KERNELS AND PERCEPTRONS
The perceptron

\( x \) is a vector
\( y \) is -1 or +1

Compute:
\[ y_i = \text{sign}(v_k \cdot x_i) \]

If mistake:
\[ v_{k+1} = v_k + y_i x_i \]

mistake bound:
\[ k \leq \left( \frac{R}{\gamma} \right)^2 \]

Depends on how easy the learning problem is, not dimension of vectors \( x \)

Fairly intuitive:
- “Similarity” of \( v \) to \( u \) looks like
  \[ (v \cdot u)/|v \cdot v| \]
- \((v \cdot u)\) grows by \( \geq \gamma \) after mistake
- \((v \cdot v)\) grows by \( \leq R^2 \)
The kernel perceptron

Compute: $\hat{y}_i = \text{sign}(v_k \cdot x_i)$

If mistake: $v_{k+1} = v_k + y_i x_i$

Mathematically the same as before … but allows use of the kernel trick
The kernel perceptron

\[ K(x, x_k) \equiv x \cdot x_k \]

\[ \hat{y} = \text{sign}( \sum_{x_{k^+} \in FN} K(x_i, x_{k^+}) - \sum_{x_{k^-} \in FP} K(x_i, x_{k^-}) ) \]

Compute: \( \hat{y} = \sum_{x_{k^+} \in FN} x_i \cdot x_{k^+} - \sum_{x_{k^-} \in FP} x_i \cdot x_{k^-} \)

Mathematically the same as before … but allows use of the “kernel trick”

Other kernel methods (SVM, Gaussian processes) aren’t constrained to limited set (+1/-1/0) of weights on the \( K(x,v) \) values.
Some common kernels

- Linear kernel:
  \[ K(x, x') \equiv x \cdot x' \]

- Polynomial kernel:
  \[ K(x, x') \equiv (x \cdot x' + 1)^d \]

- Gaussian kernel:
  \[ K(x, x') \equiv e^{-\|x-x'|^2 / \sigma} \]

- More later….
Kernels 101

• Duality
  – and computational properties
  – Reproducing Kernel Hilbert Space (RKHS)
• Gram matrix
• Positive semi-definite
• Closure properties
Kernels 101

- Duality: two ways to look at this

\[
\hat{y} = x \cdot w
\]

\[
w = \sum_{x_{k^+} \in FN} x_{k^+} - \sum_{x_{k^-} \in FP} x_{k^-}
\]

\[
\hat{y} = \sum_{x_{k^+} \in FN} K(x, x_{k^+}) - \sum_{x_{k^-} \in FP} K(x, x_{k^-})
\]

\[
K(x, x_k) = \phi(x) \cdot \phi(x_k)
\]

- Two different computational ways of getting the same behavior
Kernels 101

- Duality
- Gram matrix: $k_{ij} = K(x_i, x_j)$

$$K(x, x') = K(x', x) \Rightarrow \text{Gram matrix is symmetric}$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$K(1, 1)$</td>
<td>$K(1, 2)$</td>
<td>$K(1, 3)$</td>
<td>...</td>
<td>$K(1, m)$</td>
</tr>
<tr>
<td>2</td>
<td>$K(2, 1)$</td>
<td>$K(2, 2)$</td>
<td>$K(2, 3)$</td>
<td>...</td>
<td>$K(2, m)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>$K(m, 1)$</td>
<td>$K(m, 2)$</td>
<td>$K(m, 3)$</td>
<td>...</td>
<td>$K(m, m)$</td>
</tr>
</tbody>
</table>
Kernels 101

• Duality
• Gram matrix: \( K: k_{ij} = K(x_i, x_j) \)

\[ K(x, x') = K(x', x) \implies \text{Gram matrix is symmetric} \]

\( K \) is “positive semi-definite” \( \iff \ldots \iff z^T K z \geq 0 \)
for all \( z \)

Fun fact: Gram matrix positive semi-definite \( \iff \)
\( K(x_i, x_j) = \phi(x_i), \phi(x_j) \) for some \( \phi \)

Proof: \( \phi(x) \) uses the eigenvectors of \( K \) to represent \( x \)
HASH KERNELS AND “THE HASH TRICK”
Question

• Most of the weights in a classifier are small and not important
  – So we can use the “hash trick”
Hash Kernels

Qinfeng Shi, James Petterson
Australian National University and NICTA,
Canberra, Australia

John Langford, Alex Smola, Alex Strehl
Yahoo! Research
New York, NY and Santa Clara, CA, USA

Gideon Dror
Department of Computer Science
Academic College of Tel-Aviv-Yaffo, Israel

Vishy Vishwanathan
Department of Statistics
Purdue University, IN, USA
The hash trick as a kernel

• Usually we *implicitly* map from \(x\) to \(\phi(x)\)
  
  – All computations of learner are in terms of
    \[ K(x,x') = \langle \phi(x), \phi(x') \rangle \]
  
  – Because \(\phi(x)\) is large

• In this case we *explicitly* map from \(x\) to \(\phi(x)\)
  
  – \(\phi(x)\) is small
Some details

Slightly different hash to avoid systematic bias

\[ V[h] = \sum_{j: \text{hash}(j) \% R = h} x_i^j \]

\[ \varphi[h] = \sum_{j: \text{hash}(j) \% m = h} \xi(j) x_i^j, \quad \text{where } \xi(j) \in \{-1,+1\} \]

\( m \) is the number of buckets you hash into (\( R \) in my discussion)
Some details

Slightly different hash to avoid systematic bias

\[ \varphi[h] = \sum_{j : \text{hash}(j) \% m = h} \xi(j)x^j_i, \quad \text{where } \xi(j) \in \{-1, +1\} \]

Lemma 2 The hash kernel is unbiased, that is

\[ E_{\varphi}[\langle x, x' \rangle_\varphi] = \langle x, x' \rangle. \text{ Moreover, the variance is} \]

\[ \sigma^2_{x,x'} = \frac{1}{m} \left( \sum_{i \neq j} x_i^2 x_j'^2 + x_i x_i' x_j x_j' \right), \text{ and thus, for} \]

\[ \|x\|_2 = \|x'\|_2 = 1, \sigma^2_{x,x'} = O \left( \frac{1}{m} \right). \]

\[ m \text{ is the number of buckets you hash into (R in my discussion)} \]
Some details

**Theorem 3** Let $\epsilon < 1$ be a fixed constant and $x$ be a given instance. Let $\eta = \frac{\|x\|_{\infty}}{\|x\|_2}$. Under the assumptions above, the hash kernel satisfies the following inequality

$$\Pr \left\{ \left| \frac{\|x\|_{\phi}^2 - \|x\|_2^2}{\|x\|_2^2} \right| \geq \sqrt{2} \sigma_{x,x} + \epsilon \right\} \leq \exp \left( -\frac{\sqrt{\epsilon}}{4\eta} \right).$$

I.e. – a hashed vector is probably close to the original vector
Some details

Corollary 4 For two vectors \( x \) and \( x' \), let us define

\[
\sigma := \max(\sigma_{x,x}, \sigma_{x',x'}, \sigma_{x-x',x-x'})
\]

\[
\eta := \min\left(\frac{\|x\|_\infty}{\|x\|_2}, \frac{\|x'\|_\infty}{\|x'\|_2}, \frac{\|x-x'\|_\infty}{\|x-x'\|_2}\right).
\]

Also let \( \Delta = \|x\|^2 + \|x'\|^2 + \|x-x'\|^2 \). Under the assumptions above, we have that

\[
\Pr\left[|\langle x, x' \rangle_\phi - \langle x, x' \rangle| > (\sqrt{2}\sigma + \epsilon)\Delta/2\right] < 3e^{-\frac{\sqrt{\epsilon}}{4\eta}}.
\]

I.e. the inner products between \( x \) and \( x' \) are probably not changed too much by the hash function: a classifier will probably still work.
Corollary 5  Denote by $X = \{x_1, \ldots, x_n\}$ a set of vectors which satisfy \[ \|x_i - x_j\|_\infty \leq \eta \|x_i - x_j\|_2 \] for all pairs $i, j$. In this case with probability $1 - \delta$ we have for all $i, j$

\[
\frac{|\|x_i - x_j\|_\phi^2 - \|x_i - x_j\|_2^2|}{\|x_i - x_j\|_2^2} \leq \sqrt{\frac{2}{m}} + 64\eta^2 \log^2 \frac{n}{2\delta}.
\]

This means that the number of observations $n$ (or correspondingly the size of the un-hashed kernel matrix) only enters logarithmically in the analysis.
The hash kernel: implementation

• One problem: debugging is harder
  – Features are no longer meaningful
  – There’s a new way to ruin a classifier
    • Change the hash function 😞

• You can separately compute the set of all words that hash to $h$ and guess what features mean
  – Build an inverted index $h \rightarrow w_1, w_2, \ldots,$
ADAPTIVE GRADIENTS
Motivation

• What’s the best learning rate?
  – If a feature is rare, but relevant, it should be high, else learning will be slow
    • Regularization makes this better/worse?
  – But then you could overshoot the local minima when you train
Motivation

• What’s the best learning rate?
  – Depends on typical gradient for a feature
    • Small $\rightarrow$ fast rate
    • Large $\rightarrow$ slow rate
  – Sadly we can’t afford to ignore rare features
    • We could have a lot of them
Motivation

• What’s the best learning rate?
  – Let’s pretend our observed gradients are from a zero-mean Gaussian and find variances, then scale dimension $j$ by $sd(j)^{-1}$
Motivation

• What’s the best learning rate?
  – Let’s pretend our observed gradients are from a zero-mean Gaussian and find variances, then scale dimension $j$ by $\text{sd}(j)^{-1}$
  – Ignore co-variances for efficiency
Motivation

• What’s the best learning rate?
  – Let’s pretend our observed gradients are from a zero-mean Gaussian and find variances, then scale dimension $j$ by $sd(j)^{-1}$
  – Ignore co-variances for efficiency
Adagrad

\[ G = \sum_{\tau=1}^{t} g_\tau g_\tau^T \]

Gradient at time \( \tau \) - covariances

\[
w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j.
\]

\[
:= W_j - \frac{\eta g_j}{\sum_{\tau} g_{\tau,j}^2}
\]

\( \eta = 1 \) is usually ok
\[ g_\tau = \nabla Q_i(w) \]

\[ w := w - \eta \text{diag}(G)^{-\frac{1}{2}} \circ g \]

\[ G = \sum_{\tau=1}^{t} g_\tau g_\tau^\top \]

\[ \sqrt{G_i} = \sqrt{\sum_{\tau=1}^{t} g_\tau^2} \]

\[ w := w - \eta \text{diag}(G)^{-\frac{1}{2}} \circ g \]

\[ w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j. \]
ALL-REDUCE
Introduction

• Common pattern:
  – do some learning in parallel
  – aggregate local changes from each processor
    • to shared parameters
  – distribute the new shared parameters
    • back to each processor

  – and repeat….  

• AllReduce implemented in MPI, recently in VW code (John Langford) in a Hadoop/compatible scheme
Allreduce initial state

5  7  6
1  2  3  4

Allreduce final state

28  28  28
28  28  28  28  28
Create Binary Tree

7

5
1 2

6
3 4
Reducing, step 1

7

8

1 2

13

3 4
Reducing, step 2

28

8

1

2

13

3

4
Broadcast, step 1

```
  28
 /   \
28    28
|     |
1     2
  |       |
  3       4
```
Allreduce final state

AllReduce = Reduce + Broadcast
Gory details of VW Hadoop-AllReduce

• Spanning-tree server:
  – Separate process constructs a spanning tree of the compute nodes in the cluster and then acts as a server

• Worker nodes (“fake” mappers):
  – Input for worker is locally cached
  – Workers all connect to spanning-tree server
  – Workers all execute the same code, which might contain AllReduce calls:
    • Workers synchronize whenever they reach an all-reduce
“Map” job moves program to data.

Delayed initialization: Most failures are disk failures. First read (and cache) all data, before initializing allreduce. Failures auto-restart on different node with identical data. don’t wait for duplicate job

Speculative execution: In a busy cluster, one node is often slow. Hadoop can speculatively start additional mappers. We use the first to finish reading all data once.
1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online, gradient descent.
   2. L-BFGS - Second-order method - like Newton’s method
   3. Use (1) to warmstart (2).

2. Use map-only Hadoop for process control and error recovery.

3. Use AllReduce code to sync state.

4. Always save input examples in a cachefile to speed later passes.

5. Use hashing trick to reduce input complexity.

Open source in Vowpal Wabbit 6.1. Search for it.
$2^{24}$ features

$\approx 100$ non-zeros/example

2.3B examples

example is user/page/ad and conjunctions of these, positive if there was a click-thru on the ad

Figure 2: Speed-up for obtaining a fixed test error, on the display advertising problem, relative to the run with 10 nodes, as a function of the number of nodes. The dashed corresponds to the ideal speed-up, the solid line is the average speed-up over 10 repetitions and the bars indicate maximum and minimal values.
Table 3: Computing time on the splice site recognition data with various number of nodes for obtaining a fixed test error. The first 3 rows are average per iteration (excluding the first one).

<table>
<thead>
<tr>
<th>Nodes</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm time / pass</td>
<td>5</td>
<td>12</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>Median comp time / pass</td>
<td>167</td>
<td>105</td>
<td>43</td>
<td>34</td>
</tr>
<tr>
<td>Max comp time / pass</td>
<td>462</td>
<td>271</td>
<td>172</td>
<td>95</td>
</tr>
<tr>
<td>Wall clock time</td>
<td>3677</td>
<td>2120</td>
<td>938</td>
<td>813</td>
</tr>
</tbody>
</table>

50M examples

explicitly constructed kernel ➔ 11.7M features

3,300 nonzeros/example

old method: SVM, 3 days: reporting time to get to fixed test error
Table 5: Average training time per iteration of an internal logistic regression implementation using either MapReduce or AllReduce for gradients aggregation. The dataset is the display advertising one and a subset of it.

<table>
<thead>
<tr>
<th></th>
<th>Full size</th>
<th>10% sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce</td>
<td>1690</td>
<td>1322</td>
</tr>
<tr>
<td>AllReduce</td>
<td>670</td>
<td>59</td>
</tr>
</tbody>
</table>