Higher-order Factorization Machines

Mathieu Blondel

NTT Communication Science Laboratories
Kyoto, Japan

Joint work with M. Ishihata, A. Fujino and N. Ueda

2016/9/28

Regression analysis

- Variables

$$y \in \mathbb{R}: \text{ target variable}$$
$$\boldsymbol{x} \in \mathbb{R}^d: \text{ explanatory variables (features)}$$

- Training data

$$\boldsymbol{y} = [y_1, \ldots, y_n]^\mathrm{T} \in \mathbb{R}^n$$
$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$$

- Goal
  - Learn model parameters
  - Compute prediction y for a new $\boldsymbol{x}$

Linear regression

- Model

$$\hat{y}_{LR}(\boldsymbol{x}; \boldsymbol{w}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{j=1}^{d} w_j x_j$$

- Parameters

$$\boldsymbol{w} \in \mathbb{R}^d\text{: feature weights}$$

- Pros and cons
  - ☺ $O(d)$ predictions
  - ☺ Learning $\boldsymbol{w}$ can be cast as a convex optimization problem
  - ☹ Does not use feature interactions

Polynomial regression

- Model

$$\hat{y}_{PR}(\boldsymbol{x}; \boldsymbol{w}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \boldsymbol{x}^{\mathrm{T}} \boldsymbol{W} \boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j,j'=1}^{d} w_{j,j'} x_j x_{j'}$$

- Parameters

$$\boldsymbol{w} \in \mathbb{R}^d: \text{ feature weights}$$
$$\boldsymbol{W} \in \mathbb{R}^{d \times d}: \text{ weight matrix}$$

- Pros and cons

☺ Learning $\boldsymbol{w}$ and $\boldsymbol{W}$ can be cast as a convex optimization problem

☹ $O(d^2)$ time and memory cost

Kernel regression

- Model

$$\hat{y}_{KR}(\boldsymbol{x}; \boldsymbol{\alpha}) := \sum_{i=1}^{n} \alpha_i \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x})$$

- Parameters

$$\boldsymbol{\alpha} \in \mathbb{R}^n: \text{ instance weights}$$

- Pros and cons
  - ☺ Can use non-linear kernels (RBF, polynomial, etc...)
  - ☺ Learning $\boldsymbol{\alpha}$ can be cast as a convex optimization problem
  - ☹ $O(dn)$ predictions (linear dependence on training set size)

Factorization Machines (FMs) (Rendle, ICDM 2010)

- Model

jth row of $\boldsymbol{P}$

$$\hat{y}_{FM}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j' > j} \langle \bar{\boldsymbol{p}}_j, \bar{\boldsymbol{p}}_{j'} \rangle x_j x_{j'}$$

- Parameters

$$\boldsymbol{w} \in \mathbb{R}^d: \text{ feature weights}$$
$$\boldsymbol{P} \in \mathbb{R}^{d \times k}: \text{ weight matrix}$$

- Pros and cons

☺ Takes into account feature combinations

☺ $O(2dk)$ predictions (linear-time) instead of $O(d^2)$

☹ Parameter estimation involves a non-convex optimization problem

Application 1: recsys without features

- Formulate it as a matrix completion problem

	Movie 1	Movie 2	Movie 3	Movie 4
Alice	$\star\star$	?	$\star\star\star$	?
Bob	$\star$	?	$\star\star$	?
Charlie	$\star\star$	?	?	$\star\star$

- Matrix factorization: find $\boldsymbol{U}, \boldsymbol{V}$ that approximately reconstruct the rating matrix

$$\boldsymbol{R} \approx \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$$

Conversion to a regression problem

	Movie 1	Movie 2	Movie 3	Movie 4
Alice	⋆⋆	?	⋆⋆⋆	?
Bob	⋆	?	⋆⋆	?
Charlie	⋆⋆	?	?	⋆⋆

$\Downarrow$ one-hot encoding

$$\underbrace{\begin{bmatrix} \star\star \\ \star\star\star \\ \star \\ \star\star \\ \star\star \\ \star\star \end{bmatrix}}_{\boldsymbol{y}} \quad \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{X}}$$

Using this representation, FMs are equivalent to MF!

Generalization ability of FMs

- The weight of $x_j x_{j'}$ is $\langle \bar{\boldsymbol{p}}_j, \bar{\boldsymbol{p}}_{j'} \rangle$ compared to $w_{j,j'}$ for PR
- The same parameters $\bar{\boldsymbol{p}}_j$ are shared for the weight of $x_j x_{j'}$ $\forall j > j'$
- This increases the amount of data used to estimate $\bar{\boldsymbol{p}}_j$ at the cost of introducing some bias (low-rank assumption)
- This allows to generalize to feature interactions that were not observed in the training set

Application 2: recsys with features

Rating	User		Movie	
	Gender	Age	Genre	Director
⋆⋆	M	20-30	Adventure	S. Spielberg
⋆⋆⋆	F	0-10	Anime	H. Miyazaki
⋆	M	20-30	Drama	A. Kurosawa
⋮	⋮	⋮	⋮	⋮

- Interactions between **categorical variables**
  - Gender × genre: {M, F} × {Adventure, Anime, Drama, ...}
  - Age × director: {0-10, 10-20, ...} × {S. Spielberg, H. Miyazaki, A. Kurosawa, ...}
- In practice, the number of interactions can be **huge**!

Conversion to regression

Rating	User		Movie	
	Gender	Age	Genre	Director
⋆⋆	M	20-30	Adventure	S. Spielberg
⋆ ⋆ ⋆	F	0-10	Anime	H. Miyazaki
⋆	M	20-30	Drama	A. Kurosawa
⋮	⋮	⋮	⋮	⋮

$\Downarrow$ one-hot encoding

$$\underbrace{\begin{bmatrix} \star\star \\ \star\star\star \\ \star \\ \vdots \end{bmatrix}}_{\boldsymbol{y}} \quad \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \dots \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & \dots \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{bmatrix}}_{\boldsymbol{X}}$$

very sparse binary data!

FMs revisited (Blondel+, ICML 2016)

- ANOVA kernel of degree $m = 2$ (Stitson+, 1997; Vapnik, 1998)

$$\mathcal{A}^2(\boldsymbol{p}, \boldsymbol{x}) := \sum_{j'>j} p_j x_j \; p_{j'} x_{j'}$$

- Then

$$\begin{aligned} \hat{y}_{FM}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}) &= \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j'>j} \langle \bar{\boldsymbol{p}}_j, \bar{\boldsymbol{p}}_{j'} \rangle x_j x_{j'} \\ &= \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{s=1}^{k} \mathcal{A}^2(\boldsymbol{p}_s, \boldsymbol{x}) \end{aligned}$$

$\uparrow$ s$^{\text{th}}$ column of $\boldsymbol{P}$

ANOVA kernel (arbitrary-order case)

- ANOVA kernel of degree $2 \leq m \leq d$

$$\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) := \sum_{j_m > \cdots > j_1} (p_{j_1} x_{j_1}) \dots (p_{j_m} x_{j_m})$$

↑ All possible m-combinations of $\{1, \dots, d\}$

- Intuitively, the kernel uses all m-combinations of features without replacement: $x_{j_1} \dots x_{j_m}$ for $j_1 \neq \cdots \neq j_m$
- Computing $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ naively takes $O(d^m)$ ☹

Higher-order FMs (HOFMs)

- Model

$$\hat{y}_{HOFM}(\boldsymbol{x}; \boldsymbol{w}, \{\boldsymbol{P}^t\}_{t=2}^m) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{t=2}^{m} \sum_{s=1}^{k} \mathcal{A}^t(\boldsymbol{p}_s^t, \boldsymbol{x})$$

- Parameters

$$\boldsymbol{w} \in \mathbb{R}^d\text{: feature weights}$$
$$\boldsymbol{P}^2, \ldots, \boldsymbol{P}^m \in \mathbb{R}^{d \times k}\text{: weight matrices}$$

- Pros and cons
  - ☺ Takes into account higher-order feature combinations
  - ☺ $O(dkm^2)$ prediction cost using our proposed algorithms
  - ☹ More complex than 2nd-order FMs

Learning HOFMs (1/2)

- We use alternating mimimization w.r.t. $\boldsymbol{w}$, $\boldsymbol{P}^2$, ..., $\boldsymbol{P}^m$
- Learning $\boldsymbol{w}$ alone reduces to linear regression
- Learning $\boldsymbol{P}^m$ can be cast as minimizing

$$F(\boldsymbol{P}) := \frac{1}{n} \sum_{i=1}^{n} \ell \left(y_i, \sum_{s=1}^{k} \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i) + o_i \right) + \frac{\beta}{2} \|\boldsymbol{P}\|^2$$

where o_i is the contribution of degrees other than m

Learning HOFMs (2/2)

- Stochastic gradient update

$$\boldsymbol{p}_s \leftarrow \boldsymbol{p}_s - \eta \ell'(y_i, \hat{y}_i) \nabla \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i) - \eta \beta \boldsymbol{p}_s$$

  where η is a learning rate hyper-parameter and $\hat{y}_i := \sum_{s=1}^{k} \mathcal{A}^m(\boldsymbol{p}_s, \boldsymbol{x}_i) + o_i$

- We propose $O(dm)$ (linear time) DP algorithms for
  - Evaluating ANOVA kernel $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) \in \mathbb{R}$
  - Computing gradient $\nabla \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) \in \mathbb{R}^d$

Evaluating the ANOVA kernel (1/3)

- Recursion (Blondel+, ICML 2016)

$$\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = \mathcal{A}^m(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) + \; p_j x_j \; \mathcal{A}^{m-1}(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) \; \forall j$$

where $\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j} \in \mathbb{R}^{d-1}$ are vectors with the j^{th} element removed

- We can use this recursion to remove features until computing the kernel becomes trivial

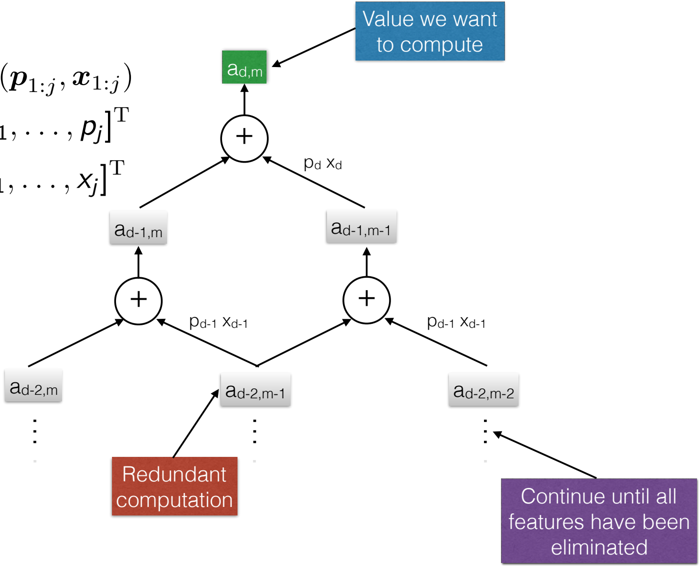
Evaluating the ANOVA kernel (2/3)

Shortcut:

$$a_{j,t} \coloneqq \mathcal{A}^t(\boldsymbol{p}_{1:j}, \boldsymbol{x}_{1:j})$$

$$\boldsymbol{p}_{1:j} \coloneqq [p_1, \ldots, p_j]^{\mathrm{T}}$$

$$\boldsymbol{x}_{1:j} \coloneqq [x_1, \ldots, x_j]^{\mathrm{T}}$$



Evaluating the ANOVA kernel (3/3)

Ways to avoid redundant computations:

- Top-down approach with memory table
- **Bottum-up dynamic programming (DP)**

	$j=0$	$j=1$	$j=2$	$\dots$	$j=d$	
$t=0$	1	1	1	1	1	
$t=1$	0	$a_{1,1}$	$a_{2,1}$	$\dots$	$a_{d,1}$	
$t=2$	0	start 0	$\xrightarrow{} a_{2,2}$ ($p_2 x_2$)	$\dots$	$a_{d,2}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
$t=m$	0	0	0	$\dots$	$a_{d,m}$	$\leftarrow$ goal

Algorithm 1 Evaluating $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ in $O(dm)$

Input: $\boldsymbol{p} \in \mathbb{R}^d, \boldsymbol{x} \in \mathbb{R}^d$

$a_{j,t} \leftarrow 0 \; \forall t \in \{1, \dots, m\}, j \in \{0, 1, \dots, d\}$

$a_{j,0} \leftarrow 1 \; \forall j \in \{0, 1, \dots, d\}$

for $t := 1, \dots, m$ **do**

  for $j := t, \dots, d$ **do**

    $a_{j,t} \leftarrow a_{j-1,t} + p_j x_j a_{j-1,t-1}$

  end for

end for

Output: $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = a_{d,m}$

Backpropagation (chain rule)

Ex: compute derivatives of composite function $f(g(h(\boldsymbol{p})))$

- Forward pass

$$
\begin{aligned}
a &= h(\boldsymbol{p}) \\
b &= g(a) \\
c &= f(b)
\end{aligned}
$$

- Backward pass

Only the last part depends on j

$$\frac{\partial c}{\partial p_j} = \frac{\partial c}{\partial b}\,\frac{\partial b}{\partial a}\,\frac{\partial a}{\partial p_j} = f'(b)\; g'(a)\; \overset{\downarrow}{h_j'(\boldsymbol{p})}$$

Can compute all derivatives in one pass!

Gradient computation (1/2)

- We want to compute $\nabla \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = [\tilde{p}_1, \ldots, \tilde{p}_d]^{\mathrm{T}}$
- Using the chain rule, we have

$$\tilde{p}_j := \frac{\partial a_{d,m}}{\partial p_j} = \sum_{t=1}^{m} \underbrace{\frac{\partial a_{d,m}}{\partial a_{j,t}}}_{:=\tilde{a}_{j,t}} \underbrace{\frac{\partial a_{j,t}}{\partial p_j}}_{=a_{j-1,t-1} x_j} = \sum_{t=1}^{m} \tilde{a}_{j,t} \; a_{j-1,t-1} \; x_j$$

  since p_j influences $a_{j,t}$ $\forall t \in [m]$

- $\tilde{a}_{j,t}$ can be computed recursively in reverse order

$$\tilde{a}_{j,t} = \tilde{a}_{j+1,t} + p_{j+1} x_{j+1} \; \tilde{a}_{j+1,t+1}$$

Gradient computation (2/2)

goal	$j=1$	$j=2$	$\dots$	$j=d-1$	$j=d$	
$t=1$	$\tilde{a}_{1,1}$	$\tilde{a}_{2,1}$	$\dots$	0	0	
$t=2$	0	$\tilde{a}_{2,2}$	$\ddots$	0	0	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\tilde{a}_{d-1,t-1} \leftarrow$	0	
$t=m$	0	0	0	1	$p_d x_d$ 1	$\leftarrow$ start
$t=m+1$	0	0	0	0	0	

Algorithm 2 Computing $\nabla \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ in $O(dm)$

Input: $\boldsymbol{p} \in \mathbb{R}^d$, $\boldsymbol{x} \in \mathbb{R}^d$, $\{a_{j,t}\}_{j,t=0}^{d,m}$
$\tilde{a}_{j,t} \leftarrow 0 \; \forall t \in [m+1], j \in [d]$
$\tilde{a}_{d,m} \leftarrow 1$
for $t := m, \ldots, 1$ **do**
  for $j := d-1, \ldots, t$ **do**
    $\tilde{a}_{j,t} \leftarrow \tilde{a}_{j+1,t} + \tilde{a}_{j+1,t+1} p_{j+1} x_{j+1}$
  end for
end for
$\tilde{p}_j := \sum_{t=1}^{m} \tilde{a}_{j,t} a_{j-1,t-1} x_j \; \forall j \in [d]$
Output: $\nabla \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x}) = [\tilde{p}_1, \ldots, \tilde{p}_d]^{\mathrm{T}}$

Summary so far

- HOFMs can be expressed using the ANOVA kernel $\mathcal{A}^m$
- We proposed $O(dm)$ time algorithms for computing $\mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$ and $\nabla \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$
- The cost per epoch of stochastic gradient algorithms for learning $\boldsymbol{P}^m$ is therefore $O(dnkm)$
- The prediction cost is $O(dkm^2)$

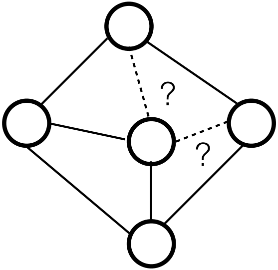
Other contributions

- Coordinate-descent algorithm for learning $\boldsymbol{P}^m$ based on a different recursion
  - Cost per epoch is $O(dnk\mathbf{m^2})$ ☹
  - However, no learning rate to tune! ☺
- HOFMs with shared parameters: $\boldsymbol{P}^2 = \cdots = \boldsymbol{P}^m$
  - Total prediction cost is $O(dkm)$ instead of $O(dkm^2)$ ☺
  - Corresponds to using new kernels derived from the ANOVA kernel

Experiments

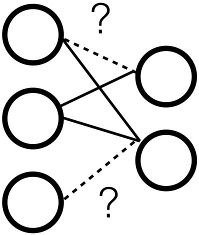
Application to link prediction

Goal: predict missing links between nodes in a graph



Graph:

- Co-author network
- Enzyme network



Bipartite graph:

- User-movie
- Gene-disease

Application to link prediction

- We assume two sets of nodes A (e.g., users) and B (e.g, movies) of size n_A and n_B
- Nodes in A are represented by feature vectors $\boldsymbol{a}_i \in \mathbb{R}^{d_A}$
- Nodes in B are represented by feature vectors $\boldsymbol{b}_j \in \mathbb{R}^{d_B}$
- We are given a matrix $\boldsymbol{Y} \in \{-1, +1\}^{n_A \times n_B}$ such that $y_{i,j} = +1$ if there is a link between $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$
- Number of positive samples is n_+

Datasets

Dataset	n_+	Columns of $\boldsymbol{A}$	n_A	d_A	Columns of $\boldsymbol{B}$	n_B	d_B
NIPS	4,140	Authors	2,037	13,649			
Enzyme	2,994	Enzymes	668	325			
GD	3,954	Diseases	3,209	3,209	Genes	12,331	25,275
ML 100K	21,201	Users	943	49	Movies	1,682	29

Features:

- NIPS: word occurence in author publications
- Enzyme: phylogenetic information, gene expression information and gene location information
- GD: MimMiner similarity scores (diseases) and HumanNet similarity scores (genes)
- ML 100K: age, gender, occupation, living area (users); release year, genre (movies)

Models compared

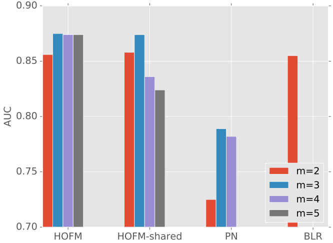
Goal: predict if there is a link between $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$

vector concatenation

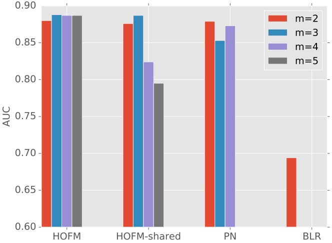
- HOFM: $\hat{y}_{i,j} = \hat{y}_{HOFM}(\boldsymbol{a}_i \oplus \boldsymbol{b}_j; \boldsymbol{w}, \{\boldsymbol{P}^t\}_{t=2}^m)$
- HOFM-shared: same but with $\boldsymbol{P}^2 = \cdots = \boldsymbol{P}^m$
- Polynomial network (PN): replace ANOVA kernel by polynomial kernel
- Bilinear regression (BLR): $\hat{y}_{i,j} = \boldsymbol{a}_i \boldsymbol{U} \boldsymbol{V}^{\mathrm{T}} \boldsymbol{b}_j$

Experimental protocol

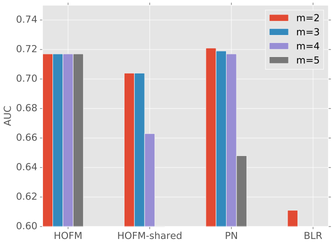
- We sample $n_{-} = n_{+}$ negatives samples (missing edges are treated as negative samples)
- We use 50% for training and 50% for testing
- We use ROC-AUC (area under ROC curve) for evaluation
- β tuned by CV, k fixed to 30
- $\boldsymbol{P}^2, \ldots, \boldsymbol{P}^m$ initialized randomly
- ℓ is set to the squared loss



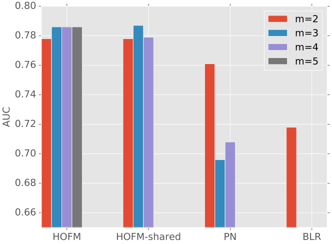
(a) NIPS



(b) Enzyme



(c) GD



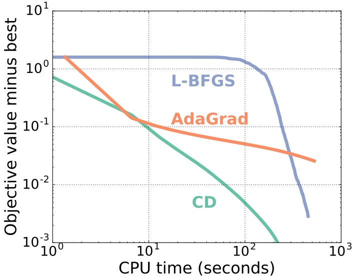
(d) ML100K

Solver comparison

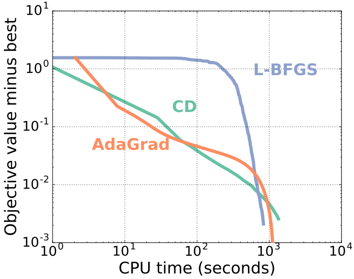
- Coordinate descent
- AdaGrad
- L-BFGS

AdaGrad and L-BFGS use the proposed DP algorithm to compute $\nabla \mathcal{A}^m(\boldsymbol{p}, \boldsymbol{x})$

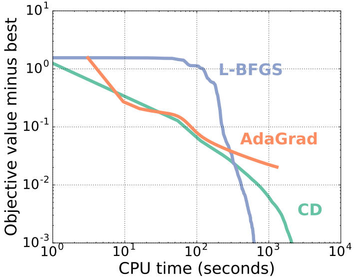
NIPS dataset



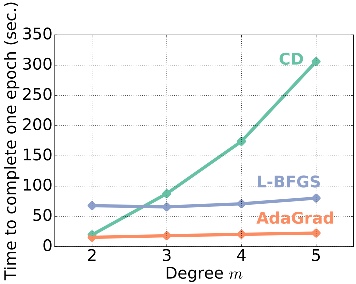
(a) Convergence when $m = 2$



(b) Convergence when $m = 3$



(c) Convergence when $m = 4$



(d) Scalability w.r.t. m