Learning with Large Datasets

Léon Bottou

NEC Laboratories America

Why Large-scale Datasets?

- Data Mining

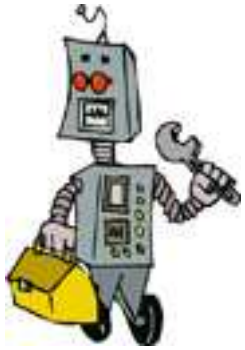
Gain competitive advantages by
analyzing data that describes the life of
our computerized society.

- Artificial Intelligence

Emulate cognitive capabilities of humans.
Humans learn from abundant and diverse data.

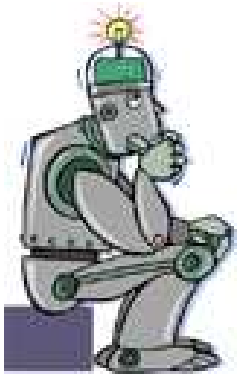
The Computerized Society Metaphor

- A society with just two kinds of computers:



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

Thinkers analyze the data to increase revenue by finding 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.

Part I

Statistical Efficiency versus Computational Costs.

This part is based on a joint work with Olivier Bousquet.

Simple Analysis

- **Statistical Learning Literature:**

"It is good to optimize an objective function than ensures a fast estimation rate when the number of examples increases."

- **Optimization Literature:**

"To efficiently solve large problems, it is preferable to choose an optimization algorithm with strong asymptotic properties, e.g. superlinear."

- **Therefore:**

"To address large-scale learning problems, use a superlinear algorithm to optimize an objective function with fast estimation rate.
Problem solved."

The purpose of this presentation is. . .

Too Simple an Analysis

- **Statistical Learning Literature:**
"It is good to optimize an objective function than ensures a fast estimation rate when the number of examples increases."

- **Optimization Literature:**
"To efficiently solve large problems, it is preferable to choose an optimization algorithm with strong asymptotic properties, e.g. superlinear."

- **Therefore:** *(error)*
"To address large-scale learning problems, use a superlinear algorithm to optimize an objective function with fast estimation rate.
Problem solved."

. . . to show that this is completely wrong !

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?

- We need a theory that joins Statistics and Computation!

– 1967: Vapnik's theory does not discuss computation.

– 1981: Valiant's learnability excludes exponential time algorithms, but (i) polynomial time can be too slow, (ii) few actual results.

– We propose a simple analysis of approximate optimization. . .

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 $f^*_{\mathcal{F}} \in \mathcal{F}$ 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)

$$E(\tilde{f}_n) - E(f^*) = E(f^*_{\mathcal{F}}) - E(f^*) \quad \text{Approximation error}$$

$$+ E(f_n) - E(f^*_{\mathcal{F}}) \quad \text{Estimation error}$$

$$+ E(\tilde{f}_n) - E(f_n) \quad \text{Optimization error}$$

Problem:

Choose $\mathcal{F}$, n, and ρ to make this as small as possible,

subject to budget constraints $\begin{cases} \text{maximal number of examples } n \\ \text{maximal computing time } T \end{cases}$

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 ρ.

Computing time T: (Algorithm dependent)

– decreases with ρ

– increases with n

– increases with $\mathcal{F}$

Small-scale vs. Large-scale Learning

We can give *rigorous definitions*.

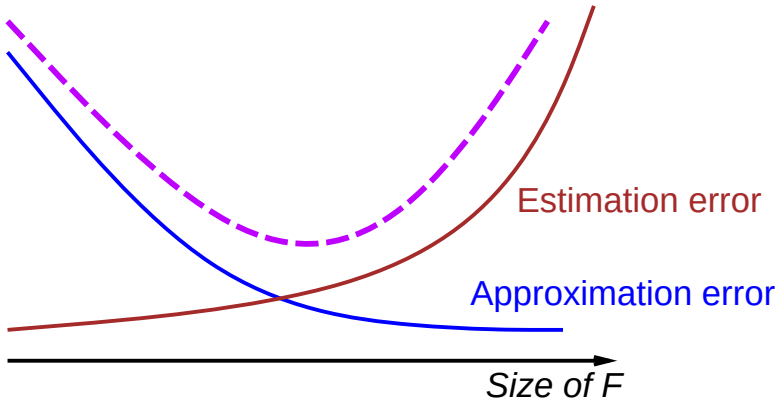
- **Definition 1:**
  We have a **small-scale learning** problem when the **active budget constraint is the number of examples n.**

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

Small-scale Learning

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.

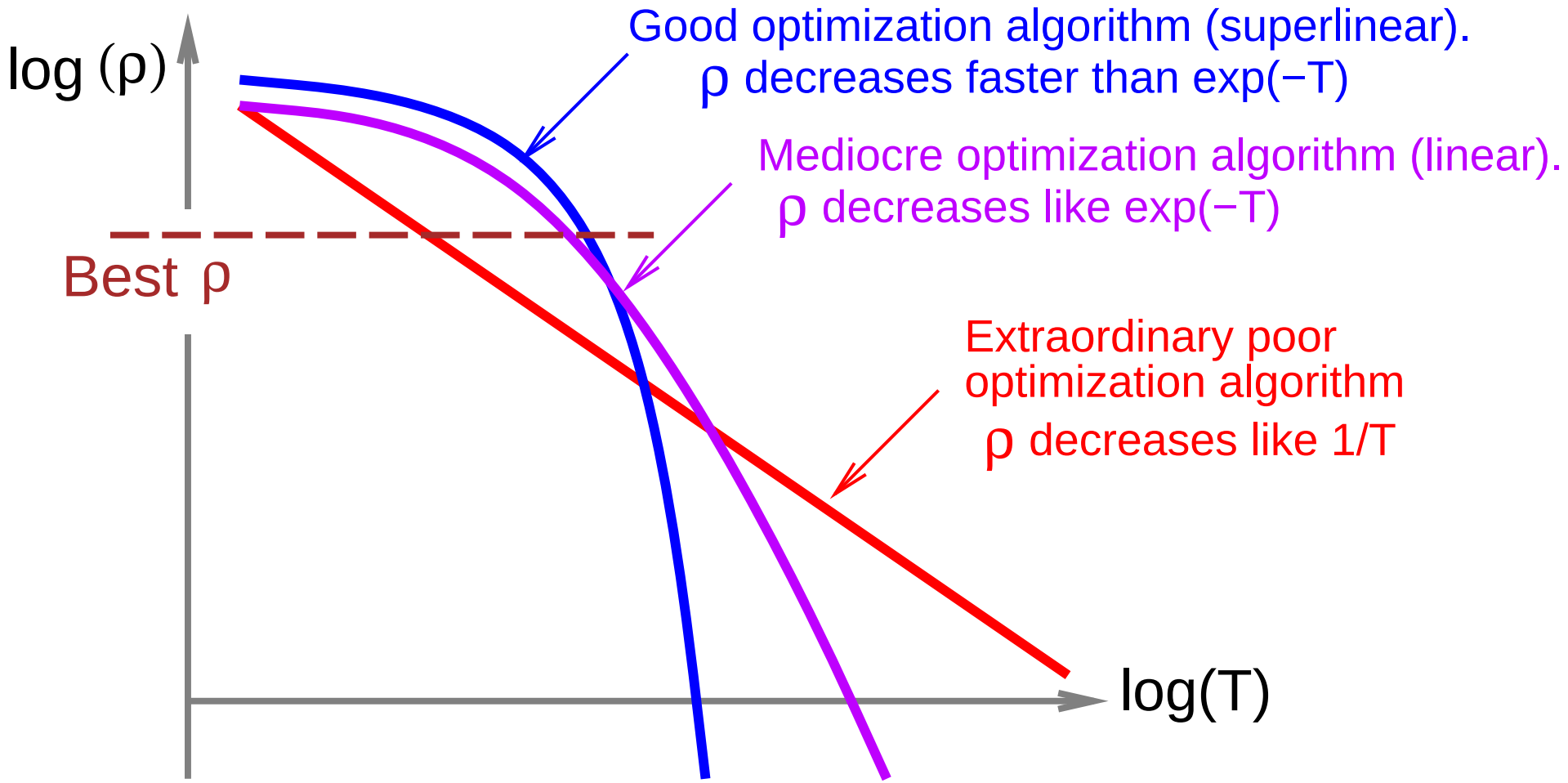
- More complicated tradeoffs.
  The computing time depends on the three variables: $\mathcal{F}$, n, and ρ.

- Example.
  If we choose ρ small, we decrease the optimization error. But we must also decrease $\mathcal{F}$ 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.

Executive Summary



Asymptotics: Estimation

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

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

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

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

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

Scaling laws for ρ when $\mathcal{F}$ is fixed

The approximation error is constant.

- No need to choose ρ smaller than $\mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha}\right)$.
- Not advisable to choose ρ larger than $\mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha}\right)$.

. . . Approximation+Estimation+Optimization

When $\mathcal{F}$ is chosen via a λ-regularized cost

- Uniform convergence theory provides bounds for simple cases
  (Massart-2000; Zhang 2005; Steinwart et al., 2004-2007; . . .)
- Computing time depends on both λ and ρ.
- Scaling laws for λ and ρ 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 $f_w(x)$ linearly 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).
3. Stochastic gradient descent.
4. Stochastic second order gradient descent.

Quantities of Interest

- Empirical Hessian at the empirical optimum w_n.

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

- Empirical Fisher Information matrix at the empirical optimum w_n.

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

- **Condition number**

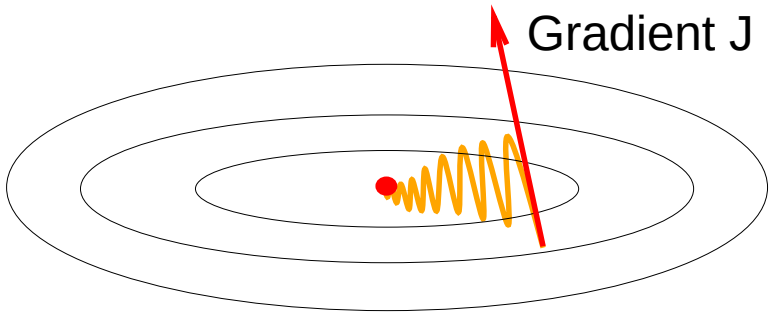
  We assume that there are $\lambda_{\min}$, $\lambda_{\max}$ and ν such that
  - $\text{trace}\left(GH^{-1}\right) \approx \nu$.
  - $\text{spectrum}\left(H\right) \subset [\lambda_{\min}, \lambda_{\max}]$.

  and we define the condition number $\kappa = \lambda_{\max}/\lambda_{\min}$.

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 $\eta = \frac{1}{\lambda_{\max}}$.
(e.g., Dennis & Schnabel, 1983)

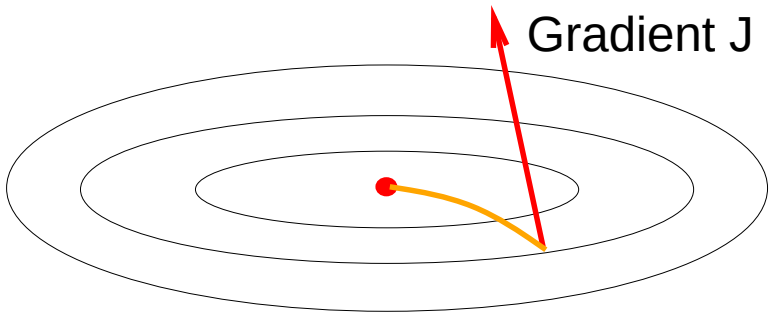
	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
GD	$\mathcal{O}(nd)$	$\mathcal{O}\left(\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(nd\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2 \kappa}{\varepsilon^{1/\alpha}} \log^2 \frac{1}{\varepsilon}\right)$

– In the last column, n and ρ are chosen to reach ε as fast as possible.
– Solve for ε to find the best error rate achievable in a given time.
– Remark: abuses of the $\mathcal{O}()$ notation

Second Order Gradient Descent (2GD)

Iterate

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



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

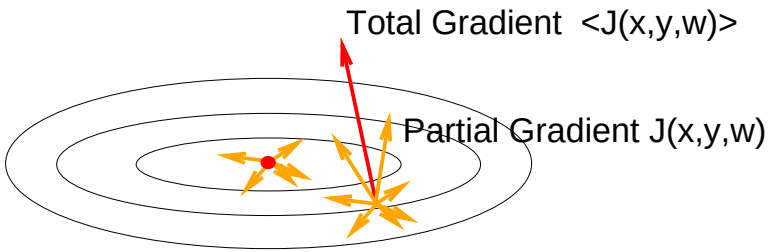
	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
2GD	$\mathcal{O}\big(d\big(d+n\big)\big)$	$\mathcal{O}\Big(\log\log\frac{1}{\rho}\Big)$	$\mathcal{O}\Big(d\big(d+n\big)\log\log\frac{1}{\rho}\Big)$	$\mathcal{O}\Big(\frac{d^2}{\varepsilon^{1/\alpha}}\log\frac{1}{\varepsilon}\log\log\frac{1}{\varepsilon}\Big)$

– Optimization speed is much faster.
– Learning speed only saves the condition number $\boldsymbol{\kappa}$.

Stochastic Gradient Descent (SGD)

Iterate

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



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

	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
SGD	$\mathcal{O}(d)$	$\frac{\nu\, k}{\rho} + o\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d\, \nu\, k}{\rho}\right)$	$\mathcal{O}\left(\frac{d\, \nu\, k}{\varepsilon}\right)$

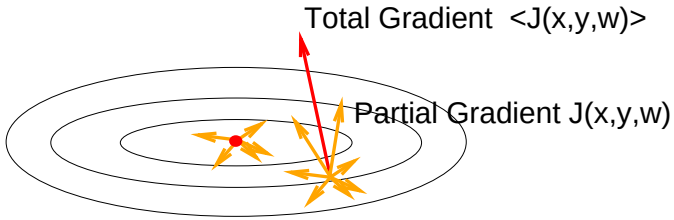
With $1 \leq k \leq \kappa^2$

– Optimization speed is *catastrophic*.
– Learning speed does not depend on the statistical estimation rate α.
– Learning speed depends on condition number κ but *scales very well*.

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 $\frac{1}{t} H^{-1}$.

	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
2SGD	$\mathcal{O}(d^2)$	$\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2\,\nu}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2\,\nu}{\varepsilon}\right)$

– Each iteration is d times more expensive.
– The number of iterations is reduced by κ^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

Download from `http://leon.bottou.org/projects/sgd`.
These simple programs are very short.

See also (Shalev-Schwartz et al., 2007; Vishwanathan et al., 2006)

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 $w \leftarrow w - \eta_t \nabla(w_t, x_t, y_t) = w - \eta_t \left(\lambda w + \frac{\partial \ell(w\, x_t + b,\, y_t)}{\partial w} \right)$

Same setup as (Shalev-Schwartz et al., 2007) but plain SGD.

Text Categorization with SVMs

- **Results: Linear SVM**

  $\ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\}$ $\lambda = 0.0001$

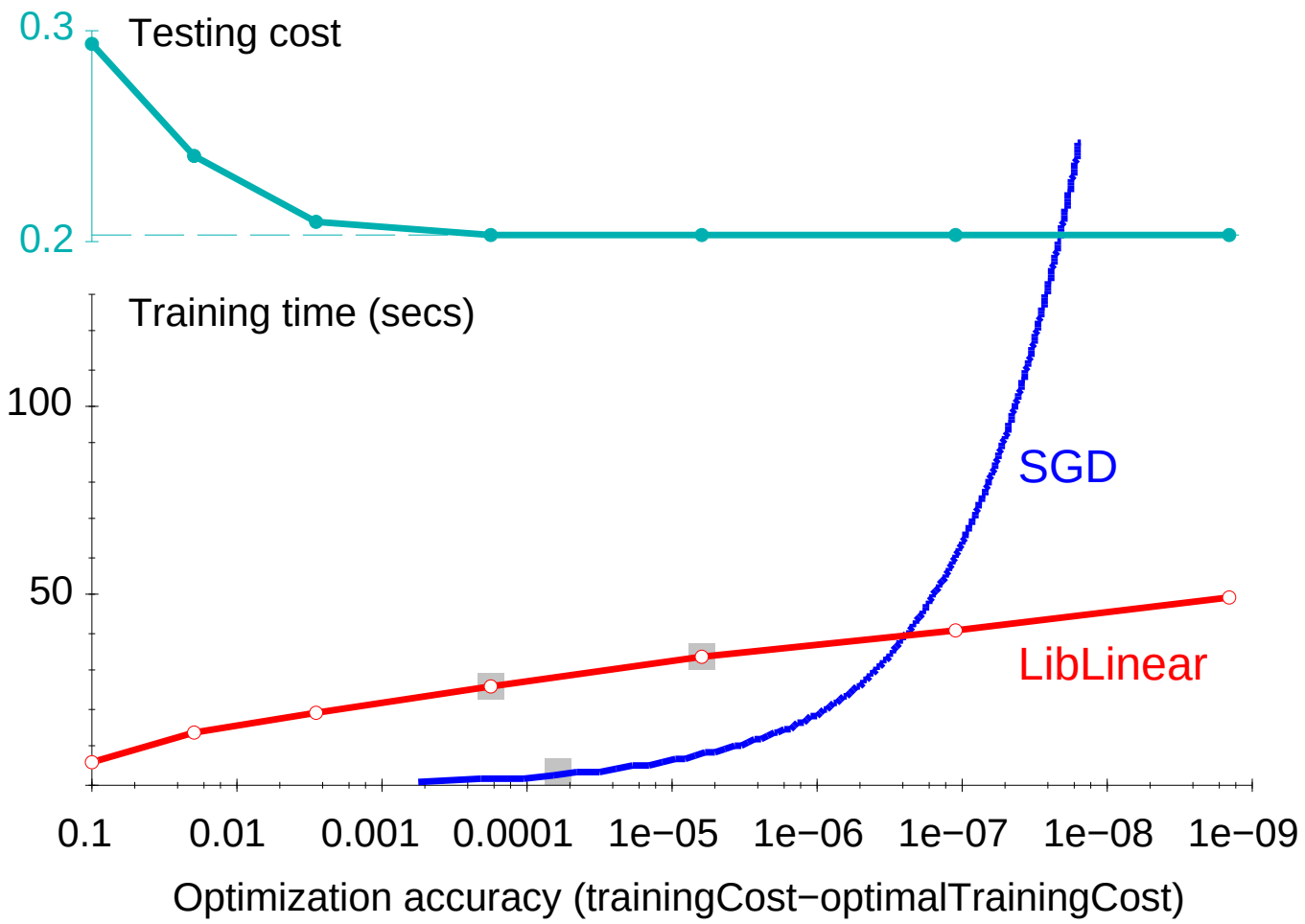
	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	*0.2275*	*6.02%*
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	*0.2275*	*6.02%*

- **Results: Log-Loss Classifier**

  $\ell(\hat{y}, y) = log(1 + exp(-y\hat{y}))$ $\lambda = 0.00001$

	Training Time	Primal cost	Test Error
LibLinear ($\varepsilon = 0.01$)	30 secs	0.18907	*5.68%*
LibLinear ($\varepsilon = 0.001$)	44 secs	*0.18890*	5.70%
SGD	2.3 secs	0.18893	*5.66%*

The Wall


0.3
Testing cost
0.2
Training time (secs)
100
50
SGD
LibLinear
0.1
0.01
0.001
0.0001
1e−05
1e−06
1e−07
1e−08
1e−09
Optimization accuracy (trainingCost−optimalTrainingCost)

More SVM Experiments

From: Patrick Haffner

Date: Wednesday 2007-09-05 14:28:50

. . . I have tried on some of our main datasets. . .

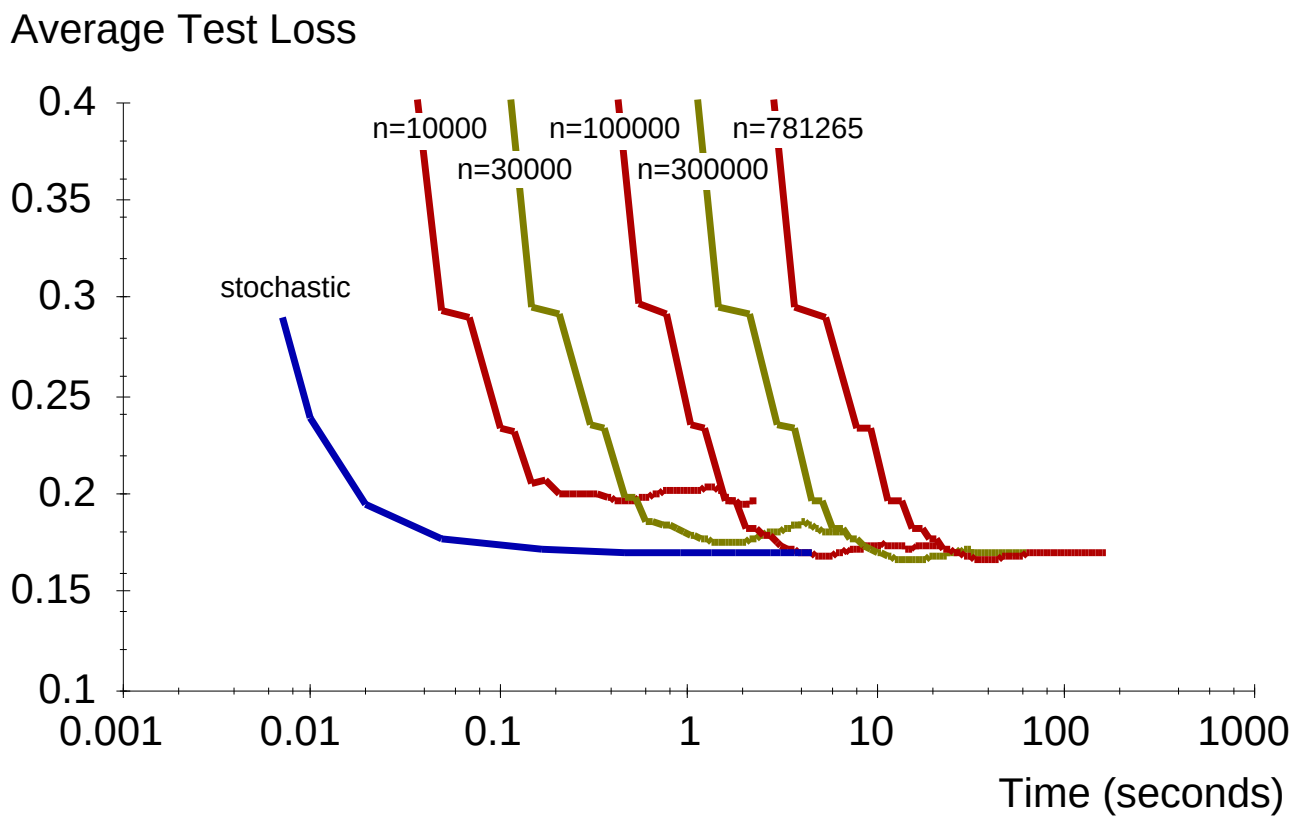
I can send you the example, it is so striking!

– Patrick

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

More SVM Experiments

From: Olivier Chapelle
Date: Sunday 2007-10-28 22:26:44
. . . you should really run batch with various training set sizes . . .
– Olivier



Log-loss problem

Batch Conjugate Gradient on various training set sizes

Stochastic Gradient on the full set

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.

Same setup as (Vishwanathan et al., 2006) but plain SGD.

Text Chunking with CRFs

- **Results**

	Training Time	Primal cost	Test F1 score
L-BFGS	4335 secs	*9042*	93.74%
SGD	568 secs	9098	*93.75%*

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

- **Example: the SVM benchmark**
  - Use $\eta = 1/\lambda$ because $\lambda \leq \lambda_{\min}$.
  - 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 η on the subsample.
- Pick the gain η 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.

Getting the Engineering Right

The very simple SGD update offers lots of engineering opportunities.

Example: Sparse Linear SVM

The update $w \leftarrow w - \eta\big(\lambda w - \nabla \ell(w x_i, y_i)\big)$
can be performed in two steps:

i) $w \leftarrow w - \eta \nabla \ell(w x_i, y_i)$ (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 Kernel Machines

- **SGD for Linear SVM**
  - Both w and $\nabla \ell(w x_t, y_t)$ represented using coordinates.
  - SGD updates w by combining coordinates.
- **SGD for SVM with Kernel $K(x_i, x_j) = < \Phi(x_i), \Phi(x_j) >$**
  - Represent w with its kernel expansion $\sum \alpha_i \, \Phi(x_i)$.
  - Usually, $\nabla \ell(w x_t, y_t) \; = \; -\mu \, \Phi(x_t)$.
  - SGD updates w by combining coefficients:

$$\alpha_i \longleftarrow (1 - \eta\lambda) \, \alpha_i + \begin{cases} \eta \, \mu & \text{if } i = t, \\ 0 & \text{otherwise.} \end{cases}$$

- **So, one just needs a good sparse vector library ?**

SGD for Kernel Machines

- **Sparsity Problems.**

$$\alpha_i \longleftarrow (1-\eta\lambda)\,\alpha_i + \begin{cases} \eta\,\mu & \text{if } i = t, \\ 0 & \text{otherwise.} \end{cases}$$

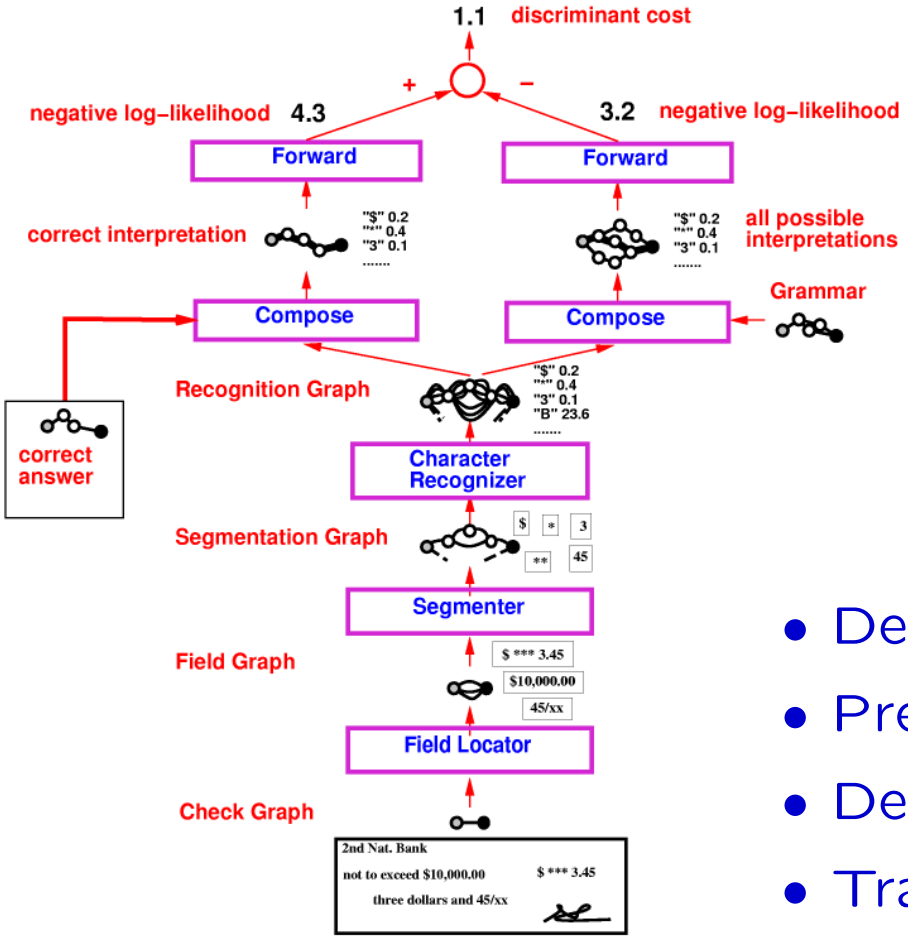
  - Each iteration potentially makes one α coefficient non zero.
  - Not all of them should be support vectors.
  - Their α coefficients take a long time to reach zero (Collobert, 2004).

- **Dual algorihms related to primal SGD avoid this issue.**
  - Greedy algorithms (Vincent et al., 2000; Keerthi et al., 2007)
  - LaSVM and related algorithms (Bordes et al., 2005)

    More on them later...

- **But they still need to compute the kernel values!**
  - Computing kernel values can be slow.
  - Caching kernel values can require lots of memory.

SGD for Real Life Applications



A Check Reader

Examples are pairs (image,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.

Credits: Bengio, Bottou, Burges, Haffner, LeCun, Nohl, Simard, et al.

Part III

Learning with a Single Pass over the Examples

This part is based on joint works with
Antoine Bordes, Seyda Ertekin, Yann LeCun, and Jason Weston.

Why learning with a Single Pass?

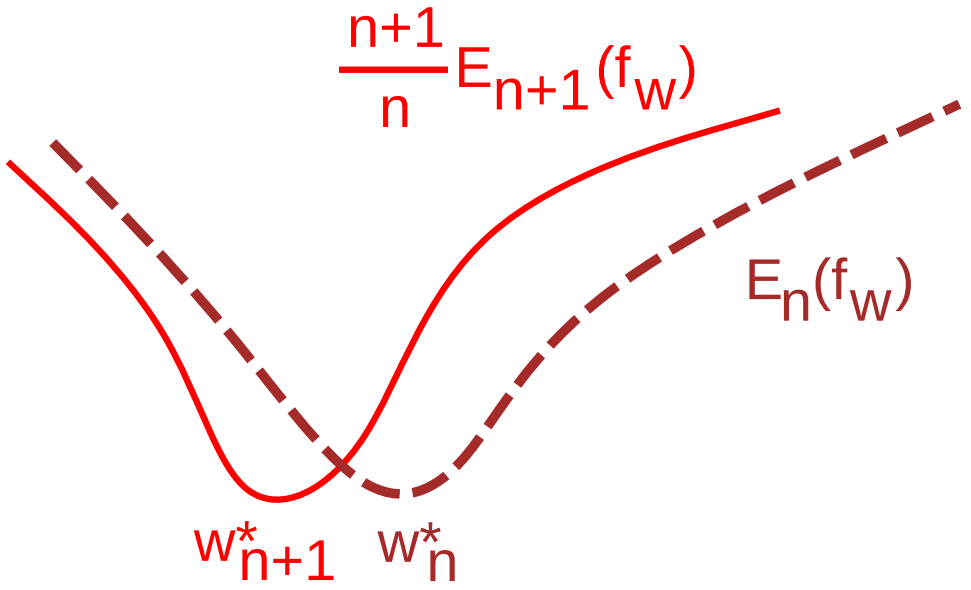
- **Motivation**
  - Sometimes there is too much data to store.
  - Sometimes retrieving archived data is too expensive.
- **Related Topics**
  - Streaming data.
  - Tracking nonstationarities.
  - Novelty detection.
- **Outline**
  - One-pass learning with second order SGD.
  - One-pass learning with kernel machines.
  - Comparisons

Effect of one Additional Example (i)

Compare

$$w_n^* = \arg\min_w E_n(f_w)$$

$$w_{n+1}^* = \arg\min_w E_{n+1}(f_w) = \arg\min_w \left[E_n(f_w) + \frac{1}{n}\ell\big(f_w(x_{n+1}), y_{n+1}\big) \right]$$



Effect of one Additional Example (ii)

- **First Order Calculation**

$$w^*_{n+1} \;=\; w^*_n \;-\; \frac{1}{n}\, H^{-1}_{n+1}\, \frac{\partial\, \ell\big(f_{w_n}(x_n), y_n\big)}{\partial w} \;+\; \mathcal{O}\left(\frac{1}{n^2}\right)$$

where H_{n+1} is the empirical Hessian on $n+1$ examples.

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

$$w_{t+1} \;=\; w_t \;-\; \frac{1}{t}\, H^{-1}\, \frac{\partial\, \ell\big(f_{w_t}(x_n), y_n\big)}{\partial w}$$

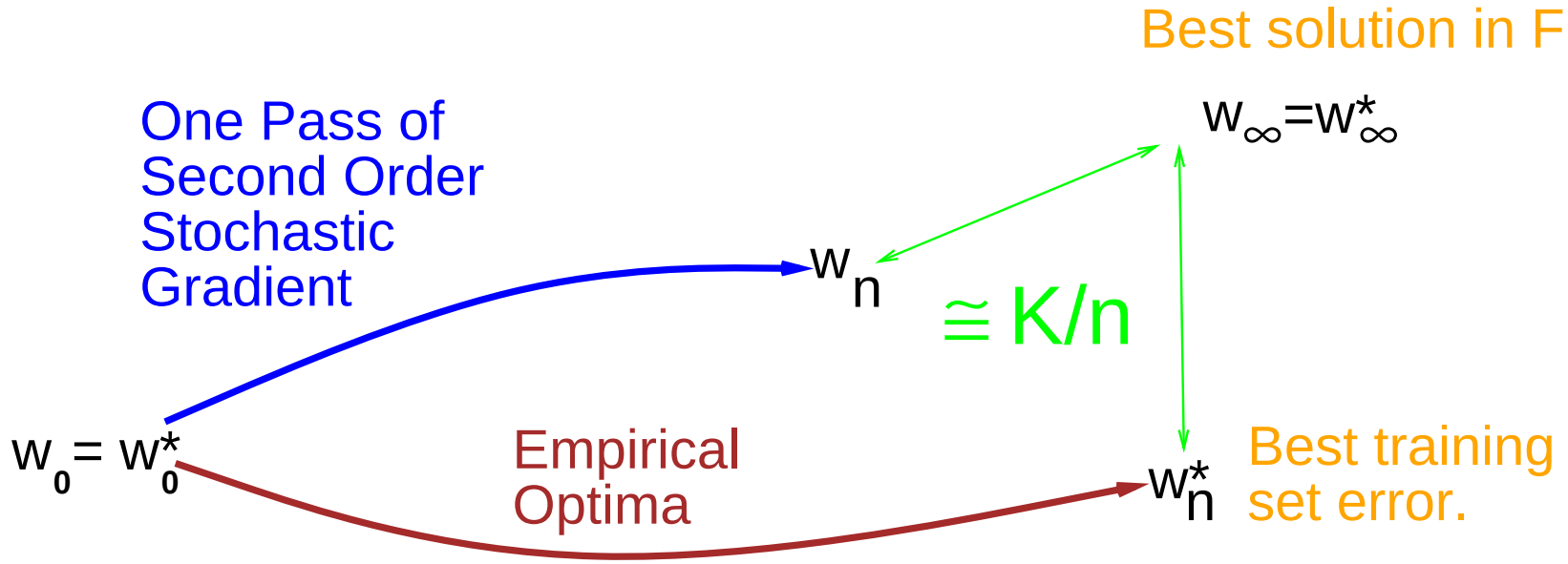
- **Could they converge with the same speed?**

Yes they do! But what does it mean?

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

  Under "adequate conditions"

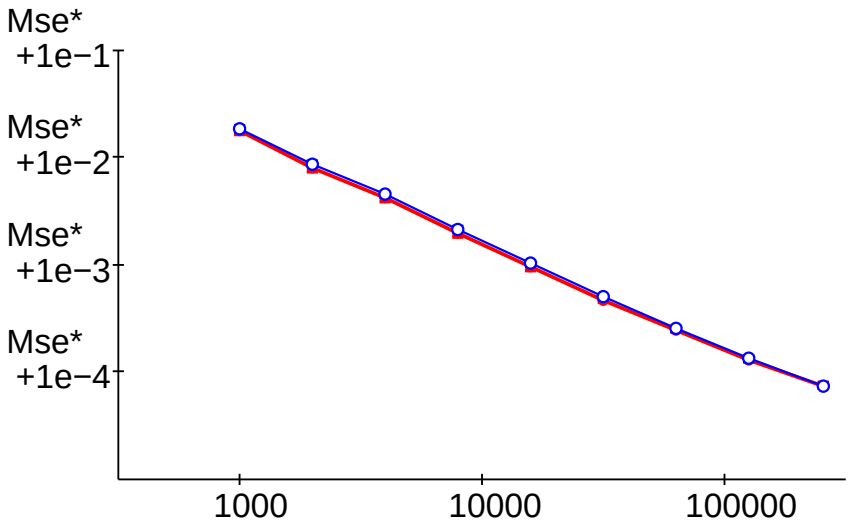
$$\lim_{n\to\infty} n\,\|w_\infty^* - w_n^*\|^2 = \lim_{t\to\infty} t\,\|w_\infty - w_t\|^2 = \mathrm{tr}(H^{-1}G\,H^{-1})$$
$$\lim_{n\to\infty} n\,[E(f_{w_n^*}) - E(f_{\mathcal{F}})] = \lim_{t\to\infty} t\,[E(f_{w_t}) - E(f_{\mathcal{F}})] = \mathrm{tr}(G\,H^{-1})$$



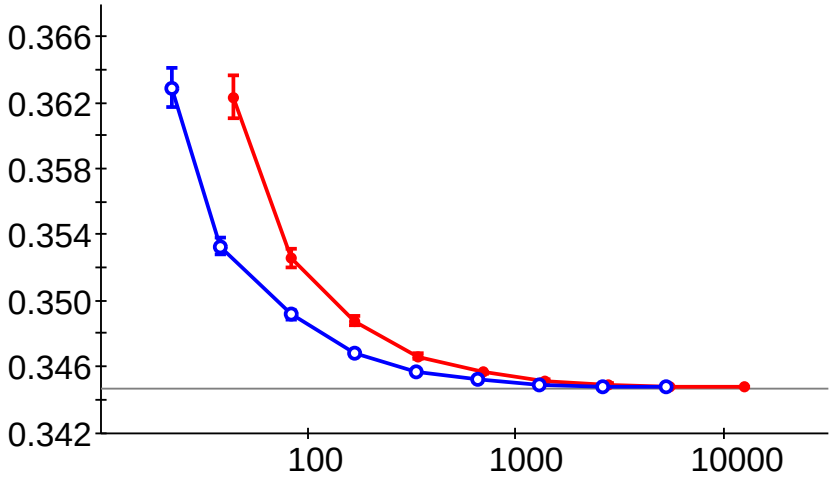
Optimal Learning in One Pass

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



Number of examples



Milliseconds

Unfortunate Practical Issues

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

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

  - 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.

- **Research Directions**

  Limited storage approximations of H^{-1}.

  - Reduce the number of epochs
  - Rarely sufficient for fast one-pass learning.
  - Diagonal approximation (Becker &LeCun, 1989)
  - Low rank approximation (e.g., LeCun et al., 1998)
  - Online L-BFGS approximation (Schraudolph, 2007)

Disgression: Stopping Criteria for SGD

	2SGD	SGD
Time to reach accuracy ρ	$\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right)$	$\frac{k\,\nu}{\rho} + o\left(\frac{1}{\rho}\right)$
Number of epochs to reach same test cost as the full optimization.	1	k $1 \leq k \leq \kappa^2$

There are many ways to make constant k smaller:

– Exact second order stochastic gradient descent.

– Approximate second order stochastic gradient descent.

– Simple preconditionning tricks.

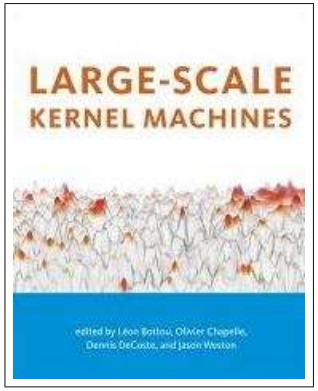
Disgression: Stopping Criteria for SGD

- **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.

One-pass learning for Kernel Machines?



Challenges for Large-Scale Kernel Machines:

– Bulky kernel matrix ($n \times n$.)

– Managing the kernel expansion $\boldsymbol{w} = \sum \boldsymbol{\alpha_i}\, \boldsymbol{\Phi}(\boldsymbol{x_i})$.

– Managing memory.

Issues of SGD for Kernel Machines:

– Conceptually simple.

– Sparsity issues in kernel expansion.

Stochastic and Incremental SVMs:

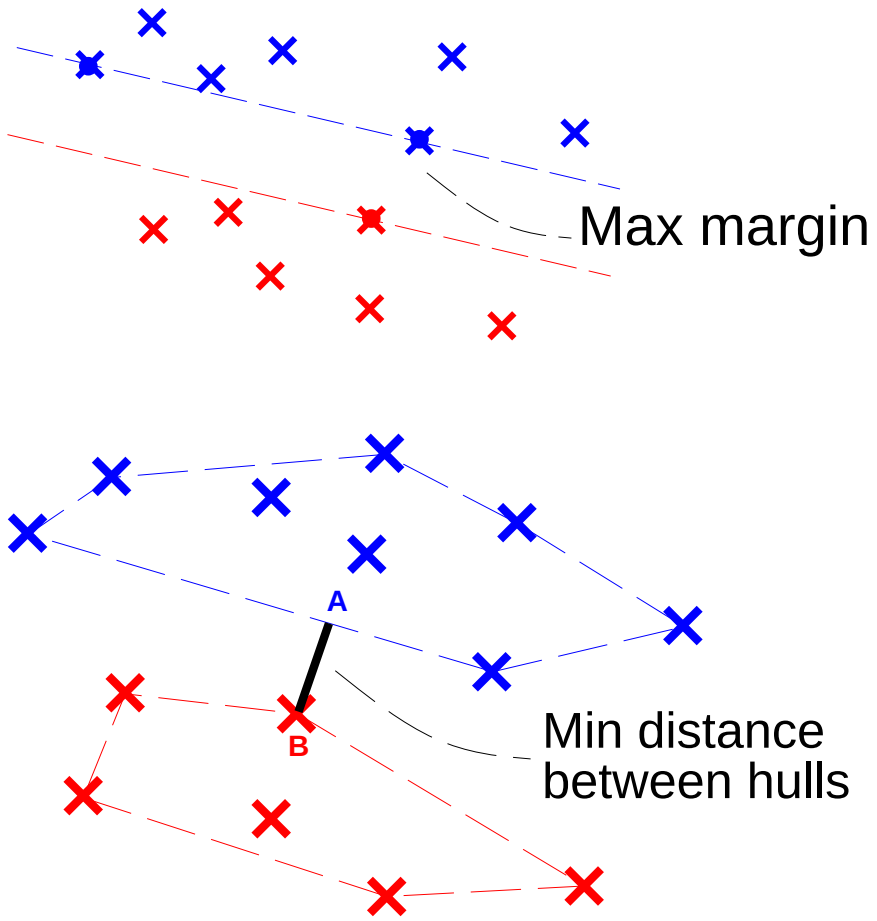
– Iteratively constructing the kernel expansion.

– Which candidate support vectors to store and discard?

– Managing the memory required by the kernel values.

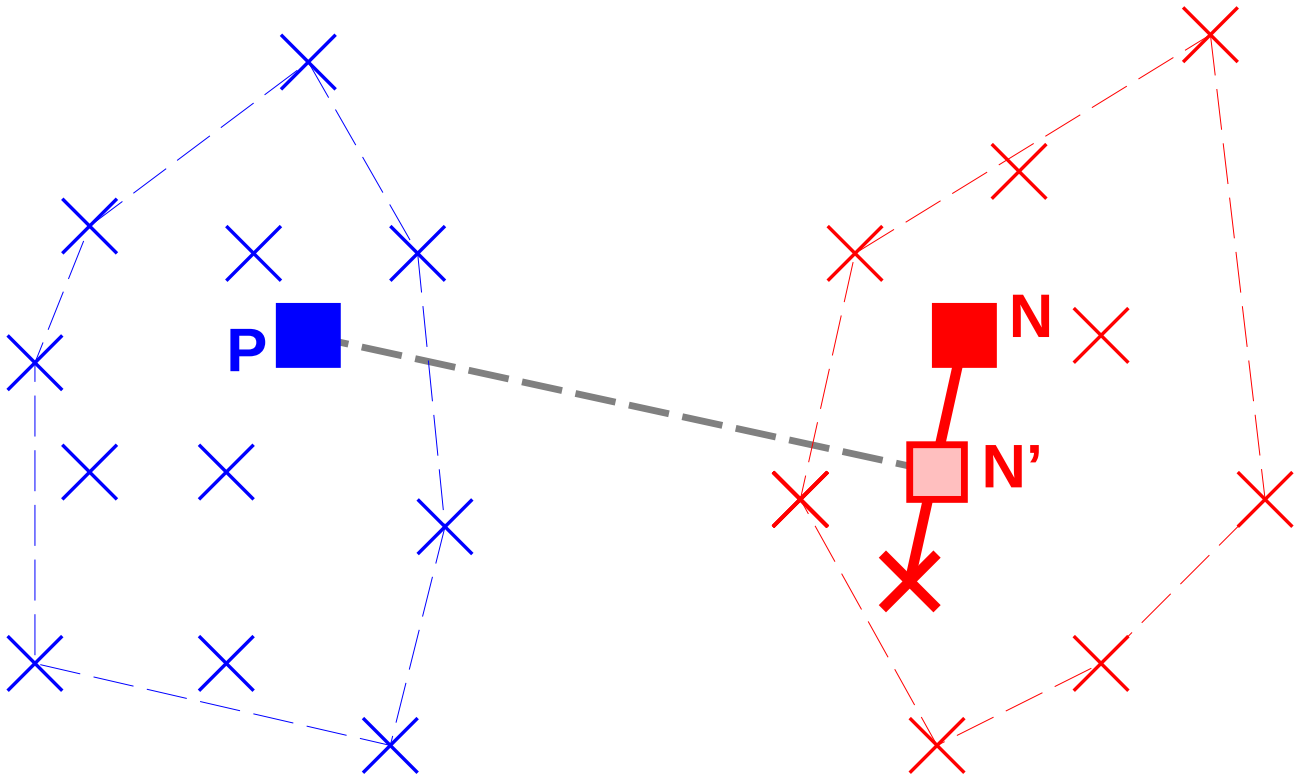
– One-pass learning?

Learning in the dual



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

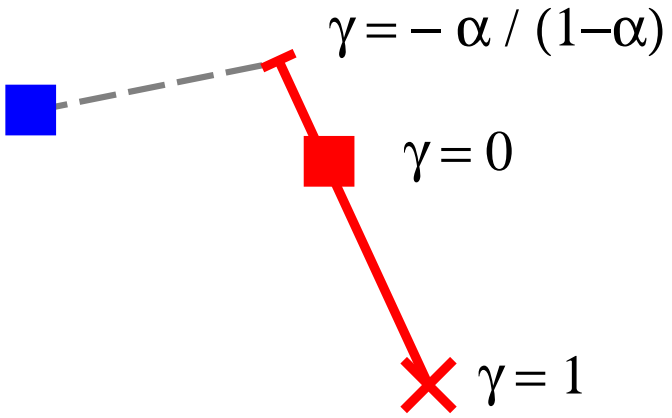
An Inefficient Dual Optimizer



- 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 α too slowly.
– Same problem as SGD in fact.
– Solution: Allow γ to be slightly negative.

- **Processing 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.
– Solution: Draw support vectors more often.

The Huller and its Derivatives

- **The Huller**

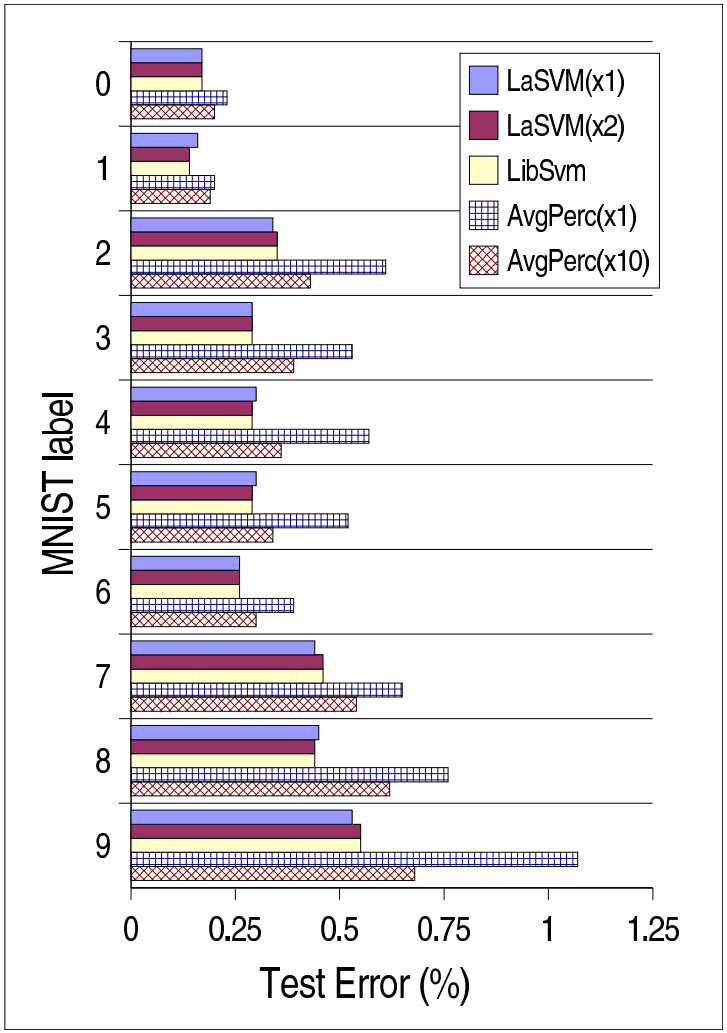
Repeat	
PROCESS:	Pick a random fresh example and project.
REPROCESS:	Pick a random support vector and project.

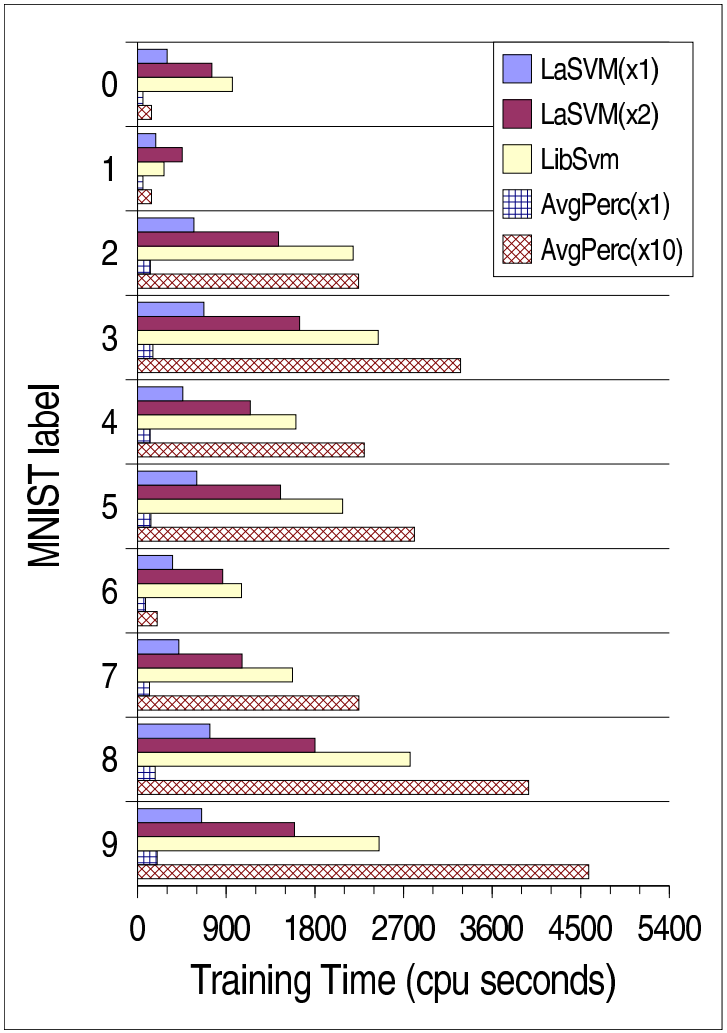
  - Compare with incremental learning and retraining.
  - PROCESS potentially adds support vectors.
  - REPROCESS potentially discard support vectors.

- **Derivatives of the Huller**
  - LASVM handles soft-margins and is connected to SMO.
  - LARANK handles multiclass problems and structured outputs.

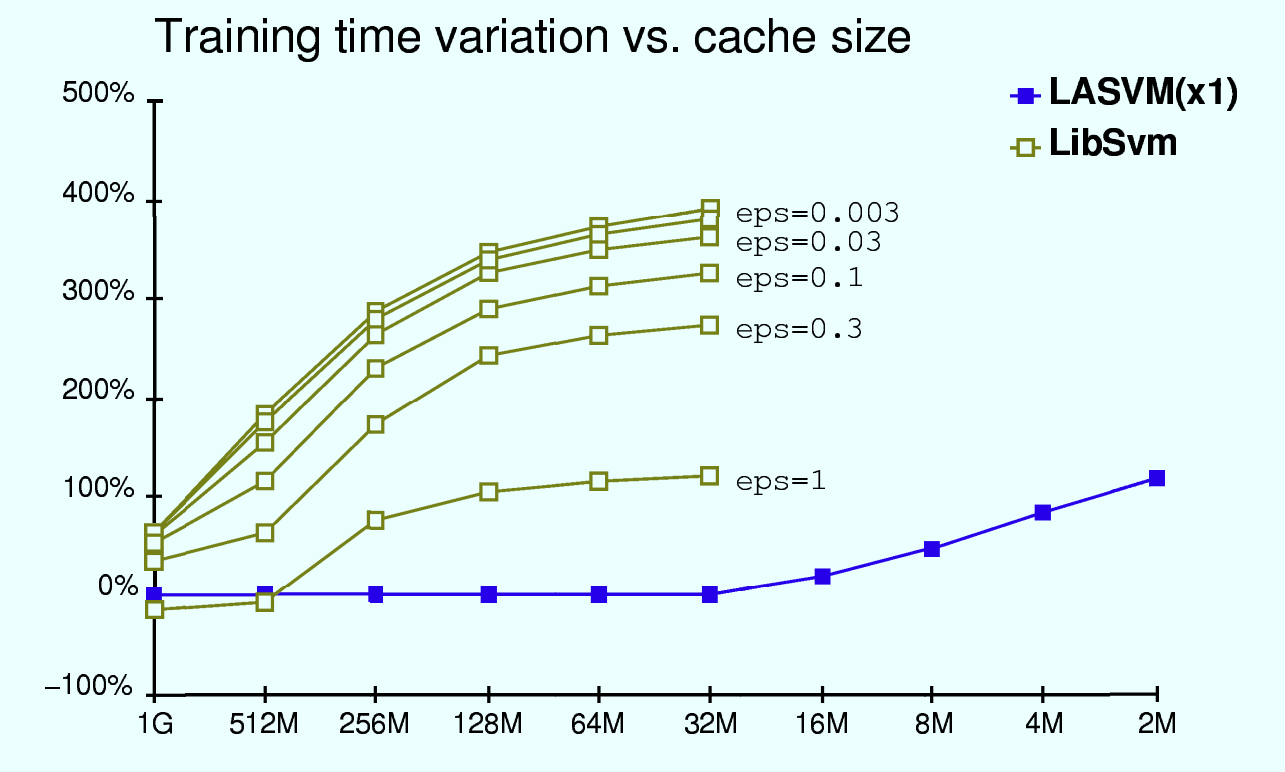
(Bordes et al., 2005, 2006, 2007)

One Pass Learning with Kernels





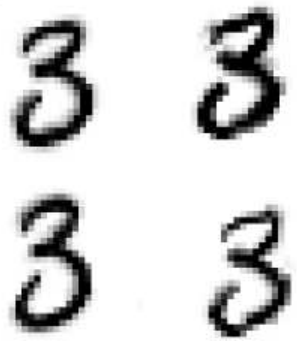
Time and Memory



	LibSVM	**LaSVM**	**2SGD**	SGD
Time	$n\,s$	$n\,s$	$n\,d^2$	$n\,d\,k$
Memory	$n\,r$	r^2	d^2	d

Careless comparisons: $n \gg s \gg r$ and $r \approx d$

Are we there yet?

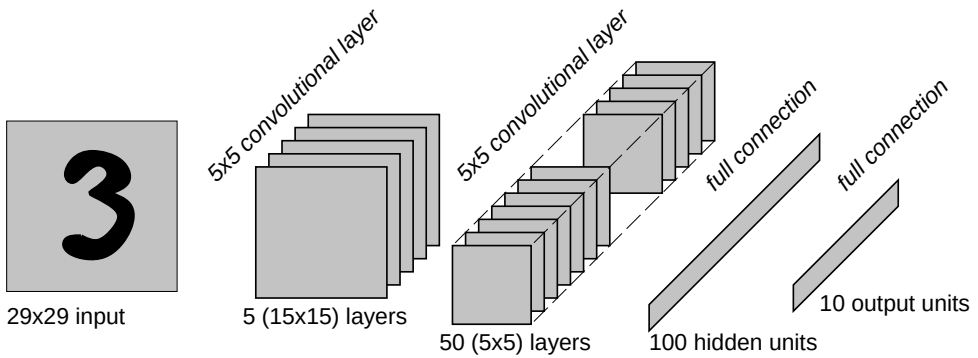


- Handwritten digits recognition with on-the-fly generation of distorted training patterns.
- Very difficult problem for local kernels.
- Potentially many support vectors.
- More a challenge than a solution.

Number of binary classifiers	**10**
Memory for the kernel cache	**6.5GB**
Examples per classifiers	**8.1M**
Total training time	**8 days**
Test set error	**0.67%**

- Trains in one pass: 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?



Training algorithm	**SGD**
Training examples	$\approx$ **4M.**
Total training time	**2-3 hours**
Test set error	**0.4%**

(Simard et al., ICDAR 2003)

- RBF kernels **cannot compete** with task specific models.
- The kernel SVM is **slower** because it needs **more memory**.
- The kernel SVM **trains with a single pass**.

Conclusion

- Connection between Statistics and Computation.
- Qualitatively different tradeoffs for small– and large–scale.
- Plain SGD rocks in theory and in practice.
- One-pass learning feasible with 2SGD or dual techniques.
  Current algorithms still slower than plain SGD.
- Important topics not addressed today:
  Example selection, data quality, weak supervision.