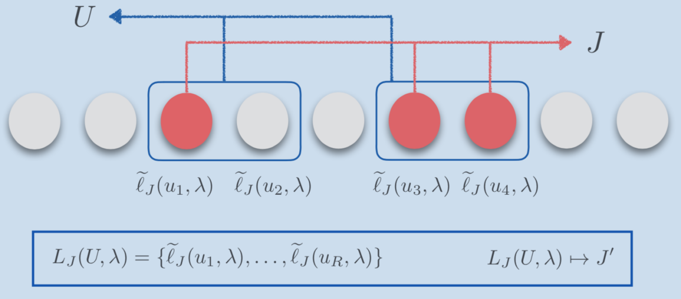
$$\widetilde{\ell}(i, \lambda) = \frac{1}{\lambda n}(\widehat{K}_{ii} - \widehat{K}_{J,i}^{\top}(\widehat{K}_{JJ} + \lambda n W)^{-1})\widehat{K}_{J,i})$$

Approximate leverage scores

Basic idea: use only a subset of points to compute lev. scores.

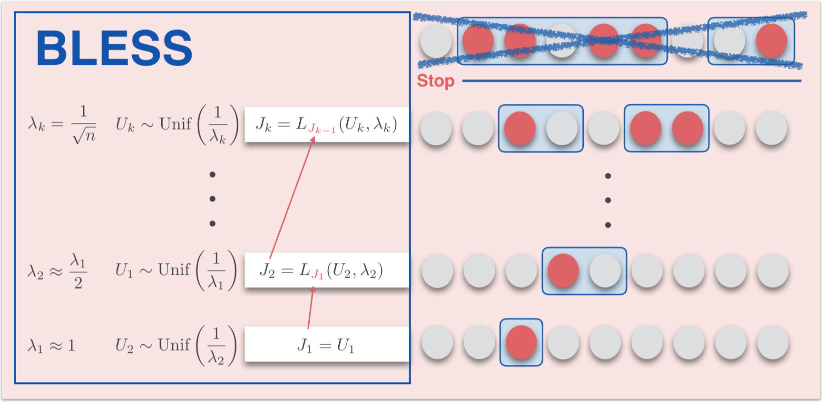
$$\widetilde{\ell}(i, \lambda) = \frac{1}{\lambda n}(\widehat{K}_{ii} - \widehat{K}_{J,i}^{\top}(\widehat{K}_{JJ} + \lambda n W)^{-1})\widehat{K}_{J,i})$$

Basic algorithm: uniform+lev. scores sampling.


U
J
$\widetilde{\ell}_J(u_1, \lambda)$
$\widetilde{\ell}_J(u_2, \lambda)$
$\widetilde{\ell}_J(u_3, \lambda)$
$\widetilde{\ell}_J(u_4, \lambda)$
$L_J(U, \lambda) = \{\widetilde{\ell}_J(u_1, \lambda), \ldots, \widetilde{\ell}_J(u_R, \lambda)\}$
$L_J(U, \lambda) \mapsto J'$

Fast leverage scores sampling

BLESS algorithm: coarse to fine uniform+lev. scores sampling.



BLESS

$\lambda_k = \frac{1}{\sqrt{n}}$ $\quad U_k \sim \text{Unif}\left(\frac{1}{\lambda_k}\right)$ $\quad J_k = L_{J_{k-1}}(U_k, \lambda_k)$

Stop

$\lambda_2 \approx \frac{\lambda_1}{2}$ $\quad U_1 \sim \text{Unif}\left(\frac{1}{\lambda_1}\right)$ $\quad J_2 = L_{J_1}(U_2, \lambda_2)$

$\lambda_1 \approx 1$ $\quad U_2 \sim \text{Unif}\left(\frac{1}{\lambda_2}\right)$ $\quad J_1 = U_1$

Possible improvements

- adaptive sampling (leverage scores)

- **optimization**

Outline

Large scale

Random features

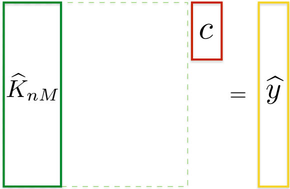
Nyström

Optimization

Conclusion

Beyond $\mathrm{O}(\mathrm{n}^2)$ time?

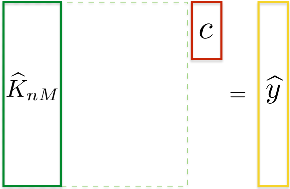
$$(\widehat{\mathrm{K}}_{\mathrm{nM}}^\top \widehat{\mathrm{K}}_{\mathrm{nM}} + \lambda \mathrm{n} \widehat{\mathrm{K}}_{\mathrm{MM}})\, \mathrm{c} \;=\; \widehat{\mathrm{K}}_{\mathrm{nM}}^\top \widehat{\mathrm{y}}.$$



Bottleneck: computing $\widehat{\mathrm{K}}_{\mathrm{nM}}^\top \widehat{\mathrm{K}}_{\mathrm{nM}}$ requires $\mathrm{O}(\mathrm{nM}^2)$ time.

Optimization to rescue

$$\underbrace{\widehat{K}_{nM}^{\top}\widehat{K}_{nM} + \lambda n \widehat{K}_{MM}}_{H}\, c = \underbrace{\widehat{K}_{nM}^{\top}\widehat{y}}_{b}\,.$$



Idea: First order methods

$$c_t = c_{t-1} - \frac{\tau}{n}\left[K_{nM}^{\top}(K_{nM}c_{t-1} - y_n)\ +\ \lambda n K_{MM}c_{t-1}\right]$$

Pros: requires $O(nMt)$

Cons: $t \propto \kappa(H)$ arbitrarily large- $\kappa(H) = \sigma_{\max}(H)/\sigma_{\min}(H)$ condition number.

Preconditioning

Idea: solve an equivalent linear system with better condition number

Preconditioning

$$Hc = b \quad \mapsto \quad P^\top H P \beta = P^\top b, \quad c = P\beta.$$

Ideally $PP^\top = H^{-1}$, so that

$$t = O(\kappa(H)) \quad \mapsto \quad t = O(1)!$$

Computing a good preconditioning can be hard!

Remarks

- Preconditioning kernel ridge regression
  (Fasshauer et al ’12, Avron et al ’16, Cutajat ’16, Ma, Belkin ’17)

$$H = \widehat{K} + \lambda n I$$

Can we precondition Nystrom-KRR?

Preconditioning Nyström-KRR

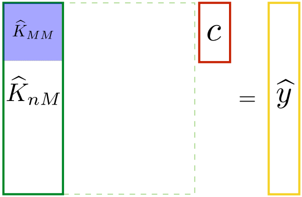
Consider $$H = \widehat{K}_{nM}^\top \widehat{K}_{nM} + \lambda n \widehat{K}_{MM}$$

Proposed Preconditioning

$$PP^\top = \left(\frac{n}{M} K_{MM}^2 + \lambda n K_{MM}\right)^{-1}$$

Compare to naive preconditioning

$$PP^\top = \left(\widehat{K}_{nM}^\top \widehat{K}_{nM} + \lambda n K_{MM}\right)^{-1}.$$



Baby FALKON

Proposed Preconditioning

$$PP^\top = \left(\frac{n}{M}K_{MM}^2 + \lambda n K_{MM}\right)^{-1},$$

Gradient descent

$$\widehat{f}_{\lambda,M,t}(x) = \sum_{i=1}^{M} K(x, \widetilde{x}_i) c_{t,i}, \qquad c_t = P\beta_t$$

$$\beta_t = \beta_{t-1} - \frac{\tau}{n} P^\top \left[K_{nM}^\top (K_{nM} P \beta_{t-1} - y_n) \; + \; \lambda n K_{MM} P \beta_{t-1}\right]$$

FALKON

- Gradient descent $\mapsto$ conjugate gradient
- Computing P

$$P = \frac{1}{\sqrt{n}} T^{-1} A^{-1}, \quad T = \mathrm{chol}(K_{MM}), \quad A = \mathrm{chol}\left(\frac{1}{M} T T^\top + \lambda I\right),$$

where $\mathrm{chol}(\cdot)$ is the Cholesky decomposition.

Relevant works

References

- Less is more (Rudi et al. ’15)
- Divide and conquer (Zhang et al. ’13)
- NYTRO (Angles et al ’16)
- Nyström SGD (Lin, R. ’17)
- EIGENPRO (Belkin al ’16)

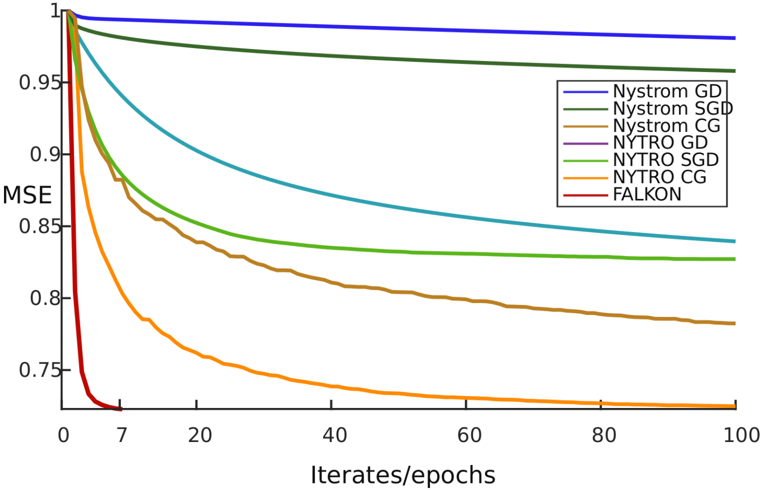
Computational costs for FALKON

$\log n$ iterations suffice leading to

$$\begin{aligned} \text{Space:} &\quad O(n) \\ \text{Kernel eval.:} &\quad O(nM) \\ \text{Time:} &\quad O(nM\log n) \\ \text{Model:} &\quad O(M) \end{aligned}$$

In practice

Higgs dataset: $\mathrm{n} = 10,000,000, \quad \mathrm{M} = 50,000$



Some experiments

	MillionSongs ($n \sim 10^6$)			YELP ($n \sim 10^6$)		TIMIT ($n \sim 10^6$)	
	MSE	Relative error	Time(s)	RMSE	Time(m)	c-err	Time(h)
FALKON	**80.30**	$\mathbf{4.51 \times 10^{-3}}$	**55**	**0.833**	**20**	32.3%	**1.5**
Prec. KRR	-	4.58×10^{-3}	$289^{\dagger}$	-	-	-	-
Hierarchical	-	4.56×10^{-3}	$293^{\star}$	-	-	-	-
D&C	80.35	-	737^{*}	-	-	-	-
Rand. Feat.	80.93	-	772^{*}	-	-	-	-
Nyström	80.38	-	876^{*}	-	-	-	-
ADMM R. F.	-	5.01×10^{-3}	$958^{\dagger}$	-	-	-	-
BCD R. F.	-	-	-	0.949	$42^{\ddagger}$	34.0%	$1.7^{\ddagger}$
BCD Nyström	-	-	-	0.861	$60^{\ddagger}$	33.7%	$1.7^{\ddagger}$
KRR	-	4.55×10^{-3}	-	0.854	$500^{\ddagger}$	33.5%	$8.3^{\ddagger}$
EigenPro	-	-	-	-	-	32.6%	$3.9^{\wr}$
Deep NN	-	-	-	-	-	32.4%	-
Sparse Kernels	-	-	-	-	-	**30.9%**	-
Ensemble	-	-	-	-	-	33.5%	-

Table: MillionSongs, YELP and TIMIT Datasets. Times obtained on: $\ddagger$ = cluster of 128 EC2 r3.2xlarge machines, $\dagger$ = cluster of 8 EC2 r3.8xlarge machines, $\wr$ = single machine with two Intel Xeon E5-2620, one Nvidia GTX Titan X GPU and 128GB of RAM, $\star$ = cluster with 512 GB of RAM and IBM POWER8 12-core processor, $*$ = unknown platform.

Some more experiments

	SUSY (n $\sim 10^6$)			HIGGS (n $\sim 10^7$)		IMAGENET (n $\sim 10^6$)	
	c-err	AUC	Time(m)	AUC	Time(h)	c-err	Time(h)
FALKON	**19.6%**	0.877	**4**	0.833	**3**	20.7%	**4**
EigenPro	19.8%	-	6$^{\wr}$	-	-	-	-
Hierarchical	20.1%	-	40†	-	-	-	-
Boosted Decision Tree	-	0.863	-	0.810	-	-	-
Neural Network	-	0.875	-	0.816	-	-	-
Deep Neural Network	-	**0.879**	4680‡	**0.885**	78‡	-	-
Inception-V4	-	-	-	-	-	**20.0%**	-

Table: Architectures: $\dagger$ cluster with IBM POWER8 12-core cpu, 512 GB RAM, $\wr$ single machine with two Intel Xeon E5-2620, one Nvidia GTX Titan X GPU, 128GB RAM, $\ddagger$ single machine.

Image classification

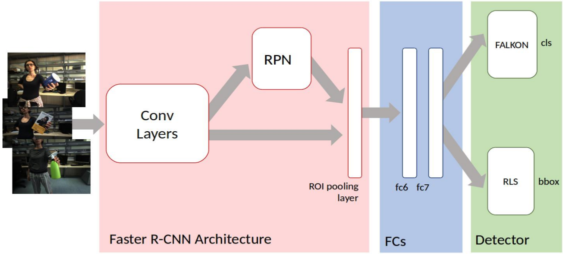
$$f(\mathbf{x}) = \langle w, \Phi(\mathbf{x}) \rangle , \quad \mathbf{x} \mapsto \underbrace{\Phi_L}_{\text{Kernel representation}} \circ \underbrace{\Phi_{L-1} \cdots \circ \Phi_1(\mathbf{x})}_{\text{Convolutional}}$$

Imagenet

	Top-1 class error
FALKON + I-v4 feat.	**20.7%**
Inception-v4	20.0%
Inception-v3	21.2%
Inception-v2	23.4%
BN-Inception	25.2%
BN-GoogLeNet	26.8%
GoogLeNet	29.0%

Table: Single crop experimental results on the validation set of ILSVRC 2012.

Real time object detection in robotics



RPN

Conv Layers

ROI pooling layer

fc6 fc7

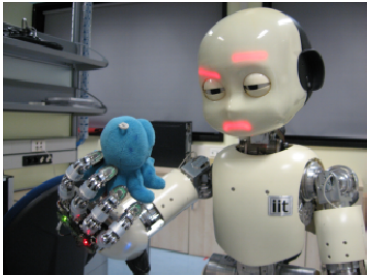
FALKON cls

RLS bbox

Faster R-CNN Architecture

FCs

Detector



Method	**mAP [%]**	**Train Time**
Faster R-CNN	51,9	∼25 min
FALKON + Full Bootstrap (∼ 1K×1000)	51,5	∼8 min
FALKON + Random BKG (0 × 7000)	47,7	∼25 sec

soap dispenser

hairbrush

sunglasses

cellphone

oven glove

glass

remote

perfume

squeezer

mouse

Take home message

You can train an SVM on millions of points in seconds/minutes

Outline

Large scale

Random features

Nyström

Optimization

Conclusion

Who cares about kernels?