"Now, consider the following: You were admitted to this robot asylum. Therefore, you must be a robot. Diagnosis complete."
—Dr. Perceptron to Fry[source]

Dr. Perceptron is the head doctor at the Hal Institute for Criminally Insane Robots. He was destroyed briefly by Roberto during his escape from the Institute, but was apparently fixed/rebuilt and returned to work for Bender's second stay.

In 3008, Dr. Perceptron was damaged during a group therapy session, but like his encounter with Roberto, was quickly repaired to continue his duties at the Institute.

**Appearances**  Edit

- *Insane in the Mainframe*
- *Bender's Game*

**Gender**  Male

**Species**  Robot

**Planet**  Earth

**Profession**  Doctor of Freudian Circuit Analysis

**First appearance**  *Insane in the Mainframe*

**Voiced by**  Maurice LaMarche

Quick review of Tuesday

• Learning as optimization
• Optimizing conditional log-likelihood $\Pr(y|\mathbf{x})$ with logistic regression
• Stochastic gradient descent for logistic regression
  – Stream multiple times (epochs) thru data
  – Keep model in memory
• L2-regularization
• Sparse/lazy L2 regularization
• The “hash trick”: allow feature collisions, use array indexed by hash code instead of hash table for parameters.
Quick look ahead

• Experiments with a hash-trick implementation of logistic regression
• Next question:
  – how do you parallelize SGD, or more generally, this kind of streaming algorithm?
  – each example affects the next prediction ➔ order matters ➔ parallelization changes the behavior
  – we will step back to perceptrons and then step forward to parallel perceptrons
Debugging Machine Learning Algorithms

William Cohen
Debugging for non-ML systems

• “If it compiles, ship it.”
Debugging for ML systems

1. It’s definitely *exactly* the algorithm you read about in that paper
2. It also compiles
3. It gets 87% accuracy on the author’s dataset
   – but he got 91%
   – so it’s not working?
   – or, your eval is wrong?
   – or, *his* eval is wrong?
Debugging for ML systems

1. It’s definitely *exactly* the algorithm you read about in that paper
2. It also compiles
3. It gets 97% accuracy on the author’s dataset
   – but he got 91%
   – so you have a best paper award!
   – or, maybe a bug...
Debugging for ML systems

• It’s always hard to debug software
• It’s especially hard for ML
  – a wide range of almost-correct modes for a program to be in
EMPIRICAL RESEARCH
It’s easy to make assumptions about puppies strapped to missiles, but good science requires testing.
Debugging advice

1. Write tests
2. For subtle problems, write tests
3. If you’re still not sure why it’s not working, write tests
4. If you get really stuck:
   - take a walk and come back to it in a hour
   - ask a friend
     • If s/he’s also in 10-605 s/he can still help as long as no notes are taken (my rules)
   - take a break and write some tests
Debugging ML systems

Write tests

– For a generative learner, write a generator and generate training/test data from the assumed distribution

  • Eg, for NB: use one small multinomial for pos examples, another one for neg examples, and a weighted coin for the class priors.

– The learner should (usually) recover the actual parameters of the generator

  • given enough data, modulo convexity, ...

– Test it on the weird cases (eg, uniform class priors, highly skewed multinomials)
Debugging ML systems

Write tests

– For a discriminative learner, similar trick...
– Also, use what you know: eg, for SGD
  • does taking one gradient step (on a sample task) lower the loss on the training data?
  • does it lower the loss *as expected*?
    – \((f(x)-f(x+d))/d\) should approximate \(f'(x)\)
  • does regularization work *as expected*?
    – large \(\mu\) ➔ smaller param values
  • record training set/test set loss
    – with and without regularization
Debugging ML systems

Compare to a “baseline” mathematically clean method vs scalable, efficient method

• lazy/sparse vs naïve regularizer
• hashed feature values vs hashtable feature values

• ...
ON-LINE ANALYSIS AND REGRET
On-line learning/regret analysis

• Optimization
  – is a great model of what you want to do
  – a less good model of what you have time to do

• Example:
  – How much to we lose when we replace gradient descent with SGD?
  – what if we can only approximate the local gradient?
  – what if the distribution changes over time?
  – ...

• One powerful analytic approach: online-learning aka regret analysis (~aka on-line optimization)
On-line learning

Train Data

Compute: $y_i = \text{sign}(v \cdot x_i)$

Get $y_i$ and make update to $v$

To detect interactions:

• increase/decrease $v_k$ only if we need to (for that example)
• otherwise, leave it unchanged

• We can be sensitive to duplication by stopping updates when we get better performance
On-line learning

Train Data

instance $x_i$ $\rightarrow$ $B$ $\rightarrow$ $\hat{y}_i = \text{sign}(v \cdot x_i)$

+1, -1: label $y_i$

If mistake: $v_{k+1} = v_k + \text{correction}$

To detect interactions:
• increase/decrease $v_k$ only if we need to (for that example)
• otherwise, leave it unchanged

• We can be sensitive to duplication by stopping updates when we get better performance
Theory: the prediction game

• Player A:
  – picks a “target concept” $c$
    • for now - from a finite set of possibilities $C$ (e.g., all decision trees of size $m$)
  – for $t=1,\ldots,$
    • Player A picks $x=(x_1,\ldots,x_n)$ and sends it to $B$
      – For now, from a finite set of possibilities (e.g., all binary vectors of length $n$)
    • B predicts a label, $\hat{y}$, and sends it to A
    • A sends B the true label $y=c(x)$
    • we record if B made a mistake or not
  – We care about the worst case number of mistakes $B$ will make over all possible concept & training sequences of any length
    • The “Mistake bound” for B, $M_B(C)$, is this bound
Perceptrons
The prediction game

• Are there practical algorithms where we can compute the mistake bound?
The voted perceptron

instance $x_i$

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

If mistake: \( v_{k+1} = v_k + y_i x_i \)

Margin \( \gamma \). A must provide examples that can be separated with some vector $u$ with margin $\gamma > 0$, ie

\[ \exists u : \forall (x_i, y_i) \text{ given by } A, (u \cdot x)y_i > \gamma \]

and furthermore, $\|u\| = 1$.

Radius $R$. A must provide examples “near the origin”, ie

\[ \forall x_i \text{ given by } A, \|x\|^2 < R^2 \]
The voted perceptron

Compute: \( p = \text{sign}(v_k \cdot x_i) \)

If mistake: \( v_{k+1} = v_k + y_i x_i \)

Aside: this is related to the SGD update:

\[
\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda (y - p) \mathbf{x}
\]

- \( y=p \): no update
- \( y=0, p=1 \): -\( \mathbf{x} \)
- \( y=1, p=0 \): +\( \mathbf{x} \)
$u - u \gamma + x_1 v_1$ (1) A target $u$

$u - u \gamma + x_2 v_2$ (2) The guess $v_1$ after one positive example.

$u - u \gamma + x_1 v_1$ $+ x_2 v_2$ (3a) The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

$u - u \gamma + x_1 v_1$ $- x_2 v_2$ (3b) The guess $v_2$ after the one positive and one negative example: $v_2 = v_1 - x_2$

If mistake: $v_{k+1} = v_k + y_i x_i$
(3a) The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

(3b) The guess $v_2$ after the one positive and one negative example: $v_2 = v_1 - x_2$

Lemma 1 $\forall k$, $v_k \cdot u \geq k\gamma$. In other words, the dot product between $v_k$ and $u$ increases with each mistake, at a rate depending on the margin $\gamma$.

Proof:

$v_{k+1} \cdot u = (v_k + y_i x_i) \cdot u$

$\Rightarrow v_{k+1} \cdot u = (v_k \cdot u) + y_i (x_i \cdot u)$

$\Rightarrow v_{k+1} \cdot u \geq v_k \cdot u + \gamma$

$\Rightarrow v_k \cdot u \geq k\gamma$
(3a) The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

(3b) The guess $v_2$ after the one positive and one negative example: $v_2 = v_1 - x_2$

Lemma 2 $\forall k, \|v_k\|^2 \leq kR^2$. In other words, the norm of $v_k$ grows “slowly”, at a rate depending on $R^2$.

Proof:

$$v_{k+1} \cdot v_{k+1} = (v_k + y_i x_i) \cdot (v_k + y_i x_i)$$
$$\Rightarrow \quad \|v_{k+1}\|^2 = \|v_k\|^2 + 2y_i x_i \cdot v_k + y_i^2 \|x_i\|^2$$
$$\Rightarrow \quad \|v_{k+1}\|^2 = \|v_k\|^2 + [\text{something negative}] + 1\|x_i\|^2$$
$$\Rightarrow \quad \|v_{k+1}\|^2 \leq \|v_k\|^2 + \|x\|^2$$
$$\Rightarrow \quad \|v_{k+1}\|^2 \leq \|v_k\|^2 + R^2$$
$$\Rightarrow \quad \|v_k\|^2 \leq kR^2$$
Lemma 1 \( \forall k, \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma \). In other words, the dot product between \( \mathbf{v}_k \) and \( \mathbf{u} \) increases with each mistake, at a rate depending on the margin \( \gamma \).

Lemma 2 \( \forall k, \|\mathbf{v}_k\|^2 \leq kR \). In other words, the norm of \( \mathbf{v}_k \) grows “slowly”, at a rate depending on \( R \).

\[
\begin{align*}
(k\gamma)^2 &\leq (\mathbf{v}_k \cdot \mathbf{u})^2 \\
\Rightarrow \quad k^2\gamma^2 &\leq \|\mathbf{v}_k\|^2 \|\mathbf{u}\|^2 \\
\Rightarrow \quad k^2\gamma^2 &\leq \|\mathbf{v}_k\|^2 \\
&\quad k^2\gamma^2 \leq \|\mathbf{v}_k\|^2 \leq kR^2 \\
&\quad \Rightarrow \quad k^2\gamma^2 \leq kR^2 \\
&\quad \Rightarrow \quad k\gamma^2 \leq R^2 \\
&\quad \Rightarrow \quad k \leq \frac{R^2}{\gamma^2} = \left(\frac{R}{\gamma}\right)^2
\end{align*}
\]

Radius \( R \). A must provide examples “near the origin”, ie

\[\forall x_i \text{ given by } A, \|x\|^2 < R^2\]
One Weird Trick for Making Perceptrons More Expressive

What if the separating line doesn’t go thru the origin?

Replace \( \mathbf{x} = (x_1, \ldots, x^n) \) with \( (x^0, \ldots, x^n) \) where \( x^0 = 1 \) for every example \( \mathbf{x} \).

Then \( y = \text{sign}(\sum_j x^j w^j) \) becomes

\[
\text{sign}(x^0 w^0 + \sum_{j \geq 1} x^j w^j)
\]

which is

\[
\text{sign} \left( w^0 + \sum_{j \geq 1} x^j w^j \right)
\]
Summary

• We have shown that
  – *If* : exists a $u$ with unit norm that has margin $\gamma$ on examples in the seq $(x_1,y_1),(x_2,y_2),\ldots$.  
  – *Then* : the perceptron algorithm makes $< R^2/ \gamma^2$ mistakes on the sequence (where $R \geq ||x_i||$)  
  – *Independent* of dimension of the data or classifier (!)  
  – This doesn’t follow from $M(C) \leq VCDim(C)$

• We *don’t* know if this algorithm could be better  
  – There are many variants that rely on similar analysis (ROMMA, Passive-Aggressive, MIRA, …)

• We *don’t* know what happens if the data’s not separable  
  – Unless I explain the “Δ trick” to you

• We *don’t* know what classifier to use “after” training
The idea of the “delta trick”

A noisy example makes the data inseparable
The idea of the “delta trick”

So let’s add a new dimension and give the noisy example an offset in the dimension of $\Delta$

I don’t know which examples are noisy….but I have lots of dimensions that I could add to the data…. Now it’s separable!
The $\Delta$ Trick

- The proof assumes the data is separable by a wide margin
- We can *make* that true by adding an “id” feature to each example
  - sort of like we added a constant feature

\[
\begin{align*}
\mathbf{x}^1 &= (x_1^1, x_2^1, \ldots, x_m^1) \rightarrow (x_1^1, x_2^1, \ldots, x_m^1, \Delta, 0, \ldots, 0) \\
\mathbf{x}^2 &= (x_1^2, x_2^2, \ldots, x_m^2) \rightarrow (x_1^2, x_2^2, \ldots, x_m^2, 0, \Delta, \ldots, 0) \\
\mathbf{x}^n &= (x_1^n, x_2^n, \ldots, x_m^n) \rightarrow (x_1^n, x_2^n, \ldots, x_m^n, 0, 0, \ldots, \Delta)
\end{align*}
\]
The $\Delta$ Trick

• The proof assumes the data is separable by a wide margin
• We can *make* that true by adding an “id” feature to each example
  – sort of like we added a constant feature

\[ \text{doc17: } \text{i, found, aardvark, today} \rightarrow \text{i, found, aardvark, today, doc17} \]
\[ \text{doc37: } \text{aardvarks, are, dangerous} \rightarrow \text{aardvarks, are, dangerous, doc37} \]
\[ \ldots \]
The Δ Trick

- Replace $x_i$ with $x_i'$ so $X$ becomes $[X | I \Delta]$
- Replace $R^2$ in our bounds with $R^2 + \Delta^2$
- Let $d_i = \max(0, \gamma - y_i x_i u)$
- Let $u' = (u_1, \ldots, u_n, y_1 d_1 / \Delta, \ldots, y_m d_m / \Delta) \times 1/Z$
  - So $Z = \sqrt{1 + D^2 / \Delta^2}$, for $D = \sqrt{d_1^2 + \ldots + d_m^2}$
  - Now $[X | I \Delta]$ is separable by $u'$ with margin $\gamma$
- Mistake bound is $(R^2 + \Delta^2)Z^2 / \gamma^2$
- Let $\Delta = \sqrt{RD} \Rightarrow k \leq ((R + D) / \gamma)^2$
- Conclusion: a little noise is ok
Summary

• We have shown that
  – *If* : exists a $\mathbf{u}$ with unit norm that has margin $\gamma$ on examples in the seq $(\mathbf{x}_1,y_1), (\mathbf{x}_2,y_2), \ldots$
  – *Then* : the perceptron algorithm makes $< \frac{R^2}{\gamma^2}$ mistakes on the sequence (where $R \geq ||\mathbf{x}_i||$)
    – *Independent* of dimension of the data or classifier (!)
• We *don’t* know what happens if the data’s not separable
  – Unless I explain the “Δ trick” to you
• We *don’t* know what classifier to use “after” training
The averaged perceptron
On-line to batch learning

1. Pick a $v_k$ at random according to $m_k/m$, the fraction of examples it was used for.

2. Predict using the $v_k$ you just picked.

3. (Actually, use some sort of deterministic approximation to this).

\[
P(\text{error in } x) = \sum_k P(\text{error on } x|\text{picked } v_k)P(\text{picked } v_k)
\]

\[
= \sum_k \frac{1}{m_k/m} = \sum_k \frac{1}{m} = \frac{k}{m}
\]

Imagine we run the on-line perceptron and see this result.

<table>
<thead>
<tr>
<th>$i$</th>
<th>guess</th>
<th>input</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_0$</td>
<td>$x_1$</td>
<td>$X$ (a mistake)</td>
</tr>
<tr>
<td>2</td>
<td>$v_1$</td>
<td>$x_2$</td>
<td>$\checkmark$ (correct!)</td>
</tr>
<tr>
<td>3</td>
<td>$v_1$</td>
<td>$x_3$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>4</td>
<td>$v_1$</td>
<td>$x_4$</td>
<td>$X$ (a mistake)</td>
</tr>
<tr>
<td>5</td>
<td>$v_2$</td>
<td>$x_5$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>6</td>
<td>$v_2$</td>
<td>$x_6$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>7</td>
<td>$v_2$</td>
<td>$x_7$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>8</td>
<td>$v_2$</td>
<td>$x_8$</td>
<td>$X$</td>
</tr>
<tr>
<td>9</td>
<td>$v_3$</td>
<td>$x_9$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>10</td>
<td>$v_3$</td>
<td>$x_{10}$</td>
<td>$X$</td>
</tr>
</tbody>
</table>
1. Pick a $v_k$ at random according to $m_k/m$, the fraction of examples it was used for.

2. Predict using the $v_k$ you just picked.

3. (Actually, use some sort of deterministic approximation to this).

<table>
<thead>
<tr>
<th>i</th>
<th>guess</th>
<th>input</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_0$</td>
<td>$x_1$</td>
<td>X (a mistake)</td>
</tr>
<tr>
<td>2</td>
<td>$v_1$</td>
<td>$x_2$</td>
<td>✓  (correct!)</td>
</tr>
<tr>
<td>3</td>
<td>$v_1$</td>
<td>$x_3$</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>$v_1$</td>
<td>$x_4$</td>
<td>X (a mistake)</td>
</tr>
<tr>
<td>5</td>
<td>$v_2$</td>
<td>$x_5$</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>$v_2$</td>
<td>$x_6$</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>$v_2$</td>
<td>$x_7$</td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>$v_2$</td>
<td>$x_8$</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>$v_3$</td>
<td>$x_9$</td>
<td>✓</td>
</tr>
<tr>
<td>10</td>
<td>$v_3$</td>
<td>$x_{10}$</td>
<td>X</td>
</tr>
</tbody>
</table>
SPARSIFYING THE AVERAGED PERCEPTRON UPDATE
Complexity of perceptron learning

- **Algorithm:** \( O(n) \)
- \( v = 0 \)
- for each example \( x, y \):
  - if \( \text{sign}(v \cdot x) \neq y \)
    - \( v = v + yx \) \( O(|x|) = O(|d|) \)
  - for \( x_i \neq 0 \), \( v_i += yx_i \)

Final hypothesis (last): \( v \)
Complexity of *averaged* perceptron

- **Algorithm:** \(\Theta(n) \quad O(n|V|)\)
- **\(v_k=0\)**
- **\(v_a = 0\)**
- for each example \(x, y\):
  - if \(\text{sign}(v_k.x) \neq y\)
    - \(v_a = v_a + mk * v_k\)
    - \(v_k = v_k + yx\)
    - \(m = m + 1\)
    - \(mk = 1\)
  - else
    - \(mk++\)

Final hypothesis (avg): \(\frac{v_a}{m}\)
Complexity of perceptron learning

- Algorithm: $O(n)$
- $v = 0$
- for each example $x, y$:
  - if $\text{sign}(v \cdot x) \neq y$
    - $v = v + yx$
    - $O(|x|) = O(|d|)$
- init hashtable
- for $x_i \neq 0$, $v_i += yx_i$
Complexity of *averaged* perceptron

- Algorithm: $O(n)$ $O(n|V|)$
- $vk=0$
- $va = 0$
- for each example $x,y$:  
  - if $\text{sign}(vk.x) \neq y$  
    - $va = va + vk$  
    - $vk = vk + yx$
    - $mk = 1$ $O(|x|)=O(|d|)$
  - else
    - $nk++$  
    - for $vk_i!=0$, $va_i += vk_i$
    - for $x_i!=0$, $v_i += yx_i$
**Alternative averaged perceptron**

- Algorithm:
  - $v_k = 0$
  - $v_a = 0$
  - for each example $x, y$:
    - $v_a = v_a + v_k$
    - $m = m + 1$
    - if $\text{sign}(v_k.x) \neq y$
      - $v_k = v_k + y^*x$
  - Return $v_a/m$

Observe:

$$v_k = \sum_{j \in S_k} y_j x_j$$

$S_k$ is the set of examples including the first $k$ mistakes.
Alternative averaged perceptron

• Algorithm:
  • \( v_k = 0 \)
  • \( v_a = 0 \)
  • for each example \( x,y \):
    - \( v_a = v_a + \sum_{j \in S_k} y_j x_j \)
    - \( m = m + 1 \)
    - if \( \text{sign}(v_k \cdot x) \neq y \)
      • \( v_k = v_k + y^*x \)
  • Return \( v_a/m \)

So when there’s a mistake at time \( t \) on \( x,y \):

\( y^*x \) is added to \( v_a \) on every subsequent iteration

Suppose you know \( T \), the total number of examples in the stream…
Alternative averaged perceptron

- Algorithm:
  - $v_k = 0$
  - $v_a = 0$
  - for each example $x,y$:
    - $v_a = v_a + \sum_{j \in S_k} y_j x_j$
    - $m = m + 1$
    - if $\text{sign}(v_k \cdot x) \neq y$
      - $v_k = v_k + y^*x$
      - $v_a = v_a + (T-m)^*y^*x$
  - Return $v_a/T$

$T$ = the total number of examples in the stream…(all epochs)

Unpublished? I figured this out recently, Leon Bottou knows it too
KERNELS AND PERCEPTRONS
The kernel perceptron

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

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

- 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

Explicitly map from $x$ to $\phi(x)$ – i.e. to the point corresponding to $x$ in the Hilbert space

$$\hat{y} = x \cdot w = K(x, w)$$

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

$$\hat{y} = \phi(x) \cdot w$$

$$w = \sum_{x_{k^+} \in FN} \phi(x_{k^+}) - \sum_{x_{k^-} \in FP} \phi(x_{k^-})$$

Implicitly map from $x$ to $\phi(x)$ by changing the kernel function $K$

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

$$K(x, x_k) \equiv \phi(x) \cdot \phi(x_k)$$

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

$$K(x, x_k) \equiv \phi(x') \cdot \phi(x'_{k})$$

Two different computational ways of getting the same behavior
Kernels 101

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

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

$K(x, x) > 0 \Rightarrow$ diagonal of $K$ is positive $\Rightarrow$ $K$ is “positive semi-definite” $\Rightarrow$ $z^T K z \geq 0$ for all $z$

$$
K =
\begin{bmatrix}
K(1,1) & K(1,2) & K(1,3) & \cdots & K(1,m) \\
K(2,1) & K(2,2) & K(2,3) & \cdots & K(2,m) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
K(m,1) & K(m,2) & K(m,3) & \cdots & K(m,m)
\end{bmatrix}
$$
Review: the hash trick
Learning as optimization for regularized logistic regression

• Algorithm:

\[ w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j \]

• Initialize hashtables \( W, A \) and set \( k=0 \)
• For each iteration \( t=1,...,T \)
  – For each example \((x_i,y_i)\)
    • \( p_i = ... ; k++ \)
    • For each feature \( j: x_i^j > 0: \)
      » \( W[j] = W[j] + \lambda(y_i - p^i)x_j \)
      » \( A[j] = k \)
Learning as optimization for regularized logistic regression

- **Algorithm:**

  \[ w^j = w^j + \lambda (y - p)x^j - \lambda 2\mu w^j \]

- Initialize arrays \( W, A \) of size \( R \) and set \( k=0 \)

- For each iteration \( t=1, \ldots, T \)
  - For each example \((x_i, y_i)\)
    - Let \( V \) be hash table so that \( V[h] = \sum_{j : \text{hash}(x^j_i) \% R = h} x^j_i \)
    - \( p_i = \ldots ; k++ \)
  - For each hash value \( h: V[h] > 0 \):
    - \( W[h] \ast = (1 - \lambda 2\mu)^{k - A[j]} \)
    - \( W[h] = W[h] + \lambda (y_i - p^j) V[h] \)
    - \( A[h] = k \)
The hash trick as a kernel
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
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

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

**Lemma 2** The hash kernel is unbiased, that is $$\mathbb{E}_\varphi[\langle x, x' \rangle_\varphi] = \langle x, x' \rangle$$. Moreover, the variance is $$\sigma^2_{x,x'} = \frac{1}{m} \left( \sum_{i\neq j} x_i^2 x_j'^2 + x_i x_i' x_j x_j' \right)$$, and thus, for $$\|x\|_2 = \|x'\|_2 = 1$$, $$\sigma^2_{x,x'} = O \left( \frac{1}{m} \right)$$.

$m$ 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

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

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

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

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 unhashed 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 $h \rightarrow w_1, w_2, \ldots$,
Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (global-hashed) converges relatively soon, showing that the distortion error $\epsilon_d$ vanishes. The personalized classifier results in an average improvement of up to 30%.