Fastfood – Approximating Kernel Expansions in Loglinear Time

Quoc Le, Tamas Sarlos, and Alex Smola
Presenter: Shuai Zheng (Kyle)
Large Scale Problem: ImageNet Challenge

• Large scale data
  – Number of training examples $m = 1,200,000$ in ILSVRC1000 dataset.
  – Dimensions of encoded features for most algorithm, $d$ is more than 20,000. (You can get better performance with bigger dictionary dimension, but you might have memory limit issue.)
  – Number of support vectors: usually $n > 0.1*m$
## SVM Tradeoffs

<table>
<thead>
<tr>
<th></th>
<th>Linear Kernel</th>
<th>Non-linear Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training speed</td>
<td>Very fast</td>
<td>Very slow</td>
</tr>
<tr>
<td>Training scalability</td>
<td>Very high</td>
<td>Low</td>
</tr>
<tr>
<td>Testing speed</td>
<td>Very fast</td>
<td>Very slow</td>
</tr>
<tr>
<td>Testing accuracy</td>
<td>Lower</td>
<td>Higher</td>
</tr>
</tbody>
</table>

How to get all yellow characteristics?
Additive Kernels (e.g. Efficient Additive Kernels via Explicit Feature Maps, PAMI 2011).

New direction: Approximate Kernel with fake random Gaussian Matrices (Fastfood).
When Kernel Methods meet Large Scale Problem

• In kernel methods, for large scale problems, computing the decision function is expensive, especially at prediction time.

• So shall we give up nonlinear kernel methods at all?
  – No, we have better approximation solution.
  – Turn to linear SVM + (features + complicated encoding (LLC, Fisher coding, group saliency coding) + sophisticated pooling (max pooling, average pooling, learned pooling)), and now neural network.
High-dimensional Problem vs Kernel approximation

Kernel expansion\[ f(x) = \langle w, \phi(x) \rangle = \left( \sum_{i=1}^{n} \alpha_i \phi(x_i)\phi(x) \right) = \sum_{i=1}^{m} \alpha_i k(x_i, x) \]

- $d$ is the input feature dimension,
- $n$ is the number of nonlinear basis functions
- $m$ is the number of samples.

<table>
<thead>
<tr>
<th></th>
<th>CPU Training</th>
<th>RAM Training</th>
<th>CPU Test</th>
<th>RAM Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>O($m^2d$)</td>
<td>O(md)</td>
<td>O(md)</td>
<td>O(md)</td>
</tr>
<tr>
<td>Reduced set</td>
<td>O($m^2d$)</td>
<td>O(md)</td>
<td>O(nd)</td>
<td>O(nd)</td>
</tr>
<tr>
<td>Low rank</td>
<td>O($mnd$)</td>
<td>O(nd)</td>
<td>O(nd)</td>
<td>O(nd)</td>
</tr>
<tr>
<td>Random Kitchen Sinks (RKS)</td>
<td>O($mnd$)</td>
<td>O(nd)</td>
<td>O(nd)</td>
<td>O(nd)</td>
</tr>
<tr>
<td>Fastfood</td>
<td>O($mn \log d$)</td>
<td>O($n \log d$)</td>
<td>O($n \log d$)</td>
<td>O($n$)</td>
</tr>
</tbody>
</table>
## High-dimensional Problem vs Fastfood

Kernel expansion

\[ f(x) = \langle w, \phi(x) \rangle = \left( \sum_{i=1}^{n} \alpha_i \phi(x_i)\phi(x) \right) = \sum_{i=1}^{m} \alpha_i k(x_i, x) \]

- \(d\) is the input feature dimension, e.g. \(d = 20,000\).
- \(n\) is the number of nonlinear basis functions, e.g. \(n = 120,000\).
- \(m\) is the number of samples, e.g. \(m = 1,200,000\).

<table>
<thead>
<tr>
<th></th>
<th>CPU Training</th>
<th>RAM Training</th>
<th>CPU Test</th>
<th>RAM Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>(O(m^2d))</td>
<td>(O(md))</td>
<td>(O(md))</td>
<td>(O(md))</td>
</tr>
<tr>
<td>Reduced set</td>
<td>(O(m^2d))</td>
<td>(O(md))</td>
<td>(O(nd))</td>
<td>(O(nd))</td>
</tr>
<tr>
<td>Low rank</td>
<td>(O(mnd))</td>
<td>(O(nd))</td>
<td>(O(nd))</td>
<td>(O(nd))</td>
</tr>
<tr>
<td>Random Kitchen Sinks (RKS)</td>
<td>(O(mnd))</td>
<td>(O(nd))</td>
<td>(O(nd))</td>
<td>(O(nd))</td>
</tr>
<tr>
<td>Fastfood</td>
<td>(O(mn \log d))</td>
<td>(O(n \log d))</td>
<td>(O(n \log d))</td>
<td>(O(n))</td>
</tr>
</tbody>
</table>
Random Kitchen Sinks
(Rahimi & Recht, NIPS 2007)

• Given a training set \( \{x_i, y_i\}_{i=1}^m \), the task is to fit a decision function \( f: \chi \rightarrow \mathbb{R} \) that minimize the empirical risk

\[
R_{emp}(f) = \frac{1}{m} \sum_{i=1}^{m} l(f(x_i), y_i)
\]

Where \( l(f(x_i), y_i) \) denotes the loss function, e.g. Hinge loss, etc.

Decision function is

\[
f(x) = \langle w, \phi(x) \rangle = \left\langle \sum_{i=1}^{n} \alpha_i \phi(x_i) \phi(x) \right\rangle = \sum_{i=1}^{m} \alpha_i k(x_i, x)
\]
Random Kitchen Sinks
(Rahimi & Recht, NIPS 2007)

• Most algorithms produce an approximate minimizer of the empirical risk by optimizing over:

$$\min_{w_1, \ldots, w_n} R_{emp} \left( \sum_j \alpha(w_j) \phi(x; w_j) \right)$$
Random Kitchen Sinks
(Rahimi & Recht, NIPS 2007)

• RKS attempts to approximate the decision function \( f \) via

\[
f(x) = \sum_{j=1}^{m} \alpha_j \hat{\phi}(x; w_j)
\]

Where \( \hat{\phi}(x; w_j) \) is a feature function obtained via randomization.
Random Kitchen Sinks
(Rahimi & Recht, NIPS 2007)

• Rather than computing Gaussian RBF kernel,
  \[ k(x, x') = \exp\left(-\|x - x'\|^2 / (2\sigma^2)\right) \]

• This method computes \( k(x, x') = \exp(i[Zx]_j) \)
  by drawing \( z_i \) from a normal distribution.
Randomization for Approximate Gaussian RBF kernel feature maps

• Input: Input data \{x_i\}_{i=1}^m, we have n basis functions, scale parameter is \sigma^2
• Output: Approximate Gaussian RBF kernel feature map \phi(x).

1. Draw each entry of \( Z \in \mathbb{R}^d \) iid from Normal(0, \( \sigma^{-2} \))
2. For i = 1 to m do
3. For j = 1 to n do
4. Compute the empirical feature map \( \frac{1}{\sqrt{n}} \exp(i|Zx_i|_j) \)
5. End for
6. End for

Complexity: \( O(mnd) \)
Fastfood

• The dimension of feature is d.
• The number of basis functions is n.
• Gaussian matrix cost $O(nd)$ per multiplication.
• Assume $d = 2^l$ (Pad the vector with zeros until $d = 2^l$ holds), The goal is to approximate $Z$ via a product of diagonal and simple matrices:

$$
\tilde{Z} = \frac{1}{\sigma \sqrt{d}} SHG\Pi HB
$$
Fastfood

• Assume $d = 2^l$ (Pad the vector with zeros until $d = 2^l$ holds), The goal is to approximate $Z$ via a product of diagonal and simple matrices:

$$\tilde{Z} = \frac{1}{\sigma \sqrt{d}} SHG \Pi HB$$

$S$ random diagonal scaling matrix

$H$ Walsh-Hadamard matrix admitting $O(d \log(d))$ multiply

$$H_{2d} = \begin{bmatrix} H_d & H_d \\ H_d & -H_d \end{bmatrix} \text{ and } H_1 = 1, H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$G$ random diagonal Gaussian matrix

$\Pi \in \{-1,1\}^{d \times d}$ permutation matrix

$B$ a matrix which has random$\{-1,1\}$ entries on its diagonal

Multiplication now is $O(d \log d)$, storage is $O(d)$. Draw independent blocks.
Fastfood

• When $n > d$, we replicate

$$\tilde{Z} = \frac{1}{\sigma \sqrt{d}} SHG \Pi HB$$

• For $n/d$ independent random matrices $Z_i$ and stack them via $\tilde{Z}^T = [\tilde{Z}_1, \tilde{Z}_2, ..., \tilde{Z}_{n/d}]$ until we have enough dimensions.
Walsh-Hadamard transform

The **product** of a **Boolean function** and a **Walsh matrix** is its Walsh spectrum[wiki]:

\[(1,0,1,0,0,1,1,0) \times H(8) = (4,2,0,-2,0,2,0,2)\]
Fast Walsh-Hadamard transform

This is a faster way to calculate the Walsh spectrum of \((1,0,1,0,0,1,1,0)\).

\[
\begin{array}{cccc}
1 & 1+0 = 1 & 1+2 = 3 & 3+1 = 4 \\
0 & 0+1 = 1 & 1+0 = 1 & -1+3 = 2 \\
1 & 1+1 = 2 & -2+1 = -1 & -1+1 = 0 \\
0 & 0+0 = 0 & -0+1 = 1 & -1+(-1) = -2 \\
0 & -0+1 = 1 & 1+0 = 1 & 1+(-1) = 0 \\
0 & -1+0 = -1 & -1+0 = -1 & (-1)+1 = 2 \\
1 & -1+1 = 0 & -0+1 = 1 & 1+(-1) = 0 \\
1 & -1+1 = 0 & -0+(-1) = -1 & (-1)+1 = 2 \\
0 & -0+0 = 0 & -0+(-1) = -1 & (-1)+1 = 2
\end{array}
\]
Key Observations

1. When combined with diagonal Gaussian matrices, Hadamard matrices exhibit very similar to dense Gaussian random matrices.

2. Hadamard matrices are inexpensive to multiply and store.
Key Observations

1. When combined with diagonal Gaussian matrices, Hadamard matrices exhibit very similar to dense Gaussian random matrices.

2. Hadamard matrices and diagonal matrices are inexpensive to multiply (O(dlog(d))) and store O(d).

Code:

- For C++/C user, a library called SPIRAL provides extremely fast Fast Hadamard transform.
- For Matlab user, one line code y = fwhit(x,n,ordering)
## Experiments

Runtime, speed, and memory improvements of Fastfood relative to random kitchen sinks (RKS).

<table>
<thead>
<tr>
<th>d</th>
<th>n</th>
<th>Fastfood</th>
<th>RKS</th>
<th>Speedup</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>16384</td>
<td>0.00058s</td>
<td>0.0139s</td>
<td>24x</td>
<td>256x</td>
</tr>
<tr>
<td>4096</td>
<td>32768</td>
<td>0.00136s</td>
<td>0.1224s</td>
<td>90x</td>
<td>1024x</td>
</tr>
<tr>
<td>8192</td>
<td>65536</td>
<td>0.00268s</td>
<td>0.5360s</td>
<td>200x</td>
<td>2048x</td>
</tr>
</tbody>
</table>

- d - input feature dimension, e.g. CIFAR-10, a tiny image $32*32*3$ has 3072 dimensions.
- n – n nonlinear basis functions.
Summary

• It is possible to compute n nonlinear basis functions in $O(n \log d)$ time.
• Kernel methods become more practical for problems that have large datasets and/or require real-time prediction.
• With Fastfood, we would be able to compute nonlinear feature map for large-scale problem.