Recent advances on polynomial neural networks and factorization machines

Mathieu Blondel

NTT Communication Science Laboratories
Kyoto, Japan

2017/2/23
Neural networks

\[ \tilde{x}_s := \sigma(h_s^T x), s \in [k] \]

\[ \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k \]

\[ \hat{y} := v^T \tilde{x} \]

\[ H \in \mathbb{R}^{k \times d} \]

\[ v \in \mathbb{R}^k \]
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

\[ \hat{y}_{PN} := \sum_{s=1}^{k} \nu_s \sigma_m(h_s^T x) \]

- 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 \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) s.t.

\[
f(x) = \prod_{t=1}^{m} x_{j_t} = x_{j_1} x_{j_2} \cdots x_{j_m} \quad \forall j \in \{1, \ldots, d\}^m
\]

- A homogeneous polynomial of degree \( m \) is a function \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) s.t.

\[
f(x) = \sum_{j} \beta_j \prod_{t=1}^{m} x_{j_t} \quad \forall \beta_j \in \mathbb{R}
\]

The cardinality of \( \beta \) is \( \binom{d}{m} \), i.e., \( O(d^m) \) parameters!
Efficient representation of polynomials (2/2)

- It is easy to see that

\[ \sigma_m(h_s^T x) = (h_s^T x)^m = \sum_j h_{s,j_t} x_{j_t} \]

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

\[ \hat{y}_{PN} = \sum_j \beta_j \prod_{t=1}^m x_{j_t} \quad \text{with} \quad \beta_j := \sum_{s=1}^k \nu_s \prod_{t=1}^m h_{s,j_t} \]

- 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

\[
\sigma_m([h \ \gamma]^T[x \ 1]) \\
= \sigma_m(h^T x + \gamma) \\
= \binom{m}{0} \sigma_m(h^T x) \gamma^0 + \binom{m}{1} \sigma_{m-1}(h^T x) \gamma^1 + \cdots + \binom{m}{1} \sigma_0(h^T x) \gamma^m
\]

We can simply augment the data with an all-one feature.
Relation with kernel methods

\[ \sigma_m(h^T x + \gamma) = (h^T x + \gamma)^m \] is just the usual polynomial kernel

Kernel methods

\[ \hat{y}_{KM} := \sum_{i=1}^{n} \alpha_i \sigma_m(x_i^T x + \gamma) \]

2-layer polynomial networks

\[ \hat{y}_{PN} := \sum_{s=1}^{k} v_s \sigma_m(h_s^T x + \gamma) \]

learn the “support vectors”
fix the hidden layer
Universality of polynomial networks

- Polynomials can approximate any function $f: \mathbb{R}^d \rightarrow \mathbb{R}$ 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

\[ \tilde{x}_h := \sigma(h^T x) \]

\[ \hat{y} := v^T \tilde{x} \]

Infinite features

Output
Convex Neural Networks (2/2)

- Objective (assume $f$ is smooth with constant $\beta$)

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

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

Infinite linear model view

$$
\mathbf{h}^* = \arg\max_{\|\mathbf{h}\|_2 \leq 1} |\nabla_h f(\mathbf{v})| \\
\eta = -\tau \text{sign} (\nabla_{h^*} f(\mathbf{v})) \\
\mathbf{v} \leftarrow (1 - \gamma) \mathbf{v} + \gamma \eta e_{h^*}
$$

Practical implementation

$$
\mathbf{h}^* = \arg\max_{\|\mathbf{h}\|_2 \leq 1} |\nabla_h f(\mathbf{v})| \\
\mathbf{H} \leftarrow [\mathbf{H} \mathbf{h}^*] \\
\mathbf{v} \leftarrow [(1 - \gamma) \mathbf{v} \ \gamma \eta]
$$
Case of square activation (1/2)

- For ReLu activations, finding $h^\star$ (hidden unit selection problem) is NP-hard (Bach, 2014)

- When using $\sigma_2(u) = u^2$, we can find the optimal $h^\star$ since

$$
\nabla_h f(v) = \sum_{i=1}^{n} \ell'(y_i, \hat{y}_i) \sigma_2(h^T x_i)
= h^T \left( \sum_{i=1}^{n} \ell'(y_i, \hat{y}_i) x_i x_i^T \right) h
= : h^T M h
$$

$h^\star = \text{argmax} \left| h^T M h \right| \text{ is the dominant eigenvector of } M$
Case of square activation (2/2)

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

$$O\left(\frac{\tau^2 \beta}{\epsilon}\right)$$ iterations

- Translates into a bound on $\#\text{hidden units}$ 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)

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

with the ANOVA kernel

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

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

$$\arg\max \left| \left\langle \mathcal{M}, h \otimes h \otimes h \right\rangle \right| \quad \|h\|_2 \leq 1$$

where

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

- Can no longer be solved globally unless there exists an orthogonal decomposition of $\mathcal{M}$
Recent works using conditional gradient like approach

<table>
<thead>
<tr>
<th></th>
<th>$\sigma_2$</th>
<th>$\sigma_3$</th>
<th>$a_2$</th>
<th>refitting</th>
<th>regularized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Livni et. al (2014)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Blondel et. al (2015)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Yamada et. al (2015)</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **refitting:** whether $\bm{v}$ is refitted over its current support after adding a new hidden unit
- **regularized:** whether $\bm{v}$ is regularized by the $l_1$ norm
Multi-linearity property of ANOVA activations

- Let \( \hat{y}_{FM} = \sum_{s=1}^{k} v_s a_2(h_s, x) \)

- Then there exist \( \alpha_j \in \mathbb{R}^k \) and \( \beta_j \in \mathbb{R} \) s.t.

\[
\hat{y}_{FM} = \alpha_j^T h_{ij} + \beta_j \quad \forall j \in [d]
\]

i.e., \( \hat{y}_{FM} \) is affine in \( h_{ij} \) given everything else fixed

- This implies that \( \ell(y, \hat{y}_{FM}) \) is convex in \( h_{ij} \) for any convex loss function \( \ell \)
Objective surface w.r.t. one column of $H, h_{i:j}$

Square activation ($\sigma_2$)  
Second-order anova activation ($a_2$)
Low-rank matrix decomposition view

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

\[
\hat{y}_{PN} = \sum_{s=1}^{k} v_s \sigma_2(h_s^T x) = x^T W x = \sum_{j,j'} w_{j,j'} x_j x_{j'}
\]

\[
\hat{y}_{FM} = \sum_{s=1}^{k} v_s a_2(h_s, x) = \sum_{j<j'} w_{j,j'} x_j x_{j'}
\]

where \( W := \sum_{s=1}^{k} v_s h_s h_s^T \in \mathbb{R}^{d \times d} \)
Link with nuclear norm (1/2)

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

\[ \| W \|_* = \| v \|_1 \]

where

\[ W = \sum_{s=1}^{\text{rank}(W)} v_s h_s h_s^T \] (eigendecomposition of \( W \))

- This gives us a link between the convex neural network view and the matrix decomposition view
Link with nuclear norm (2/2)

\[
\begin{align*}
\min_v & \sum_{i=1}^{n} \ell \left( y_i, \sum_{h: \|h\|_2 \leq 1} v^*_h \sigma_2(h^T x_i) \right) \quad \text{s.t. } \|v\|_1 \leq \tau \\
\iff \quad & \\
\min_{W \in \mathbb{R}^{d \times d}} & \sum_{i=1}^{n} \ell \left( y_i, x_i^T W x_i \right) \quad \text{s.t. } \|W\|_* \leq \tau
\end{align*}
\]

Can be solved using projected gradient descent
Bi-convex formulation

- We consider the change of variable $W = UV^T$
- and use the well-known variational formulation
  \[
  \|W\|_* = \min_{U,V} \frac{1}{2}(\|U\|^2 + \|V\|^2) \text{ s.t. } W = UV^T
  \]
- which leads us (Blondel et al. 2016) to
  \[
  \min_{U \in \mathbb{R}^{d \times k}, V \in \mathbb{R}^{d \times k}} \sum_{i=1}^{n} \ell(y_i, x_i^T UV^T x_i) \text{ s.t. } \frac{1}{2}(\|U\|^2 + \|V\|^2) \leq \tau
  \]

All local minima are global provided that $k \geq \text{rank}(W^*)$
Case of cubic activation (1/2)

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

\[
\hat{y}_{PN} = \sum_{s=1}^{k} v_s \sigma_3(h_s^T x) = \langle \mathcal{W}, x \otimes x \otimes x \rangle \\
= \sum_{j_1, j_2, j_3} w_{j_1, j_2, j_3} x_{j_1} x_{j_2} x_{j_3}
\]

\[\mathcal{W} \in \mathbb{R}^{d \times d \times d} \]

\[h_1 \otimes h_1 \otimes h_1 \]

\[h_2 \otimes h_2 \otimes h_2 \]

\[x \otimes x \otimes x\]
Case of cubic activation (2/2)

- We can decompose $\mathcal{W}$ into 3 matrices $U^{(1)}, U^{(2)}, U^{(3)}$ (objective is block-wise convex).

- No more link with nuclear norm but we can use
  \[ \frac{1}{2}(\|U^{(1)}\|^2 + \|U^{(2)}\|^2 + \|U^{(3)}\|^2) \leq \tau \] 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
  \[
  a_m(h, x) := \sum_{j_1 < \cdots < j_m} h_{j_1} x_{j_1} \cdots h_{j_m} x_{j_m}
  \]
- 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)
  \[
  S(h, x) := \prod_{j=1}^{d} (1 + h_j x_j)
  \]
- Corresponds to summing \(a_0\) to \(a_d\)
  \[
  S(h, x) = \sum_{t=0}^{d} a_t(h, x) = 1 + h^T x + \sum_{t=2}^{d} a_t(h, x)
  \]
  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)
  \[
  \min_{U \in \mathbb{R}^{d \times k}} \sum_{i=1}^{n} \ell(y_i, x_i^T U V^T x_i) + \frac{\lambda}{2} (\|U\|^2 + \|V\|^2)
  \]

- Direct formulation (PN case)
  \[
  \min_{H \in \mathbb{R}^{k \times d}} \sum_{i=1}^{n} \ell(y_i, \sum_{s=1}^{k} v_s \sigma_2(h_s^T x_i)) + \lambda \sum_{s=1}^{k} |v_s| \|h_s\|^2
  \]
Solver comparison (2/2)

Second-order anova activation \((a_2)\)  

Square activation \((\sigma_2)\)

E2006-tfidf dataset

\[ n = 16,087, \quad d = 150,360 \]
Low-budget polynomial regression (1/2)

Goal: learn small polynomial regression model

We compared the following methods

- PN with $\sigma_3$ 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

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>★★</td>
<td>?</td>
<td>★★★</td>
<td>?</td>
</tr>
<tr>
<td>Bob</td>
<td>★</td>
<td>?</td>
<td>★★</td>
<td>?</td>
</tr>
<tr>
<td>Charlie</td>
<td>★★</td>
<td>?</td>
<td>?</td>
<td>★★</td>
</tr>
</tbody>
</table>

- Matrix factorization: find $U, V$ that approximately reconstruct the rating matrix

$$R \approx UV^\top$$
Conversion to a regression problem

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>★★</td>
<td>?</td>
<td>★★★</td>
<td>?</td>
</tr>
<tr>
<td>Bob</td>
<td>★</td>
<td>?</td>
<td>★★</td>
<td>?</td>
</tr>
<tr>
<td>Charlie</td>
<td>★★</td>
<td>?</td>
<td>?</td>
<td>★★</td>
</tr>
</tbody>
</table>

\[ \downarrow \text{one-hot encoding} \]

\[
\begin{bmatrix}
\text{★★} \\
\text{★★★} \\
\text{★} \\
\text{★★} \\
\text{★★}
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

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., $x_{j_1}^3$, $x_{j_1}^2x_{j_2}$, $x_{j_1}x_{j_2}x_{j_3}$
- 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


References


