Learning with Large Datasets

Léon Bottou

NEC Laboratories America
Why Large-scale Datasets?

- **Data Mining**
  
  Gain competitive advantages by analyzing data that describes the life of our computerized society.

- **Artificial Intelligence**
  
  Emulate cognitive capabilities of humans. Humans learn from abundant and diverse data.
The Computerized Society Metaphor

• A society with just two kinds of computers:

  **Makers** do business and generate leftward revenue. They also produce data in proportion with their activity.

  **Thinkers** analyze the data to increase revenue by finding rightward competitive advantages.

• When the population of computers grows:

  – The ratio #Thinkers/#Makers must remain bounded.
  – The **Data** grows with the number of **Makers**.
  – The number of **Thinkers** does not grow faster than the **Data**.
Limited Computing Resources

- The computing resources available for learning do not grow faster than the volume of data.
  - The cost of data mining cannot exceed the revenues.
  - Intelligent animals learn from streaming data.

- Most machine learning algorithms demand resources that grow faster than the volume of data.
  - Matrix operations ($n^3$ time for $n^2$ coefficients).
  - Sparse matrix operations are worse.
Roadmap

I. Statistical Efficiency versus Computational Cost.

II. Stochastic Algorithms.

III. Learning with a Single Pass over the Examples.

IV. Selecting Useful Training Examples
Counter-Roadmap

• This presentation does not focus on:
  – Data quality issues (e.g., missing, inconsistent, unlabelled.)
  – Data mining specifics (e.g., data storage and access.)
  – Implementation (e.g., software and hardware.)

• Yet these topics will make appearances.
Part I

Statistical Efficiency versus Computational Costs.
Objectives and Essential Remarks

• Baseline large-scale learning algorithm

  Randomly discarding data is the simplest way to handle large datasets.
  – What are the statistical benefits of processing more data?
  – What is the computational cost of processing more data?

• Sublinear learning algorithms with independent examples.

  A learning algorithm for independent examples must
  – either check every example (\(\geq\) linear computation time)
  – or choose a priori to disregard some random examples.

  Sublinear learning algorithms are no better than baseline.
Learning Algorithms: Standard Framework

- Assumption: examples are drawn independently from an unknown probability distribution $P(x, y)$ that represents the rules of Nature.
- Expected Risk: $E(f) = \int \ell(f(x), y) dP(x, y)$.
- Empirical Risk: $E_n(f) = \frac{1}{n} \sum \ell(f(x_i), y_i)$.
- We would like $f^*$ that minimizes $E(f)$ among all functions.
- In general $f^* \notin \mathcal{F}$.
- The best we can have is $f^*_F \in \mathcal{F}$ that minimizes $E(f)$ inside $\mathcal{F}$.
- But $P(x, y)$ is unknown by definition.
- Instead we compute $f_n \in \mathcal{F}$ that minimizes $E_n(f)$. Vapnik-Chervonenkis theory tells us when this can work.
Learning with Approximate Optimization

Computing $f_n = \arg \min_{f \in \mathcal{F}} E_n(f)$ is often costly.

Since we already make lots of approximations, why should we compute $f_n$ exactly?

Let’s assume our optimizer returns $\tilde{f}_n$ such that $E_n(\tilde{f}_n) < E_n(f_n) + \rho$.

For instance, one could stop an iterative optimization algorithm long before its convergence.
Decomposition of the Error (i)

\[ E(\tilde{f}_n) - E(f^*) = E(f^*_\mathcal{F}) - E(f^*) \]
\[ + E(f_n) - E(f^*_\mathcal{F}) \]
\[ + E(\tilde{f}_n) - E(f_n) \]

Approximation error
Estimation error
Optimization error

Problem:
Choose \( \mathcal{F} \), \( n \), and \( \rho \) to make this as small as possible,

subject to budget constraints \( \left\{ \begin{array}{l}
\text{maximal number of examples } n \\
\text{maximal computing time } T
\end{array} \right. \)
Decomposition of the Error (ii)

Approximation error bound: (Approximation theory)
- decreases when $\mathcal{F}$ gets larger.

Estimation error bound: (Vapnik-Chervonenkis theory)
- decreases when $n$ gets larger.
- increases when $\mathcal{F}$ gets larger.

Optimization error bound: (Vapnik-Chervonenkis theory plus tricks)
- increases with $\rho$.

Computing time $T$: (Algorithm dependent)
- decreases with $\rho$
- increases with $n$
- increases with $\mathcal{F}$
Small-scale vs. Large-scale Learning

- **Definition 1:**
  We have a small-scale learning problem when the active budget constraint is the number of examples $n$.

- **Definition 2:**
  We have a large-scale learning problem when the active budget constraint is the computing time $T$. 
The active budget constraint is the number of examples.

- To reduce the estimation error, take \( n \) as large as the budget allows.
- To reduce the optimization error to zero, take \( \rho = 0 \).
- We need to adjust the size of \( \mathcal{F} \).

See Structural Risk Minimization (Vapnik 74) and later works.
Large-scale Learning

The active budget constraint is the computing time.

- More complicated tradeoffs.
  The computing time depends on the three variables: $F$, $n$, and $\rho$.

- Example.
  If we choose $\rho$ small, we decrease the optimization error. But we must also decrease $F$ and/or $n$ with adverse effects on the estimation and approximation errors.

- The exact tradeoff depends on the optimization algorithm.

- We can compare optimization algorithms rigorously.
Learning versus Optimization

\[ \log(\rho) \]

\[ \log(T) \]

\( \rho \) decreases faster than \( \exp(-T) \)

\( \rho \) decreases like \( 1/T \)

Extraordinary poor optimization algorithm

Good optimization algorithm (superlinear).

Mediocre optimization algorithm (linear).

Best \( \rho \)?
Asymptotics: Estimation

Uniform convergence bounds (with capacity $d + 1$)

Estimation error $\leq \mathcal{O}\left(\left[\frac{d}{n} \log \frac{n}{d}\right]^\alpha\right)$ with $\frac{1}{2} \leq \alpha \leq 1$.

There are in fact three types of bounds to consider:

- Classical V-C bounds (pessimistic): $\mathcal{O}\left(\sqrt{\frac{d}{n}}\right)$
- Relative V-C bounds in the realizable case: $\mathcal{O}\left(\frac{d}{n} \log \frac{n}{d}\right)$
- Localized bounds (variance, Tsybakov): $\mathcal{O}\left(\left[\frac{d}{n} \log \frac{n}{d}\right]^\alpha\right)$

Fast estimation rates are a big theoretical topic these days.
**Asymptotics: Estimation + Optimization**

Uniform convergence arguments give

\[
\text{Estimation error + Optimization error} \leq \mathcal{O}\left(\left[\frac{d}{n \log \frac{n}{d}}\right]^\alpha + \rho\right).
\]

This is true for all three cases of uniform convergence bounds.

**Scaling laws for \(\rho\) when \(\mathcal{F}\) is fixed**

The approximation error is constant.

- No need to choose \(\rho\) smaller than \(\mathcal{O}\left(\left[\frac{d}{n \log \frac{n}{d}}\right]^\alpha\right)\).
- Not advisable to choose \(\rho\) larger than \(\mathcal{O}\left(\left[\frac{d}{n \log \frac{n}{d}}\right]^\alpha\right)\).
When $\mathcal{F}$ is chosen via a $\lambda$-regularized cost

- Uniform convergence theory provides bounds for simple cases
  (Massart-2000; Zhang 2005; Steinwart et al., 2004-2007; ...)
- Computing time depends on both $\lambda$ and $\rho$.
- Scaling laws for $\lambda$ and $\rho$ depend on the optimization algorithm.

When $\mathcal{F}$ is realistically complicated

Large datasets matter
- because one can use more features,
- because one can use richer models.

Bounds for such cases are rarely realistic enough.

Luckily there are interesting things to say for $\mathcal{F}$ fixed.
Case Study

Simple parametric setup

- $\mathcal{F}$ is fixed.
- Functions $f_w(x)$ parametrized by $w \in \mathbb{R}^d$.

Comparing four iterative optimization algorithms for $E_n(f)$

1. Gradient descent.
2. Second order gradient descent (Newton).
4. Stochastic second order gradient descent.
Quantities of Interest

- Empirical Hessian at the empirical optimum $w_n$. 

$$H = \frac{\partial^2 E_n}{\partial w^2}(w_n) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial^2 \ell(f_n(x_i), y_i)}{\partial w^2}$$

- Empirical Fisher Information matrix at the empirical optimum $w_n$. 

$$G = \frac{1}{n} \sum_{i=1}^{n} \left[ \left( \frac{\partial \ell(f_n(x_i), y_i)}{\partial w} \right) \left( \frac{\partial \ell(f_n(x_i), y_i)}{\partial w} \right)^T \right]$$

- **Condition number**

  We assume that there are $\lambda_{\text{min}}$, $\lambda_{\text{max}}$ and $\nu$ such that
  
  - trace($GH^{-1}$) $\approx \nu$.
  - spectrum($H$) $\subset [\lambda_{\text{min}}, \lambda_{\text{max}}]$.

  and we define the condition number $\kappa = \lambda_{\text{max}}/\lambda_{\text{min}}$. 
Gradient Descent (GD)

Iterate

\[ w_{t+1} \leftarrow w_t - \eta \frac{\partial E_n(f_{w_t})}{\partial w} \]

Best speed achieved with fixed learning rate \( \eta = \frac{1}{\lambda_{\text{max}}} \).

(e.g., Dennis & Schnabel, 1983)

<table>
<thead>
<tr>
<th>Cost per iteration</th>
<th>Iterations to reach ( \rho )</th>
<th>Time to reach accuracy ( \rho )</th>
<th>Time to reach ( E(\tilde{f}_n) - E(f^*_F) &lt; \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD ( O(nd) )</td>
<td>( O\left(\kappa \log \frac{1}{\rho}\right) )</td>
<td>( O\left(nd\kappa \log \frac{1}{\rho}\right) )</td>
<td>( O\left(\frac{d^2 \kappa}{\varepsilon^{1/\alpha}} \log^2 \frac{1}{\varepsilon}\right) )</td>
</tr>
</tbody>
</table>

- In the last column, \( n \) and \( \rho \) are chosen to reach \( \varepsilon \) as fast as possible.
- Solve for \( \varepsilon \) to find the best error rate achievable in a given time.
- Remark: abuses of the \( O() \) notation
Second Order Gradient Descent (2GD)

Iterate

- \( w_{t+1} \leftarrow w_t - H^{-1} \frac{\partial E_n(f_{w_t})}{\partial w} \)

We assume \( H^{-1} \) is known in advance.
Superlinear optimization speed (e.g., Dennis & Schnabel, 1983)

<table>
<thead>
<tr>
<th>Cost per iteration</th>
<th>Iterations to reach ( \rho )</th>
<th>Time to reach accuracy ( \rho )</th>
<th>Time to reach ( E(\tilde{f}<em>n) - E(f^*</em>\mathcal{F}) &lt; \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{O}(d(d+n)) )</td>
<td>( \mathcal{O}\left(\log \log \frac{1}{\rho}\right) )</td>
<td>( \mathcal{O}\left(d(d+n) \log \log \frac{1}{\rho}\right) )</td>
<td>( \mathcal{O}\left(\frac{d^2}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon}\right) )</td>
</tr>
</tbody>
</table>

- Optimization speed is much faster.
- Learning speed only saves the condition number \( \kappa \).
Stochastic Gradient Descent (SGD)

Iterate

• Draw random example \((x_t, y_t)\).

• \(w_{t+1} \leftarrow w_t - \eta \frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}\)

Best decreasing gain schedule with \(\eta = \frac{1}{\lambda_{\text{min}}}\).
(see Murata, 1998; Bottou & LeCun, 2004)

<table>
<thead>
<tr>
<th>Cost per iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (E(\tilde{f}_n) - E(f^*_F) &lt; \varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>(O(d))</td>
<td>(\frac{\nu k}{\rho} + o\left(\frac{1}{\rho}\right))</td>
<td>(O\left(\frac{d \nu k}{\rho}\right))</td>
</tr>
</tbody>
</table>

With \(1 \leq k \leq \kappa^2\)

- Optimization speed is \textit{catastrophic}.
- Learning speed does not depend on the statistical estimation rate \(\alpha\).
- Learning speed depends on condition number \(\kappa\) but \textit{scales very well}. 

Total Gradient \(<J(x,y,w)>\)

Partial Gradient \(J(x,y,w)\)
Second order Stochastic Descent (2SGD)

Iterate

- Draw random example \((x_t, y_t)\).
- \(w_{t+1} \leftarrow w_t - \frac{1}{t} H^{-1} \frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}\)

Replace scalar gain \(\frac{\eta}{t}\) by matrix \(\frac{1}{t} H^{-1}\).

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (E(\tilde{f}_n) - E(f^*_F) &lt; \varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2SGD</strong></td>
<td>(O(d^2))</td>
<td>(\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right))</td>
<td>(O\left(\frac{d^2 \nu}{\rho}\right))</td>
<td>(O\left(\frac{d^2 \nu}{\varepsilon}\right))</td>
</tr>
</tbody>
</table>

- Each iteration is \(d\) times more expensive.
- The number of iterations is reduced by \(\kappa^2\) (or less.)
- Second order only changes the constant factors.
Part II

Learning with Stochastic Gradient Descent.
BENCHMARKING SGD IN SIMPLE PROBLEMS

• The theory suggests that SGD is very competitive.
  – Many people associate SGD with trouble.

• SGD historically associated with back-propagation.
  – Multilayer networks are very hard problems (nonlinear, nonconvex)
  – What is difficult, SGD or MLP?

• Try plain SGD on simple learning problems.
  – Support Vector Machines
  – Conditional Random Fields

The programs are very short because SGD is very simple.
Text Categorization with SVMs

• Dataset
  – Reuters RCV1 document corpus.
  – 781,265 training examples, 23,149 testing examples.
  – 47,152 TF-IDF features.

• Task
  – Recognizing documents of category CCAT.
  – Minimize $E_n = \frac{1}{n} \sum_i \left( \frac{\lambda}{2} w^2 + \ell (w x_i + b, y_i) \right)$.
  – Update $w \leftarrow w - \eta_t \nabla (w_t, x_t, y_t) = w - \eta_t \left( \lambda w + \frac{\partial \ell (w x_t + b, y_t)}{\partial w} \right)$

(see also: Shalev-Schwartz et al., ICML 2007)
Text Categorization with SVMs

- **Results: Linear SVM**
  \[ \ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\} \quad \lambda = 0.0001 \]

<table>
<thead>
<tr>
<th></th>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMLight</td>
<td>23,642 secs</td>
<td>0.2275</td>
<td>6.02%</td>
</tr>
<tr>
<td>SVMPerf</td>
<td>66 secs</td>
<td>0.2278</td>
<td>6.03%</td>
</tr>
<tr>
<td>SGD</td>
<td>1.4 secs</td>
<td>0.2275</td>
<td>6.02%</td>
</tr>
</tbody>
</table>

- **Results: Log-Loss Classifier**
  \[ \ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y})) \quad \lambda = 0.00001 \]

<table>
<thead>
<tr>
<th></th>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibLinear ((\varepsilon = 0.01))</td>
<td>30 secs</td>
<td>0.18907</td>
<td>5.68%</td>
</tr>
<tr>
<td>LibLinear ((\varepsilon = 0.001))</td>
<td>44 secs</td>
<td>0.18890</td>
<td>5.70%</td>
</tr>
<tr>
<td>SGD</td>
<td>2.3 secs</td>
<td>0.18893</td>
<td>5.66%</td>
</tr>
</tbody>
</table>
More SVM Experiments

From: Patrick Haffner
Date: Wednesday 2007-09-05 14:28:50

...I have tried on some of our main datasets...
I can send you the example, it is so striking!

Patrick

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Train size</th>
<th>Number of features</th>
<th>% non-0 features</th>
<th>LIBSVM (SDot)</th>
<th>LLAMA SVM</th>
<th>LLAMA MAXENT</th>
<th>SGDSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters</td>
<td>781K</td>
<td>47K</td>
<td>0.1%</td>
<td>210,000</td>
<td>3930</td>
<td>153</td>
<td>7</td>
</tr>
<tr>
<td>Translation</td>
<td>1000K</td>
<td>274K</td>
<td>0.0033%</td>
<td>days</td>
<td>47,700</td>
<td>1,105</td>
<td>7</td>
</tr>
<tr>
<td>SuperTag</td>
<td>950K</td>
<td>46K</td>
<td>0.0066%</td>
<td>31,650</td>
<td>905</td>
<td>210</td>
<td>1</td>
</tr>
<tr>
<td>Voicetone</td>
<td>579K</td>
<td>88K</td>
<td>0.019%</td>
<td>39,100</td>
<td>197</td>
<td>51</td>
<td>1</td>
</tr>
</tbody>
</table>
Text Chunking with CRFs

• Dataset
  – CONLL 2000 Chunking Task: Segment sentences in syntactically correlated chunks (e.g., noun phrases, verb phrases.)
  – 106,978 training segments in 8936 sentences.
  – 23,852 testing segments in 2012 sentences.

• Model
  – Conditional Random Field (all linear, log-loss.)
  – Features are $n$-grams of words and part-of-speech tags.
  – 1,679,700 parameters.

(see also: Vishwanathan et al., ICML 2006)
Text Chunking with CRFs

- **Results**

<table>
<thead>
<tr>
<th></th>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-BFGS</td>
<td>4335 secs</td>
<td>9042</td>
<td>93.74%</td>
</tr>
<tr>
<td>SGD</td>
<td>568 secs</td>
<td>9098</td>
<td>93.75%</td>
</tr>
</tbody>
</table>

- **Notes**

  - Computing the gradients with the chain rule runs faster than computing them with the forward-backward algorithm.

  - **Graph Transformer Networks** are nonlinear conditional random fields trained with stochastic gradient descent (Bottou et al., 1997).
Choosing the Gain Schedule

Decreasing gains: \[ w_{t+1} \leftarrow w_t - \frac{\eta}{t + t_0} \nabla (w_t, x_t, y_t) \]

- **Asymptotic Theory**
  - if \( s = 2 \eta \lambda_{\text{min}} < 1 \) then slow rate \( O(t^{-s}) \)
  - if \( s = 2 \eta \lambda_{\text{min}} > 1 \) then faster rate \( O\left(\frac{s^2}{s-1} t^{-1}\right) \)

- **Example: the SVM benchmark**
  - Use \( \eta = 1/\lambda \) because \( \lambda \leq \lambda_{\text{min}} \).
  - Choose \( t_0 \) to make sure that the expected initial updates are comparable with the expected size of the weights.

- **Example: the CRF benchmark**
  - Use \( \eta = 1/\lambda \) again.
  - Choose \( t_0 \) with the secret ingredient.
The Secret Ingredient for a good SGD

The sample size $n$ does not change the SGD maths!

Constant gain: \[ w_{t+1} \leftarrow w_t - \eta \nabla (w_t, x_t, y_t) \]

At any moment during training, we can:
- Select a small subsample of examples.
- Try various gains $\eta$ on the subsample.
- Pick the gain $\eta$ that most reduces the cost.
- Use it for the next 100000 iterations on the full dataset.

Examples
- The CRF benchmark code does this to choose $t_0$ before training.
- We could also perform such cheap measurements every so often. The selected gains would then decrease automatically.
The very simple SGD update offers lots of engineering opportunities.

Example: Sparse Linear SVM
The update $w \leftarrow w - \eta (\lambda w - \nabla \ell (wx_i, y_i))$
can be performed in two steps:

i) $w \leftarrow w - \eta \nabla \ell (wx_i, y_i)$ (sparse, cheap)
ii) $w \leftarrow w (1 - \eta \lambda)$ (not sparse, costly)

• Solution 1
Represent vector $w$ as the product of a scalar $s$ and a vector $v$.
Perform (i) by updating $v$ and (ii) by updating $s$.

• Solution 2
Perform only step (i) for each training example.
Perform step (ii) with lower frequency and higher gain.
SGD for Real Life Applications

A Check Reader

Examples are pairs \((\text{image}, \text{amount})\).

Problem with strong structure:
- Field segmentation
- Character segmentation
- Character recognition
- Syntactical interpretation.

- Define differentiable modules.
- Pretrain modules with hand-labelled data.
- Define global cost function (e.g., CRF).
- Train with SGD for a few weeks.

Industrially deployed in 1996. Ran billions of checks over 10 years.

Part III

Learning with a Single Pass over the Examples
Why learning with a Single Pass?

• **Motivation**
  
  – Sometimes there is too much data to store.
  – Sometimes retrieving archived data is too expensive.

• **Related Topics**
  
  – Streaming data.
  – Tracking nonstationarities.
  – Novelty detection.
Effect of one Additional Example (i)

Compare

\[ f_n = \text{arg min} \ E_n(f) \]

\[ f_{n+1} = \text{arg min} \ E_{n+1}(f) = \text{arg min} \left[ E_n(f) + \frac{1}{n} \ell(f(x_{n+1}), y_{n+1}) \right] \]
Effect of one Additional Example (ii)

- **First Order Calculation**

\[
    f_{n+1} = f_n - \frac{1}{n} H^{-1}_{n+1} \frac{\partial \ell(f_n(x_n), y_n)}{\partial w} + O\left(\frac{1}{n^2}\right)
\]

where \(H_{n+1}\) is the empirical Hessian on \(n + 1\) examples.

- **Compare with Second Order Stochastic Gradient Descent**

\[
    w_{t+1} = w_t - \frac{1}{t} H^{-1} \frac{\partial \ell(f_w(x_n), y_n)}{\partial w}
\]

- **Could they converge with the same speed?**
Yes they do! But what does it mean?

- **Theorem**  
  (Bottou & LeCun, 2003; Murata & Amari, 1998)

Under “adequate conditions”

\[
\lim_{n \to \infty} n \|f_\infty - f_n\|^2 = \lim_{t \to \infty} t \|w_\infty - w_t\|^2 = \text{tr}(H^{-1}G H^{-1})
\]

Best training set error.  

\[\cong F\]

Best solution in F.

One Pass of Second Order Stochastic Gradient

\[w_0 = f_0 \quad w_n \quad \cong K/n \quad f_F = f_\infty = w_\infty \quad \text{Best training set error.}\]
Given a large enough training set, a **Single Pass of Second Order Stochastic Gradient** generalizes as well as the Empirical Optimum.

Experiments on synthetic data

![Graph showing the relationship between number of examples and Mse* values for different error rates.](image)

- Number of examples
- Milliseconds

- Mse* +1e-1
- Mse* +1e-2
- Mse* +1e-3
- Mse* +1e-4
Unfortunate Practical Issues

• **Second Order SGD is not that fast!**

\[
\begin{align*}
w_{t+1} & \leftarrow w_t - \frac{1}{t} H^{-1} \frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}
\end{align*}
\]

– Must estimate and store \( d \times d \) matrix \( H^{-1} \).
– Must multiply the gradient for each example by the matrix \( H^{-1} \).
– Sparsity tricks no longer work because \( H^{-1} \) is not sparse.

• **Partial Workarounds**

  – Faster ways to compute \( H^{-1} \):
    · Woodbury’s formula (e.g., Amari et al., 2000)
    · Online BFGS (Schraudolph, 2007)

  – Limited storage approximations of \( H^{-1} \):
    · Diagonal approximation (Becker & LeCun, 1989)
    · Low rank approximation (e.g., LeCun et al., 1998)
    · Online L-BFGS approximation (Schraudolph, 2007)
## Summary

<table>
<thead>
<tr>
<th></th>
<th>2SGD</th>
<th>SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to reach accuracy ( \rho )</td>
<td>( \frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right) )</td>
<td>( \frac{k\nu}{\rho} + o\left(\frac{1}{\rho}\right) )</td>
</tr>
<tr>
<td>Number of epochs to reach same test cost as the full optimization.</td>
<td>1</td>
<td>( k )</td>
</tr>
</tbody>
</table>

There are many ways to make constant \( k \) smaller:
- Exact second order stochastic gradient descent.
- Approximate second order stochastic gradient descent.
- Simple preconditioning tricks.
Stopping criteria for SGD

- **Early stopping with cross validation**
  - Create a validation set by setting some training examples apart.
  - Monitor cost function on the validation set.
  - Stop when it stops decreasing.

- **Early stopping a priori**
  - Extract two disjoint subsamples of training data.
  - Train on the first subsample; stop by validating on the second.
  - The number of epochs is an estimate of $k$.
  - Train by performing that number of epochs on the full set.

This is asymptotically correct and gives reasonable results in practice.
Selecting Useful Training Examples.
Incremental Training

Add a fresh training example to the current training set.

Retrain until reaching sufficient accuracy.

Using a loss function with flat segments (e.g., hinge loss)
- Very few examples cause changes when added to the set.
- Very few examples cause changes during retraining.

Can we determine
- which examples to process, and
- which examples to store for further retraining?
Methods for Selecting Examples

- **Multiedit/Condense methods**
  - From the nearest neighbour literature (e.g., Devijver & Kittler, 82).
  - Can be adapted to other classifiers at a cost greater than $O(n^2)$.

- **Gradient methods**
  - Select examples when the loss has a sizeable gradient.
  - Many examples can carry the same information.
  - Leads to relatively costly orthogonalization schemes in $O(n^2)$, as in optimal experiment design. (e.g., Fedorov, 1971)

- **Duality in convex kernel machines**
  - Make the dual expression $w = \sum \alpha_i \Phi(x_i)$ sparse.
  - Recall the definition of support vectors.
  - Involves the bulky kernel matrix with $n \times n$ values.
  - Sparsity makes most of them irrelevant.
Large-scale Support Vector Machines?

An oxymoron and also an instructive challenge.

**Stochastic and Incremental SVMs**
- Iteratively constructing the expansion $\sum \alpha_i \Phi(x_i)$.
- Which candidate support vectors to store and discard?
- Managing the memory required by the kernel values.

**Active Learning in SVMs**
- Choose which example to process next.
- Convexity is no longer our friend.
Learning in the dual

- Convex, Kernel trick.
- Memory $nn_{SV}$
- Time $n^\alpha n_{SV}$ with $1 < \alpha \leq 2$
- Bad news $n_{SV} \sim 2\mathcal{B}n$
  (see Steinwart, 2004)
- $n_{SV}$ could be much smaller.
  (Burges, 1993; Vincent & Bengio, 2002)
- How to do it fast?
- How small?
An Inefficient Dual Optimizer

• Both $P$ and $N$ are linear combinations of examples with positive coefficients summing to one.

• Projection: $N' = (1 - \gamma)N + \gamma x$ with $0 \leq \gamma \leq 1$.

• Projection time proportional to $n_{sv}$. 
Two Problems with this Algorithm

- **Eliminating unwanted Support Vectors**
  
  - Pattern $x$ already has $\alpha > 0$. But we found better support vectors.
  - Simple algo decreases $\alpha$ too slowly.
  - Solution: let $\gamma$ be negative.

- **Process Support Vectors often enough**
  
  When drawing examples randomly,
  - Most have $\alpha = 0$ and should remain so.
  - Support vectors $(\alpha > 0)$ need adjustments but are rarely processed.
The Huller and its Derivatives

• The Huller

| Repeat | PROCESS: | Pick a random fresh example and project. |
|        | REPROCESS: | Pick a random support vector and project. |

– Compare with incremental learning and retraining.
– PROCESS potentially adds support vectors.
– REPROCESS potentially discard support vectors.

• Derivatives of the Huller

– LASVM handles soft-margins and is connected to SMO.
– LARANK handles multiclass problems and structured outputs.

(Bordes et al., 2005, 2006, 2007)
One Pass Learning with Kernels
Time and Memory

<table>
<thead>
<tr>
<th>Time</th>
<th>$n \ n_{SV}$</th>
<th>scales like plain SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>$n_{SV}^2$</td>
<td>scales much better</td>
</tr>
</tbody>
</table>

Unfortunately we still have $n_{SV} \sim 2Bn$. 

Training time variation vs. cache size

- LASVM(x1)
- LibSvm

eps: 0.003, 0.03, 0.1, 0.3, 1
Selecting Training Examples

Choosing which example to process next.

- **Randomly**

- **Gradient Selection**
  - Strongest coefficient in the gradient of the dual, i.e., selecting the most incorrect example.
    \[ k^* = \arg \min_k y_k \hat{y}(x_k) \]

- **Active Selection**
  - Gradient selection leads to selecting outliers!
  - Ignore the label with a minimax approach, i.e., selecting the example closest to the decision boundary.
    \[ k^* = \arg \min_k \max_{y=\pm 1} y \hat{y}(x_k) = \arg \min_k |\hat{y}(x_k)| \]
Selecting Examples by Sampling

Selecting Examples without Exhaustive Search

Given an example selection criterion:
– Pick $m$ random examples
– Choose the one with best criterion.

• When $m = 59$, we have 95% chances to be among the 5% best, regardless of the total number of examples.

• Heuristics to adapt $m$. 
Active Example Selection — Small-scale

UCI Adult dataset

Income $\geq 50K$

32K examples.
The Price of Convexity

Transitory solutions
- lower test error
- more sparsity

Active example selection ignores outliers.

Same as using a nonconvex loss.
(Collobert et al., 2006)
Active Example Selection — Large-scale

- Handwritten digits recognition with on-the-fly generation of distorted training patterns.
- Very difficult problem for local kernels.
- Potentially many support vectors.
- More a challenge than a solution.

<table>
<thead>
<tr>
<th>Number of binary classifiers</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory for the kernel cache</td>
<td>6.5GB</td>
</tr>
<tr>
<td>Examples per classifiers</td>
<td>8.1M</td>
</tr>
<tr>
<td>Total training time</td>
<td>8 days</td>
</tr>
<tr>
<td>Test set error</td>
<td>0.67%</td>
</tr>
</tbody>
</table>

- Each example gets only one chance to be selected.
- Maybe the largest SVM training on a single CPU.

(Loosli et al., 2006)
Are we there yet?

<table>
<thead>
<tr>
<th>Training algorithm</th>
<th>SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training examples</td>
<td>(\approx 4M.)</td>
</tr>
<tr>
<td>Total training time</td>
<td>2-3 hours</td>
</tr>
<tr>
<td>Test set error</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

(Simard et al., ICDAR 2003)

- RBF kernels do not like invariances. (see Bengio & LeCun, 2007)
- Kernel SVMs need lots of memory to cache kernel values.
- Yet the experiment validates active example selection.
Large Scale Selection of Examples

- Data Production computers can perform Example Selection because example selection can be done with limited feedback.
Large Scale Selection of Examples

- **Conclusion:** Makers can do part of the Thinking.