Recent advances on polynomial neural networks and factorization machines

Mathieu Blondel

NTT Communication Science Laboratories
Kyoto, Japan

2017/2/23

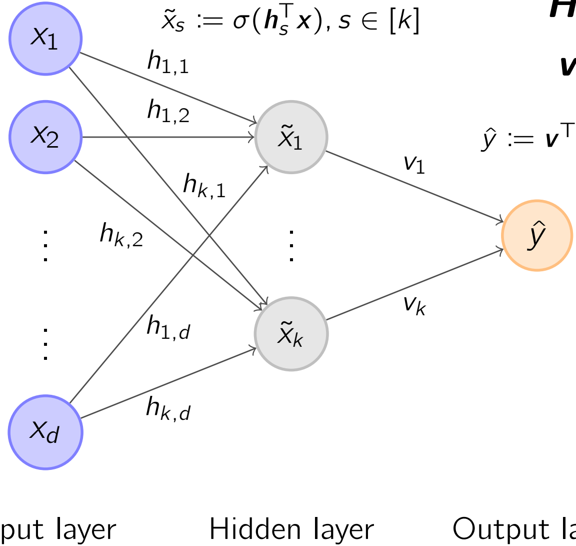
Neural networks

$$\tilde{x}_s := \sigma(\boldsymbol{h}_s^\top \boldsymbol{x}),\ s \in [k]$$

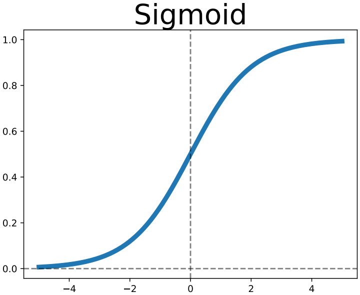
$$\boldsymbol{H} \in \mathbb{R}^{k \times d}$$

$$\boldsymbol{v} \in \mathbb{R}^{k}$$

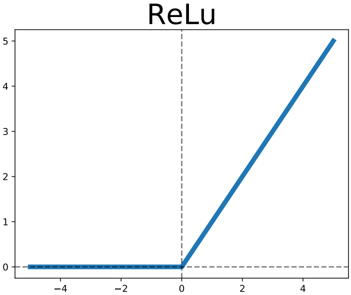
$$\hat{y} := \boldsymbol{v}^\top \tilde{\boldsymbol{x}}$$



Traditional neural networks

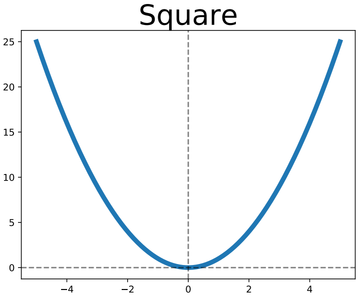


$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

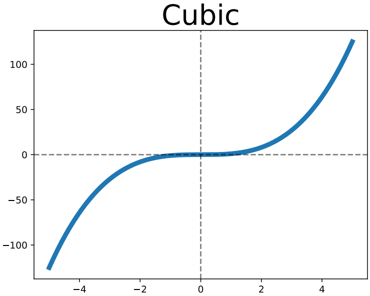


$$\sigma(u) = \max(u, 0)$$

Polynomial networks (Livni et al. 2014)



$$\sigma_2(u) := u^2$$



$$\sigma_3(u) := u^3$$

And more generally, $\sigma_m(u) := u^m$, for some degree m

Today's topics

$$\boxed{\hat{y}_{PN} := \sum_{s=1}^{k} v_s \; \sigma_m(\boldsymbol{h}_s^{\mathrm{T}} \boldsymbol{x})}$$

- Properties of polynomial networks
  - Ability to represent polynomials efficiently, universality
- How to train polynomial networks
  - Can we do better than just gradient descent?
- A very related model: factorization machines

Efficient representation of polynomials (1/2)

- A monomial of degree m is a function $f : \mathbb{R}^d \to \mathbb{R}$ s.t.

$$f(\boldsymbol{x}) = \prod_{t=1}^{m} x_{j_t} = x_{j_1} x_{j_2} \dots x_{j_m} \qquad \forall \boldsymbol{j} \in \{1, \dots, d\}^m$$

- A *homogeneous* polynomial of degree m is a function $f : \mathbb{R}^d \to \mathbb{R}$ s.t.

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{j}} \beta_{\boldsymbol{j}} \prod_{t=1}^{m} x_{j_t} \qquad \forall \beta_{\boldsymbol{j}} \in \mathbb{R}$$

The cardinality of β is $\binom{d}{m}$, i.e., $O(d^m)$ parameters!

Efficient representation of polynomials (2/2)

- It is easy to see that

$$\sigma_m(\boldsymbol{h}_s^{\mathrm{T}} \boldsymbol{x}) = (\boldsymbol{h}_s^{\mathrm{T}} \boldsymbol{x})^m = \sum_{\boldsymbol{j}} \prod_{t=1}^{m} h_{s,j_t} x_{j_t}$$

- Plugging this in $\hat{y}_{PN}$, we obtain

$$\hat{y}_{PN} = \sum_{\boldsymbol{j}} \beta_{\boldsymbol{j}} \prod_{t=1}^{m} x_{j_t} \quad \text{with} \quad \beta_{\boldsymbol{j}} := \sum_{s=1}^{k} v_s \prod_{t=1}^{m} h_{s,j_t}$$

- Factored weights: only $kd + k$ parameters instead of $O(d^m)$!

Inhomogeneous polynomials

- In practice, we would like to use monomials of degree 1 up to m, not just m

- By the binomial theorem

$$
\begin{aligned}
&\sigma_m([\boldsymbol{h}\ \gamma]^{\mathrm{T}}[\boldsymbol{x}\ 1]) \\
=&\sigma_m(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x} + \gamma) \\
=&\binom{m}{0}\sigma_m(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x})\gamma^0 + \binom{m}{1}\sigma_{m-1}(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x})\gamma^1 + \cdots + \binom{m}{1}\sigma_0(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x})\gamma^m
\end{aligned}
$$

We can simply augment the data with an all-one feature

Relation with kernel methods

$\sigma_m(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x}+\gamma) = (\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x}+\gamma)^m$ is just the usual polynomial kernel

Kernel methods

$$\hat{y}_{KM} := \sum_{i=1}^{n} \alpha_i \ \sigma_m(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}+\gamma)$$

fix the
hidden layer

learn the
"support vectors"

2-layer polynomial networks

$$\hat{y}_{PN} := \sum_{s=1}^{k} v_s \ \sigma_m(\boldsymbol{h}_s^{\mathrm{T}}\boldsymbol{x}+\gamma)$$

Universality of polynomial networks

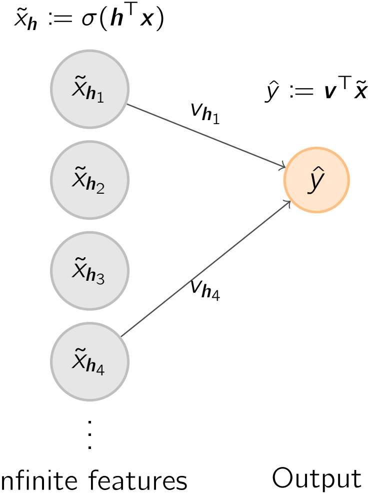
- Polynomials can approximate any function $f: \mathbb{R}^d \to \mathbb{R}$ arbitrarily well on a compact subset of $\mathbb{R}^d$ (Stone-Weierstrass theorem)
- With sufficiently many parameters, PNs can approximate any polynomial arbitrarily well
- And so PNs can approximate any function
- Livni et al. (2014) bound how many layers and units are needed for polynomial networks to approximate sigmoidal networks

Learning PNs: two points of view

- Convex neural networks view (Bengio et al. 2005, Bach 2014)
  - Conditional gradient (a.k.a. Frank-Wolfe) algorithm
- Low-rank matrix / tensor decomposition view (Blondel et al. 2016)
  - Alternating minimization of convex problems
- Both have theoretical guarantees for square activations $\sigma(u) = u^2$

Convex Neural Networks (1/2)

Key idea: learn a sparse linear model in an infinite-dimensional space



Convex Neural Networks (2/2)

- Objective (assume f is smooth with constant β)

$$\min_{\boldsymbol{v}} f(\boldsymbol{v}) := \sum_{i=1}^{n} \ell \left(y_i, \sum_{\|\boldsymbol{h}\|_2 \leq 1} v_{\boldsymbol{h}} \, \sigma(\boldsymbol{h}^{\mathrm{T}} \boldsymbol{x}_i) \right) \text{ s.t. } ||\boldsymbol{v}||_1 \leq \tau$$

- Conditional gradient (a.k.a. Frank-Wolfe) training

Infinite linear model view	Practical implementation
$\boldsymbol{h}^\star = \underset{\|\boldsymbol{h}\|_2 \leq 1}{\operatorname{argmax}} \lvert \nabla_{\boldsymbol{h}} f(\boldsymbol{v}) \rvert$	$\boldsymbol{h}^\star = \underset{\|\boldsymbol{h}\|_2 \leq 1}{\operatorname{argmax}} \lvert \nabla_{\boldsymbol{h}} f(\boldsymbol{v}) \rvert$
$\eta = -\tau \operatorname{sign}\left(\nabla_{\boldsymbol{h}^\star} f(\boldsymbol{v})\right)$	$\boldsymbol{H} \leftarrow [\boldsymbol{H} \ \boldsymbol{h}^\star]$
$\boldsymbol{v} \leftarrow (1-\gamma)\boldsymbol{v} + \gamma \eta \boldsymbol{e}_{\boldsymbol{h}^\star}$	$\boldsymbol{v} \leftarrow [(1-\gamma)\boldsymbol{v} \ \gamma\eta]$

Case of square activation (1/2)

- For ReLu activations, finding $\boldsymbol{h}^\star$ (hidden unit selection problem) is NP-hard (Bach, 2014)
- When using $\sigma_2(u) = u^2$, we can find the optimal $\boldsymbol{h}^\star$ since

$$\begin{aligned}
\nabla_{\boldsymbol{h}} f(\boldsymbol{v}) &= \sum_{i=1}^{n} \ell'(y_i, \hat{y}_i)\sigma_2(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x}_i) \\
&= \boldsymbol{h}^{\mathrm{T}} \left(\sum_{i=1}^{n} \ell'(y_i, \hat{y}_i)\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}} \right) \boldsymbol{h} \\
&=: \boldsymbol{h}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{h}
\end{aligned}$$

$\boldsymbol{h}^\star = \underset{\|\boldsymbol{h}\|_2 \le 1}{\operatorname{argmax}} |\boldsymbol{h}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{h}|$ is the dominant eigenvector of $\boldsymbol{M}$

Case of square activation (2/2)

- Standard analysis of the conditional gradient algorithm guarantees that we can obtain an ϵ-accurate solution in

$$O(\frac{\tau^2 \beta}{\epsilon}) \quad \text{iterations}$$

- Translates into a bound on #hidden units since

$$\#\text{hidden units} \leq \#\text{iterations}$$

Case of factorization machines (FMs)

- FMs are a closely-related model to deal with a large number of pairwise feature interactions (Rendle 2010)
- One can get FMs by replacing (Blondel et al. 2016)

$$\sigma_2(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x}) = (\boldsymbol{h}^{\mathrm{T}}\boldsymbol{x})^2 = \sum_{j,j'} h_j x_j h_{j'} x_{j'}$$

with the ANOVA kernel

$$a_2(\boldsymbol{h}, \boldsymbol{x}) := \sum_{j<j'} h_j x_j h_{j'} x_{j'}$$

FMs are a neural network with a different activation

Case of cubic activation

- When using $\sigma_3(u) = u^3$, we now need to solve

$$\operatorname*{argmax}_{\|\boldsymbol{h}\|_2 \le 1} \left| \langle \boldsymbol{\mathcal{M}}, \boldsymbol{h} \otimes \boldsymbol{h} \otimes \boldsymbol{h} \rangle \right|$$

$$\text{where} \quad \boldsymbol{\mathcal{M}} := \sum_{i=1}^{n} \ell'(y_i, \hat{y}_i) \boldsymbol{x}_i \otimes \boldsymbol{x}_i \otimes \boldsymbol{x}_i \in \mathbb{R}^{d \times d \times d}$$

- Can no longer be solved globally unless there exists an orthogonal decomposition of $\boldsymbol{\mathcal{M}}$

Recent works using conditional gradient like approach

	σ_2	σ_3	a_2	refitting	regularized
Livni et. al (2014)	✓	✓		✓	
Blondel et. al (2015)	✓		✓	✓	✓
Yamada et. al (2015)			✓		✓

- refitting: whether $\boldsymbol{v}$ is refitted over its current support after adding a new hidden unit
- regularized: whether $\boldsymbol{v}$ is regularized by the l_1 norm

Multi-linearity property of ANOVA activations

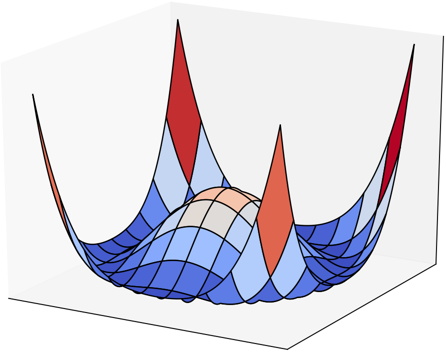
- Let $\hat{y}_{FM} = \sum_{s=1}^{k} v_s a_2(\boldsymbol{h}_s, \boldsymbol{x})$
- Then there exist $\boldsymbol{\alpha}_j \in \mathbb{R}^k$ and $\beta_j \in \mathbb{R}$ s.t.

$$\hat{y}_{FM} = \boldsymbol{\alpha}_j^{\mathrm{T}} \boldsymbol{h}_{:,j} + \beta_j \quad \forall j \in [d]$$

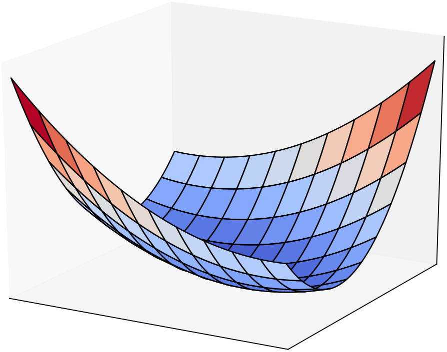
  i.e., $\hat{y}_{FM}$ is affine in $\boldsymbol{h}_{:,j}$ given everything else fixed

- This implies that $\ell(y, \hat{y}_{FM})$ is convex in $\boldsymbol{h}_{:,j}$ for any convex loss function ℓ

Objective surface w.r.t. one column of $\boldsymbol{H}$, $\boldsymbol{h}_{:,j}$



Square activation (σ_2)



Second-order anova activation (a_2)

Low-rank matrix decomposition view

- We can view PNs / FMs as learning a low-rank matrix

$$\hat{y}_{PN} = \sum_{s=1}^{k} v_s \ \sigma_2(\boldsymbol{h}_s^{\mathrm{T}} \boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{W} \boldsymbol{x} = \sum_{j,j'} w_{j,j'} x_j x_{j'}$$

$$\hat{y}_{FM} = \sum_{s=1}^{k} v_s \ a_2(\boldsymbol{h}_s, \boldsymbol{x}) = \sum_{j<j'} w_{j,j'} x_j x_{j'}$$

$$\text{where} \quad \boldsymbol{W} := \sum_{s=1}^{k} v_s \boldsymbol{h}_s \boldsymbol{h}_s^{\mathrm{T}} \in \mathbb{R}^{d \times d}$$

Link with nuclear norm (1/2)

- Nuclear norm (a.k.a. trace norm) of a symmetric matrix

$$\|\boldsymbol{W}\|_* = \|\boldsymbol{v}\|_1$$

where $\boldsymbol{W} = \sum_{s=1}^{\text{rank}(\boldsymbol{W})} v_s \boldsymbol{h}_s \boldsymbol{h}_s^{\mathrm{T}}$ (eigendecomposition of $\boldsymbol{W}$)

- This gives us a link between the convex neural network view and the matrix decomposition view

Link with nuclear norm (2/2)

$$\min_{\boldsymbol{v}} \sum_{i=1}^{n} \ell \left(y_i, \sum_{\boldsymbol{h}:\ \|\boldsymbol{h}\|_2 \leq 1} v_{\boldsymbol{h}}\ \sigma_2(\boldsymbol{h}^{\mathrm{T}} \boldsymbol{x}_i) \right) \text{ s.t. } ||\boldsymbol{v}||_1 \leq \tau$$

$$\Updownarrow$$

$$\min_{\boldsymbol{W} \in \mathbb{R}^{d \times d}} \sum_{i=1}^{n} \ell \left(y_i, \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{W} \boldsymbol{x}_i \right) \text{ s.t. } ||\boldsymbol{W}||_* \leq \tau$$

Can be solved using projected gradient descent

Bi-convex formulation

- We consider the change of variable $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$
- and use the well-known variational formulation

$$\|\boldsymbol{W}\|_* = \min_{\boldsymbol{U},\boldsymbol{V}} \frac{1}{2}(\|\boldsymbol{U}\|^2 + \|\boldsymbol{V}\|^2) \text{ s.t. } \boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$$

- which leads us (Blondel et al. 2016) to

$$\min_{\substack{\boldsymbol{U}\in\mathbb{R}^{d\times k}\\ \boldsymbol{V}\in\mathbb{R}^{d\times k}}} \sum_{i=1}^{n} \ell(y_i, \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}_i) \text{ s.t. } \frac{1}{2}(\|\boldsymbol{U}\|^2 + \|\boldsymbol{V}\|^2) \leq \tau$$

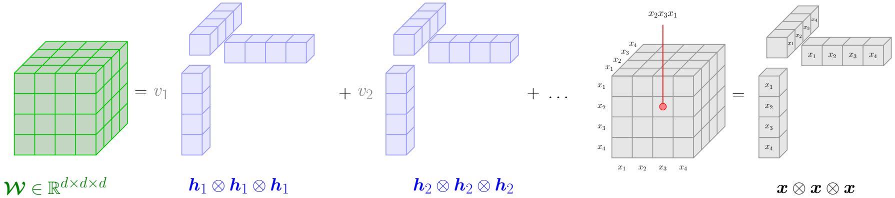
All local minima are global provided that $k \geq \text{rank}(\boldsymbol{W}^\star)$

Case of cubic activation (1/2)

- We can view PNs as learning a low-rank tensor

$$\hat{y}_{PN} = \sum_{s=1}^{k} v_s \, \sigma_3(\boldsymbol{h}_s^{\mathrm{T}} \boldsymbol{x}) = \langle \mathcal{W}, \boldsymbol{x} \otimes \boldsymbol{x} \otimes \boldsymbol{x} \rangle$$

$$= \sum_{j_1, j_2, j_3} w_{j_1, j_2, j_3} x_{j_1} x_{j_2} x_{j_3}$$



Case of cubic activation (2/2)

- We can decompose $\mathcal{W}$ into 3 matrices $\boldsymbol{U}^{(1)}$, $\boldsymbol{U}^{(2)}$, $\boldsymbol{U}^{(3)}$ (objective is block-wise convex)
- No more link with nuclear norm but we can use $\frac{1}{2}(\|\boldsymbol{U}^{(1)}\|^2 + \|\boldsymbol{U}^{(2)}\|^2 + \|\boldsymbol{U}^{(3)}\|^2) \leq \tau$ as a heuristic regularizer
- No global minimum guarantee anymore but alternating minimization works well in practice

Case of higher-order FMs

- Higher-order FMs correspond to using the ANOVA kernel of degree m as activation

$$a_m(\boldsymbol{h}, \boldsymbol{x}) := \sum_{j_1 < \cdots < j_m} h_{j_1} x_{j_1} \dots h_{j_m} x_{j_m}$$

- Naive computation takes $O(d^m)$ time
- We proposed dynamic programming algorithms to compute both the ANOVA kernel and its gradient in $O(dm)$ time (Blondel et al. 2016)

All-subsets activation

- The all-subsets kernel (Shawe-Taylor and Cristianini 2004)

$$S(\boldsymbol{h}, \boldsymbol{x}) := \prod_{j=1}^{d} (1 + h_j x_j)$$

- Corresponds to summing a_0 to a_d

$$S(\boldsymbol{h}, \boldsymbol{x}) = \sum_{t=0}^{d} a_t(\boldsymbol{h}, \boldsymbol{x}) = 1 + \boldsymbol{h}^{\mathrm{T}}\boldsymbol{x} + \sum_{t=2}^{d} a_t(\boldsymbol{h}, \boldsymbol{x})$$

  Hence uses all possible d-combinations of features

- Both the kernel and its gradient can be computed in $O(d)$ time

Some other recent related works

- Chen and Manning 2014: use cubic activation on the task of dependency parsing and train with Adagrad
- Stoudenmire and Schwab (2016), Novikov et al (2016): replace CP decomposition by tensor networks (a.k.a. tensor train) and use all *d*-combinations
- Gautier et al (2016): develop a training algorithm for PN with global optimality guarantee under the following restrictions
  - Impose non-negativity on parameter weights
  - Need one hyper-parameter per hidden unit

Experimental results

Solver comparison (1/2)

Goal: check whether optimizing the bi-convex formulation is advantageous compared to direct formulation

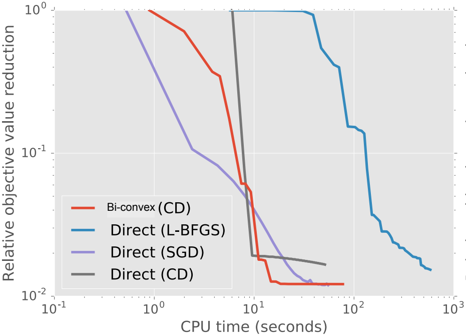
- Bi-convex formulation (PN case)

$$\min_{\substack{\boldsymbol{U}\in\mathbb{R}^{d\times k}\\ \boldsymbol{V}\in\mathbb{R}^{d\times k}}} \sum_{i=1}^{n} \ell(y_i, \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}} \boldsymbol{x}_i) + \frac{\lambda}{2}(\|\boldsymbol{U}\|^2 + \|\boldsymbol{V}\|^2)$$

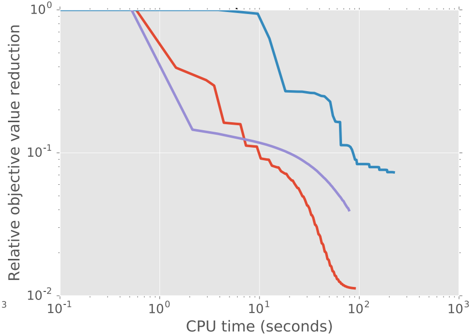
- Direct formulation (PN case)

$$\min_{\substack{\boldsymbol{H}\in\mathbb{R}^{k\times d}\\ \boldsymbol{v}\in\mathbb{R}^{k}}} \sum_{i=1}^{n} \ell(y_i, \sum_{s=1}^{k} v_s \sigma_2(\boldsymbol{h}_s^{\mathrm{T}} \boldsymbol{x}_i)) + \lambda \sum_{s=1}^{k} |v_s| \|\boldsymbol{h}_s\|^2$$

Solver comparison (2/2)



Second-order anova activation (a_2)



Square activation (σ_2)

E2006-tfidf dataset

n = 16,087, d = 150,360

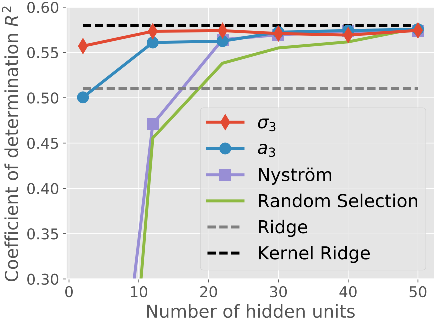
Low-budget polynomial regression (1/2)

Goal: learn small polynomial regression model

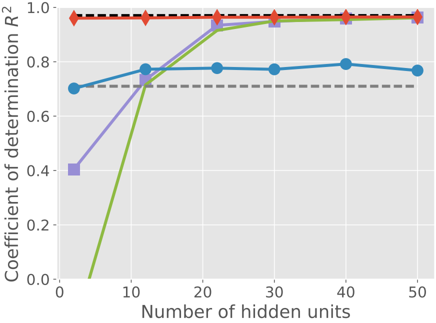
We compared the following methods

- PN with σ_3 activation (trained by coordinate descent)
- FM with a_3 activation (trained by coordinate descent)
- Random selection: fix hidden units as training samples and fit output layer only
- Nyström method
- Linear and kernel ridge regression

Low-budget polynomial regression (2/2)



Abalone



Cpusmall

Application to recommender systems

- 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

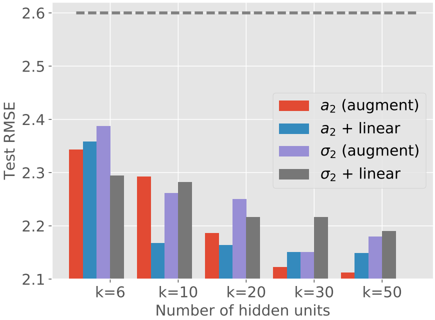
	Movie 1	Movie 2	Movie 3	Movie 4
Alice	⋆⋆	?	⋆⋆⋆	?
Bob	⋆	?	⋆⋆	?
Charlie	⋆⋆	?	?	⋆⋆

$\Downarrow$ one-hot encoding

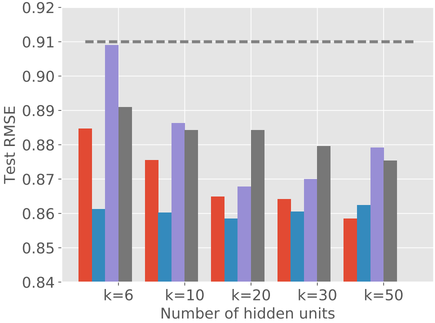
$$\underbrace{\begin{bmatrix} \star\star \\ \star\star\star \\ \star \\ \star\star \\ \star\star \\ \star\star \end{bmatrix}}_{\boldsymbol{y}} \quad \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 &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

Using this representation, FMs are equivalent to MF!

Application to recommender systems



Last.fm



MovieLens 1M

Conclusion

- PNs and FMs learn efficient representations of polynomials
- PNs: feature combinations with replacement
  - e.g., $x_{j_1}^3$, $x_{j_1}^2 x_{j_2}$, $x_{j_1} x_{j_2} x_{j_3}$
- FMs: feature combinations without replacement
  - e.g., $x_{j_1} x_{j_2} x_{j_3}$
- PNs and FMs are useful for learning fast-to-evaluate polynomial models and for recommender systems

Questions?

References

- Bach. Breaking the curse of dimensionality with convex neural networks. arXiv preprint 2014.
- Bengio et al. Convex neural networks. In NIPS, 2005.
- Blondel et al. Convex factorization machines. In ECML/PKDD, 2015
- Blondel et al. Polynomial Networks and Factorization Machines: New Insights and Efficient Training Algorithms. In ICML 2016.
- Blondel et al. Higher-order Factorization Machines. In NIPS 2016.

References

- Gautier et al. Globally optimal training of generalized polynomial neural networks with nonlinear spectral methods. In NIPS, 2016.
- Livni et al. On the computational efficiency of training neural networks. In NIPS 2014.
- Novikov et al. Exponential machines. arXiv preprint 2016.

References

- Rendle. Factorization machines. In ICDM 2010.
- Shawe-Taylor, John and Cristianini, Nello. Kernel Methods for Pattern Analysis. Cambridge University Press, 2004.
- Stoudenmire and Schwab. Supervised learning with tensor networks. In NIPS, 2016.
- Yamada et al. Convex Factorization Machine for Regression. arXiv preprint 2015.