Gradient Methods for Machine Learning

[redacted]

Nic Schraudolph

1

MLSS Canberra 2005

Course Overview

1. Mon: **Classical Gradient Methods**
   Direct (gradient-free), Steepest Descent, Newton, Levenberg-Marquardt, BFGS, Conjugate Gradient
2. Tue: **Stochastic Approximation** (SA)
   Why necessary, why difficult. Step size adaptation.
3. Thu: **Stochastic Meta-Descent** (SMD)
   Advanced stochastic step size adaptation method.
4. Fri: **Algorithmic Differentiation** (AD)
   Forward/reverse mode. Fast Hessian-vector products.

2

MLSS Canberra 2005

Classical Gradient Methods

- Note simultaneous course at AMSI (math) summer school: Nonlin. Optimization Methods (see http://wwwmaths.anu.edu.au/events/amsiss05/)
- Recommended textbook (Springer Verlag, 1999): Nocedal & Wright, Numerical Optimization
- Here: just quick overview, unconstrained only
- But will consider large, nonlinear problems

3

MLSS Canberra 2005

Function Optimization

- Goal: given (diff'able) function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ find minimum $\vec{w}^* = \arg\min_{\vec{w}} f(\vec{w})$
- Gradient methods find only local minimum
- For convex functions, local min. = global min.

In machine learning,

- Fn. is defined over data: $f(\vec{w}) = E_{\vec{x}}[f(\vec{w};\vec{x})]$
- May comprise loss and regularization terms

4

MLSS Canberra 2005

Methods by Gradient Order

- 0th order (direct, gradient-free) methods use only the function values themselves
- 1st order gradient methods additionally use function's gradient $\vec{g} = \vec{g}(\vec{w}) = \frac{\partial f(\vec{w})}{\partial \vec{w}}$
- 2nd order gradient methods also use the function's Hessian $H = H(\vec{w}) = \frac{\partial^2 f(\vec{w})}{\partial \vec{w}\, \partial \vec{w}^T}$

Direct (Gradient-Free) Methods

Many distinct algorithms:

- Simulated annealing, Monte Carlo optim.
- Perturbation methods, SPSA, Tabu search
- Genetic algorithms, evolutionary strategies, ant colony optimization, …

Differ in many implementation details but all share a common approach.

Prototypical Direct Method

```
Randomly initialize pool W of candidates
Repeat until converged:
  pick parent(s) $w_i$ from
  generate child(ren) $w_i'$ = perturb($w_i$)
  compare child to parent (or entire pool):
     $\Delta = f(w_i') - f(w_i)$
  if $\Delta < 0$ accept $w_i'$ into W (may replace $w_i$)
  else if global optimization:
     accept $w_i'$ into W with probability $P(e^{-\Delta})$
```

Direct Methods: Advantages

- No need to derive or compute gradients
  - Can solve discrete/combinatorial problems
  - Can address even non-formalized problems
- Can find (non-convex fn.'s) global optimum
- Highly and easily parallelizable
- Very fast iteration when perturbation and evaluation are both incremental, *i.e. O(1)*

Direct Methods: Disadvantages

- No sense of appropriate direction or size of step to take (perturbation is random)
  - Some algorithms try to fix this heuristically
- Takes too many iterations to converge
- Global optim. requires knowing acceptance of inferior candidates ⇒ slower still
- No strong mathematical underpinnings ⇒ jungle of ad-hoc heuristics

9

MLSS Canberra 2005

Gradient Descent

- Perturbs parameter vector in steepest downhill direction (= neg. gradient): $\vec{w}_{t+1} = \vec{w}_t - \eta\,\vec{g}_t$
- Step size η can be set
  - to small positive constant: simple gradient descent
  - by line minimization: steepest descent
  - adaptively (more on this later)

Advantage:

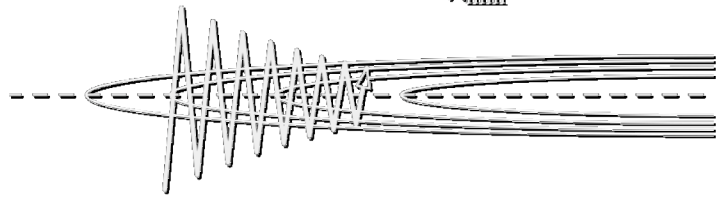
- Cheap to compute: iteration typically just ***O(n)***

10

MLSS Canberra 2005

Gradient Descent: Disadvantages

- Line minimization may be expensive
- Convergence slow for ill-conditioned problems: #iterations ≥ condition# $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ of Hessian



11

MLSS Canberra 2005

Newton's Method

- Local quadratic model

$$f(\vec{w}) = \tfrac{1}{2}(\vec{w} - \vec{w}^*)^T H (\vec{w} - \vec{w}^*)$$

has gradient $\vec{g}(\vec{w}) = H(\vec{w} - \vec{w}^*)$

therefore let $\vec{w}_{t+1} = \vec{w}_t - H_t^{-1}\vec{g}_t$

12

MLSS Canberra 2005

Newton's Method

Big advantage:

- Jumps directly to minimum of quadratic bowl (regardless of ill-conditioning)

Disadvantages:

- Hessian expensive to invert: nearly $O(n^3)$
- Hessian must be positive definite: $\lambda_{\min} > 0$
- May make huge, uncontrolled steps

13

MLSS Canberra 2005

Gauss-Newton Approximation

Let $f = l \circ \vec{m}$, $\vec{m} : \mathbb{R}^n \rightarrow \mathbb{R}^p$ = model, l = loss

Then
$$H_f = \underbrace{J_{\vec{m}}^T H_l J_{\vec{m}}} + \sum_{i=1}^{p} H_{m_i} (J_l)_i$$

Gauss-Newton: G_f

Jacobian:
$$(J_{\vec{m}})_{ij} = \frac{\partial m_i(\vec{w})}{\partial w_j}$$

- $H_l \geq 0 \Rightarrow G_f \geq 0$
- At minimum, $G_f = H_f$
- For sum-squared loss: $H_l = I$ and $G_f^{-1} \vec{g} = (J_{\vec{m}}^T J_{\vec{m}})^{-1} J_{\vec{m}}^T J_l^T = J_{\vec{m}}^{+} J_l^T$

pseudo-inverse

14

MLSS Canberra 2005

Levenberg-Marquardt

$$\vec{w}_{t+1} = \vec{w}_t - (G_t + \lambda\, \mathrm{diag}(G_t))^{-1} \vec{g}_t$$

- G_t is Gauss-Newton approximation to H_t (guaranteed positive semi-definite)
- $\lambda \geq 0$ adaptively controlled, limits step to an elliptical model-trust region
- Fixes Newton's stability issues, but still $O(n^3)$

15

MLSS Canberra 2005

Quasi-Newton: BFGS

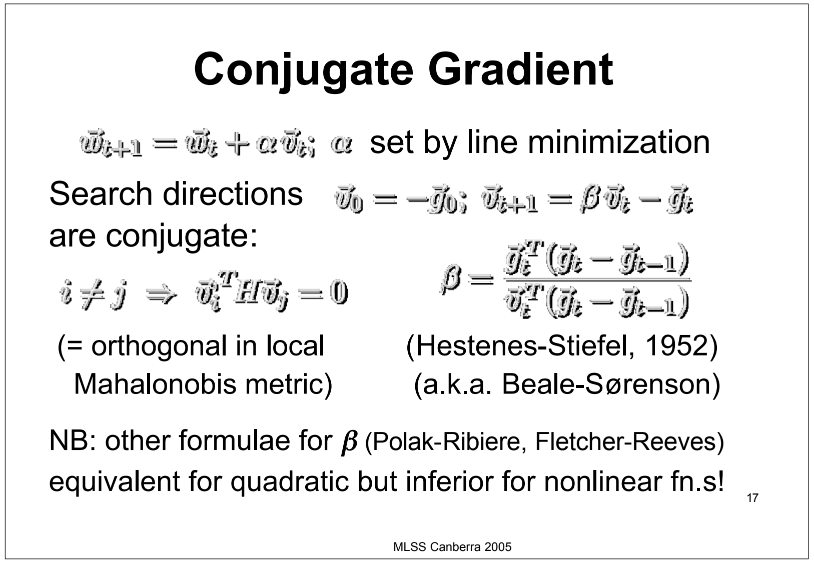
- Iteratively updates estimate B of H^{-1}
- Guarantees $B^T = B$ and $B > 0$
- Reduces complexity to $O(n^2)$ per iteration
- Requires line minimization (direction $B_t \vec{g}_t$)
- Update formula:

$$B_{t+1} = B_t + \frac{\Delta\vec{w}\, \Delta\vec{w}^T}{\Delta\vec{w}^T \Delta\vec{g}} - \frac{B_t \Delta\vec{g}\, \Delta\vec{g}^T B_t^T}{\Delta\vec{g}^T B_t \Delta\vec{g}} + \vec{u}\, \Delta\vec{g}^T B_t \Delta\vec{g}\, \vec{u}^T$$

where
$$\vec{u} \equiv \frac{\Delta\vec{w}}{\Delta\vec{w}^T \Delta\vec{g}} - \frac{B_t \Delta\vec{g}}{\Delta\vec{g}^T B_t \Delta\vec{g}}$$

16

MLSS Canberra 2005



Conjugate Gradient

$\vec{w}_{t+1} = \vec{w}_t + \alpha \vec{v}_t;\ \alpha$ set by line minimization

Search directions are conjugate: $\vec{v}_0 = -\vec{g}_0;\ \vec{v}_{t+1} = \beta \vec{v}_t - \vec{g}_t$

$i \neq j \ \Rightarrow\ \vec{v}_i^{\,T} H \vec{v}_j = 0$ (= orthogonal in local Mahalonobis metric)

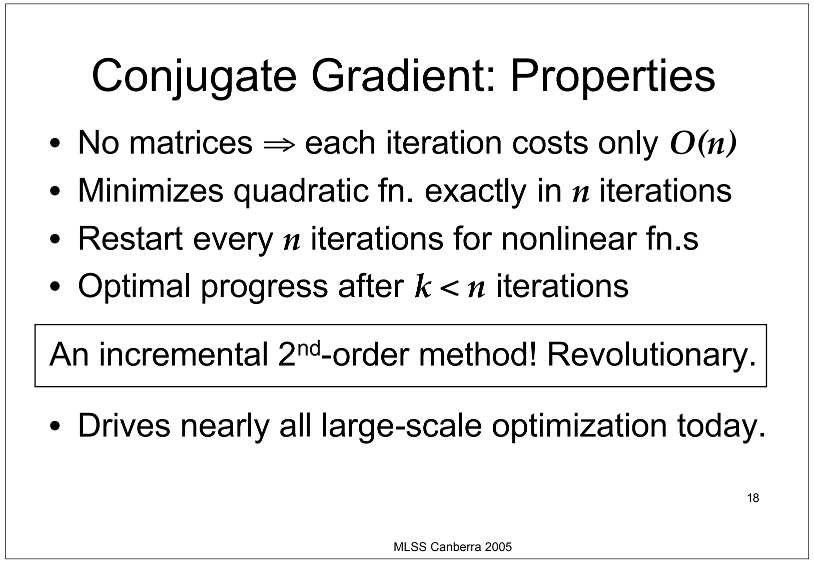
$$\beta = \frac{\vec{g}_t^{\,T}(\vec{g}_t - \vec{g}_{t-1})}{\vec{v}_t^{\,T}(\vec{g}_t - \vec{g}_{t-1})}$$

(Hestenes-Stiefel, 1952) (a.k.a. Beale-Sørenson)

NB: other formulae for β (Polak-Ribiere, Fletcher-Reeves) equivalent for quadratic but inferior for nonlinear fn.s!

17

MLSS Canberra 2005



Conjugate Gradient: Properties

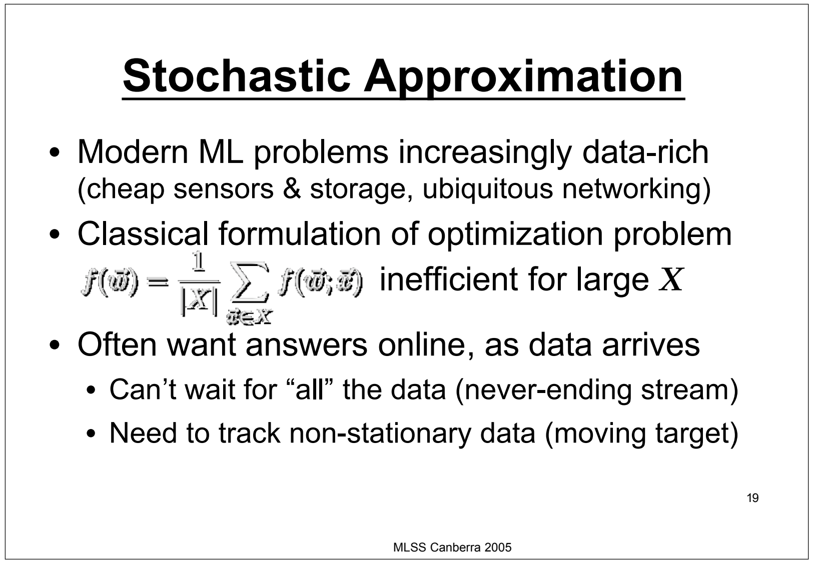
- No matrices ⇒ each iteration costs only $O(n)$
- Minimizes quadratic fn. exactly in n iterations
- Restart every n iterations for nonlinear fn.s
- Optimal progress after $k < n$ iterations

An incremental 2nd-order method! Revolutionary.

- Drives nearly all large-scale optimization today.

18

MLSS Canberra 2005

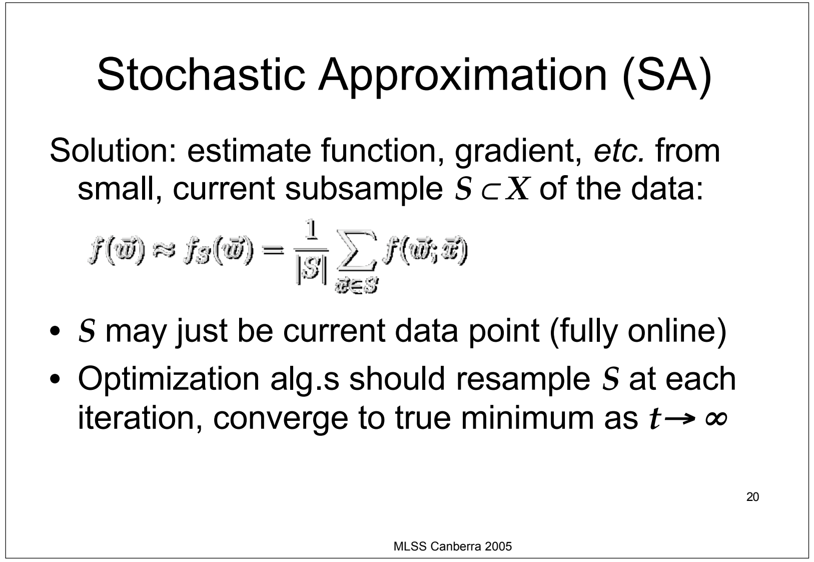


Stochastic Approximation

- Modern ML problems increasingly data-rich (cheap sensors & storage, ubiquitous networking)
- Classical formulation of optimization problem $f(\vec{w}) = \frac{1}{|X|} \sum_{\vec{x} \in X} f(\vec{w}; \vec{x})$ inefficient for large X
- Often want answers online, as data arrives
  - Can't wait for "all" the data (never-ending stream)
  - Need to track non-stationary data (moving target)

19

MLSS Canberra 2005



Stochastic Approximation (SA)

Solution: estimate function, gradient, *etc.* from small, current subsample $S \subset X$ of the data:

$$f(\vec{w}) \approx f_S(\vec{w}) = \frac{1}{|S|} \sum_{\vec{x} \in S} f(\vec{w}; \vec{x})$$

- S may just be current data point (fully online)
- Optimization alg.s should resample S at each iteration, converge to true minimum as $t \to \infty$

20

MLSS Canberra 2005

Houston, we have a problem

The best classical methods can't handle SA.

- Conjugate gradients break down with noise
- Line minimizations (BFGS, CG) are incorrect
- Newton, Levenberg-Marquardt too expensive per iteration for large-scale online operation

21

MLSS Canberra 2005

Extended Kalman Filters

Designed for online operation;
use an adaptive gain matrix: $\vec{w}_{t+1} = \vec{w}_t - P_t \vec{g}_t$

Widely used in signal processing, but

- $O(n^2)$ per iteration: don't scale to large n
- Require an explicit model of stochasticity
  - Assumes Gaussianity, i.i.d., *etc.*
  - Assumes parameters are known

22

MLSS Canberra 2005

Back to Square One

Simple gradient descent works with SA;
proven convergence if step size η_t obeys

$$\sum_{t=0}^{\infty} \eta_t = \infty, \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty$$ (Robbins & Munro)

But convergence too slow to be useful
⇒ try to accelerate such that SA still works

23

MLSS Canberra 2005

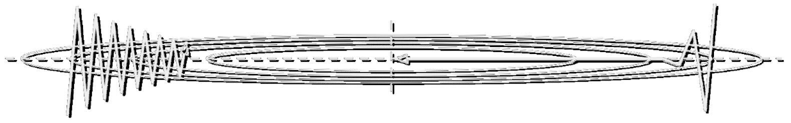
Local Step Sizes

Give each parameter its own step size:

$$\underbrace{\vec{w}_{t+1} = \vec{w}_t - \vec{p}_t \cdot \vec{g}_t}$$

Hadamard product (component-wise)

- Still $O(n)$ per iteration
- $p_i > 0$ ⇒ descent direction
- Act as diagonal conditioner:



24

MLSS Canberra 2005

Adapting Local Step Sizes

Key idea: perform simultaneous gradient descent in step sizes (“meta-descent”):

$$\boxed{\vec{w}_{t+1} = \vec{w}_t - \vec{p}_t \cdot \vec{g}_t}$$

$$\begin{aligned} \vec{p}_{t+1} &= \vec{p}_t - \mu \frac{\partial f(\vec{w}_{t+1})}{\partial \vec{p}_t} \\ &= \vec{p}_t - \mu \frac{\partial f(\vec{w}_{t+1})}{\partial \vec{w}_{t+1}} \cdot \frac{\partial \vec{w}_{t+1}}{\partial \vec{p}_t} \\ &= \vec{p}_t + \underset{\substack{\uparrow \\ \text{meta-step size}}}{\mu}\, \vec{g}_{t+1} \cdot \vec{g}_t \end{aligned}$$

Doesn’t work.

25

MLSS Canberra 2005

Problems, Problems

- p_i can go negative, have small dynamic range
  ⇒ use multiplicative update
- Autocorrelation of stoch. gradient very noisy
  ⇒ replace g_t with running average $\langle g_t \rangle$
- Step size update extremely ill-conditioned
  (condition number κ squares at meta-level!)
  ⇒ normalize gradient autocorrelation
  - radical form of normalization: use sign only

26

MLSS Canberra 2005

Sign-based Methods

We now have $\vec{p}_{t+1} = \vec{p}_t \cdot (1 + \mu\, \mathrm{sign}(\vec{g}_{t+1} \cdot \langle \vec{g}_t \rangle))$

With some variations, this is known as

- Delta-bar-delta (Jacobs 1988)
- Adaptive BP (Silva&Almeida 1990)
- SuperSAB (Tollenaere 1990)
- RPROP (Riedmiller 1993)

Don’t work online.

27

MLSS Canberra 2005

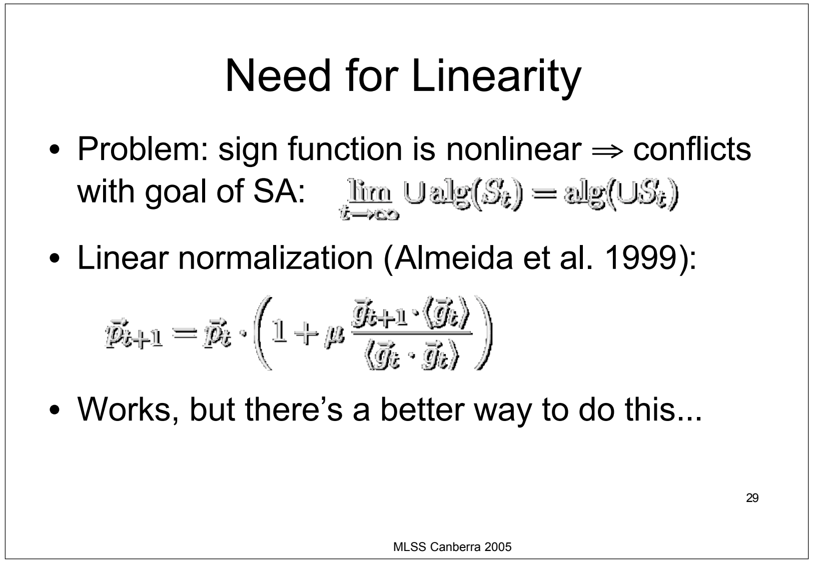
Sign-Based Methods: Problem

Consider 2-way classification task with 10% positives, classifier only learns bias (= d.c. component).

- Let e_t = error at time t. At optimum (output = 0.1)
  $E(e_t) = 0.1 \cdot (1 - 0.1) + 0.9 \cdot (0 - 0.1) = 0$.
- Assume i.i.d. sampling and step size zero. Now
  $E(e_{t+1} \cdot e_t) = E(e_{t+1}) \cdot E(e_t) = 0$
- But: $E(\mathbf{sign}(e_{t+1} \cdot e_t)) = 0.01 + 0.81 - 2 \cdot 0.09 = 0.64$
  ⇒ sign-based method will increase step size
  ⇒ will never anneal to converge on solution

28

MLSS Canberra 2005



Need for Linearity

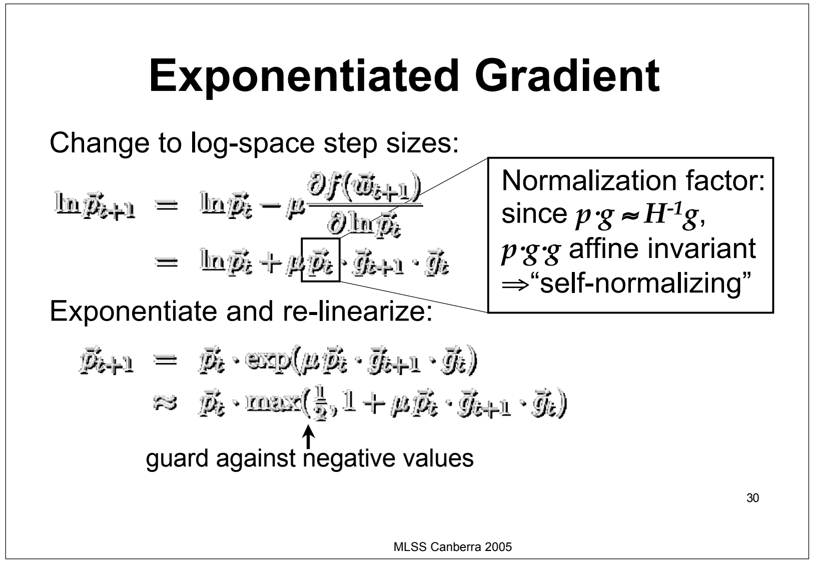
- Problem: sign function is nonlinear ⇒ conflicts with goal of SA: $\lim_{t\to\infty} \cup\, \mathrm{alg}(S_t) = \mathrm{alg}(\cup S_t)$
- Linear normalization (Almeida et al. 1999):

$$\vec{p}_{t+1} = \vec{p}_t \cdot \left(1 + \mu \frac{\vec{g}_{t+1} \cdot \langle \vec{g}_t \rangle}{\langle \vec{g}_t \cdot \vec{g}_t \rangle}\right)$$

- Works, but there's a better way to do this...

29

MLSS Canberra 2005



Exponentiated Gradient

Change to log-space step sizes:

$$\begin{aligned} \ln \vec{p}_{t+1} &= \ln \vec{p}_t - \mu \frac{\partial f(\vec{w}_{t+1})}{\partial \ln \vec{p}_t} \\ &= \ln \vec{p}_t + \mu \boxed{\vec{p}_t} \cdot \vec{g}_{t+1} \cdot \vec{g}_t \end{aligned}$$

Normalization factor: since $p{\cdot}g \approx H^{-1}g$, $p{\cdot}g{\cdot}g$ affine invariant ⇒"self-normalizing"

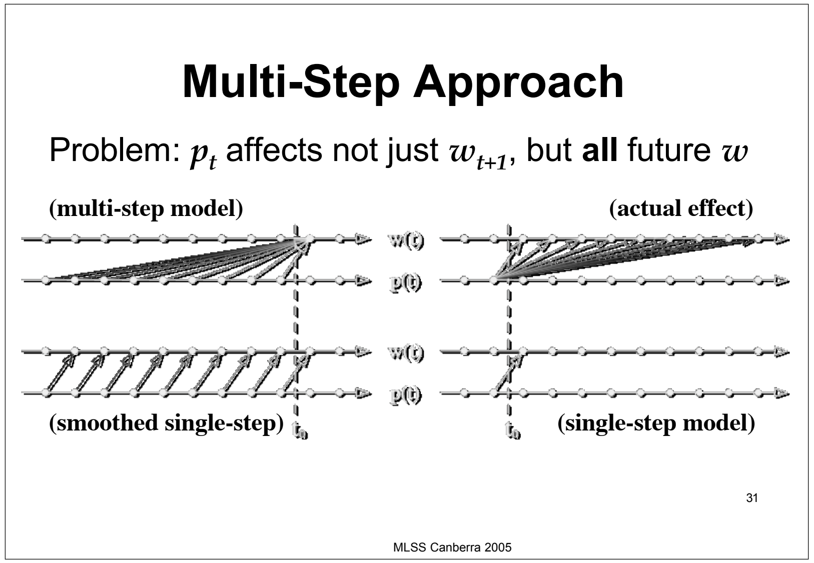
Exponentiate and re-linearize:

$$\begin{aligned} \vec{p}_{t+1} &= \vec{p}_t \cdot \exp(\mu \vec{p}_t \cdot \vec{g}_{t+1} \cdot \vec{g}_t) \\ &\approx \vec{p}_t \cdot \max(\tfrac{1}{2}, 1 + \mu \vec{p}_t \cdot \vec{g}_{t+1} \cdot \vec{g}_t) \end{aligned}$$

guard against negative values

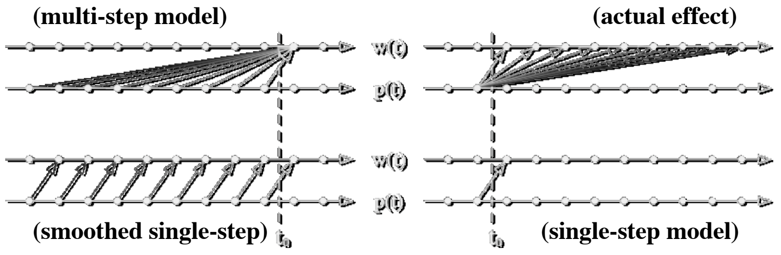
30

MLSS Canberra 2005



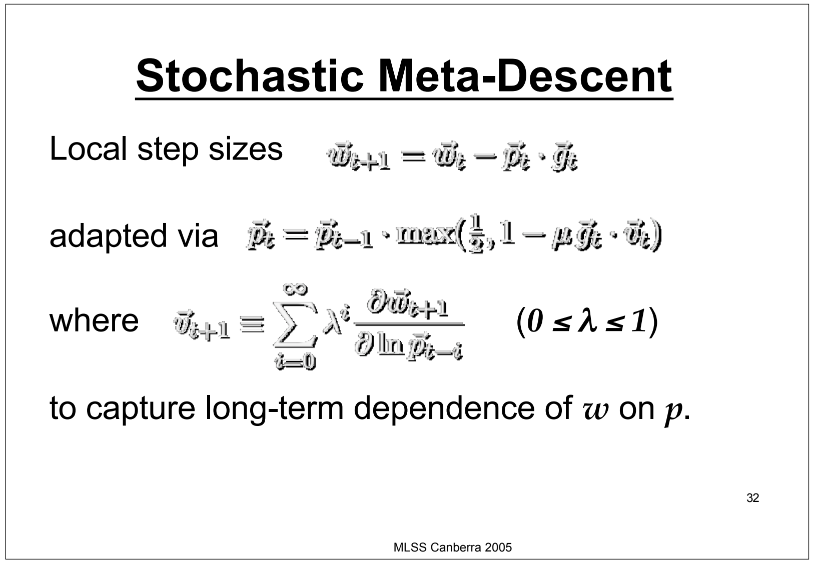
Multi-Step Approach

Problem: p_t affects not just w_{t+1}, but **all** future w



31

MLSS Canberra 2005



Stochastic Meta-Descent

Local step sizes $\vec{w}_{t+1} = \vec{w}_t - \vec{p}_t \cdot \vec{g}_t$

adapted via $\vec{p}_t = \vec{p}_{t-1} \cdot \max(\tfrac{1}{2}, 1 - \mu \vec{g}_t \cdot \vec{v}_t)$

where $\vec{v}_{t+1} \equiv \sum_{i=0}^{\infty} \lambda^i \frac{\partial \vec{w}_{t+1}}{\partial \ln \vec{p}_{t-i}}$ $(0 \le \lambda \le 1)$

to capture long-term dependence of w on p.

32

MLSS Canberra 2005

SMD: The Tricky Bit

$$\begin{aligned}
\vec{v}_{t+1} &= \sum_{i=0}^{\infty} \lambda^i \frac{\partial \vec{w}_t}{\partial \ln \vec{p}_{t-i}} - \sum_{i=0}^{\infty} \lambda^i \frac{\partial (\vec{p}_t \cdot \vec{g}_t)}{\partial \ln \vec{p}_{t-i}} \\
&= \lambda \vec{v}_t - \sum_{i=0}^{\infty} \lambda^i \frac{\partial \vec{p}_t \cdot \vec{g}_t}{\partial \ln \vec{p}_{t-i}} - \sum_{i=0}^{\infty} \lambda^i \frac{\vec{p}_t \cdot \partial \vec{g}_t}{\partial \ln \vec{p}_{t-i}} \\
&\approx \lambda \vec{v}_t - \vec{p}_t \cdot \left(\vec{g}_t + \sum_{i=0}^{\infty} \lambda^i \frac{\partial \vec{g}_t}{\partial \vec{w}_t^T} \frac{\partial \vec{w}_t}{\partial \ln \vec{p}_{t-i}} \right) \\
&= \lambda \vec{v}_t - \vec{p}_t \cdot (\vec{g}_t + \lambda\, H_t \vec{v}_t) \qquad \text{whew!}
\end{aligned}$$

33

MLSS Canberra 2005

SMD: The v Update

$$\vec{v}_{t+1} = \lambda \vec{v}_t - \vec{p}_t \cdot (\vec{g}_t + \lambda H_t \vec{v}_t)$$

- We obtain a simple iterative update for v
- Closely related to $\mathrm{TD}(\lambda)$ reinf. learning (Sutton)
- $H_t v_t$ can be computed as efficiently as two gradient eval.s (typically $O(n)$ - more later)
- Predecessors (IDBD, K1, ELK1) diagonalized H; here we have full Hessian at no extra cost!

34

MLSS Canberra 2005

SMD: Fixpoint of v

- Fixpoint of $\vec{v}_{t+1} = \lambda \vec{v}_t - \vec{p}_t \cdot (\vec{g}_t + \lambda H_t \vec{v}_t)$ is Levenberg-Marquardt style gradient step:

$$\vec{v} \;\rightarrow\; -\left[\lambda H + (1-\lambda)\mathrm{diag}(\tfrac{1}{\vec{p}})\right]^{-1} \vec{g}$$

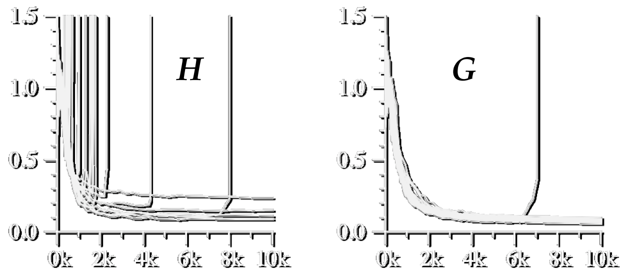
- $v{\cdot}g$ affine invariant at fixpoint (normalization)
- v too noisy for direct use (Orr & Leen); SMD stable due to double integration $v \rightarrow p \rightarrow w$

35

MLSS Canberra 2005

SMD: Gauss-Newton

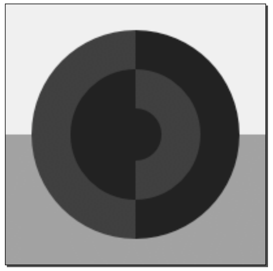
- SMD uses $\boxed{\text{Gauss-Newton}}$ approximation of Hessian for improved stability
- Fast Gv product (even a bit faster than Hv)

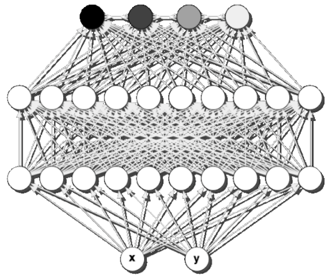


36

MLSS Canberra 2005

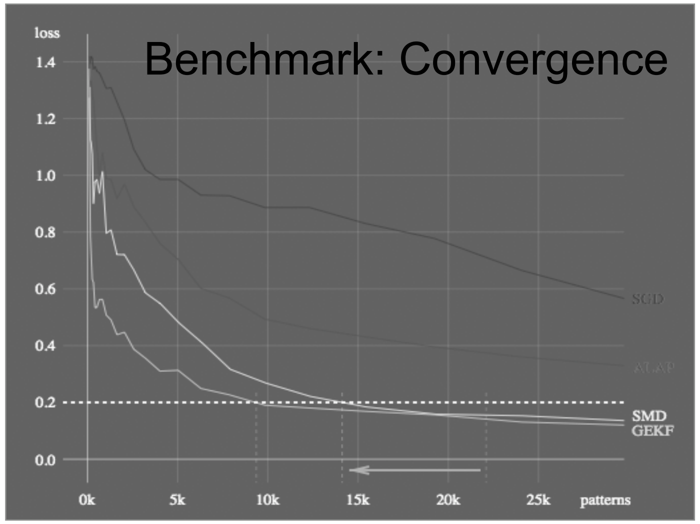
Four Regions Benchmark





Compare simple stoch. gradient (SGD), conventional step size adaptation (ALAP), stochastic meta-descent (SMD), and global extended Kalman filtering (GEKF).

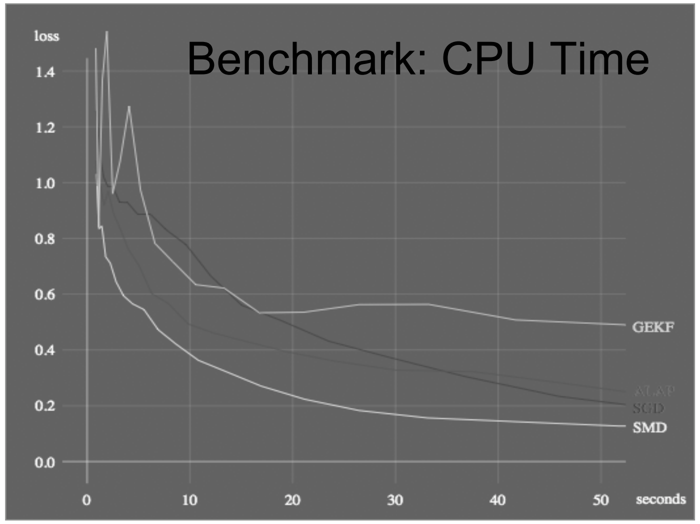
Benchmark: Convergence



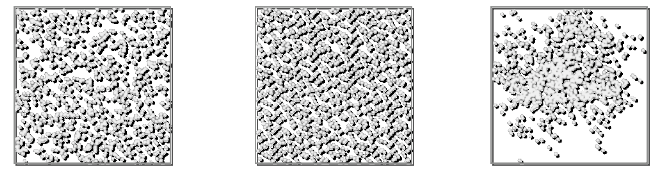
Benchmark: Cost Comparison

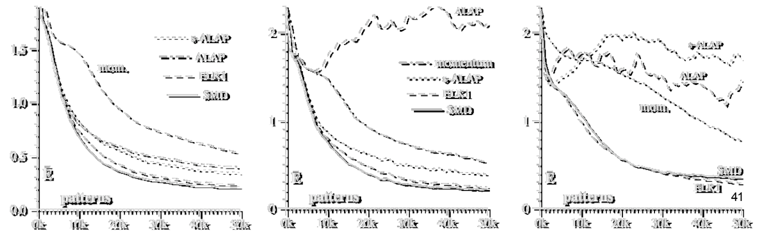
Algorithm	storage / weight	flops / update	CPU ms / pattern
SGD	**1**	**6**	**0.5**
SMD	**3**	**18**	**1.0**
ALAP	**4**	**18**	**1.0**
GEKF	**>90**	**>1500**	**40**

Benchmark: CPU Time



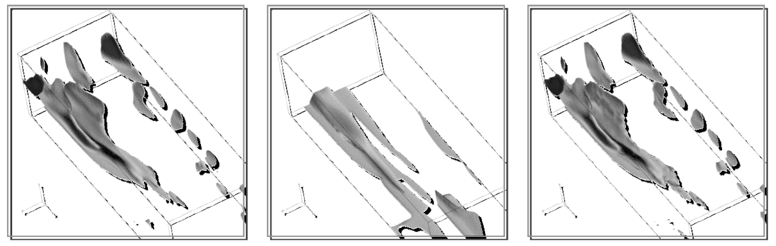
Benchmark: Autocorrelated Data


i.i.d. uniform   Sobol   Brownian



Application: Turbulent Flow

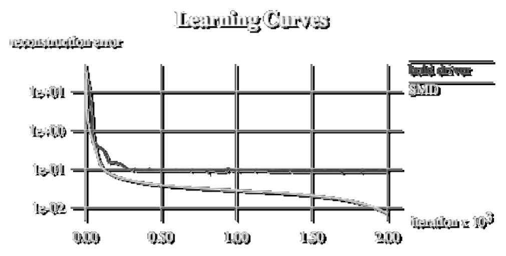
(with M. Milano, Inst. of Comput. Science, ETH Zürich)


original simulation (75'000 d.o.f.)   linear PCA (160 p.c.)   neural network (160 nonlin. p.c.)

Application: Turbulent Flow

- 15 neural nets, each about 180'000 weights
- generic model has over 20'000'000 weights!

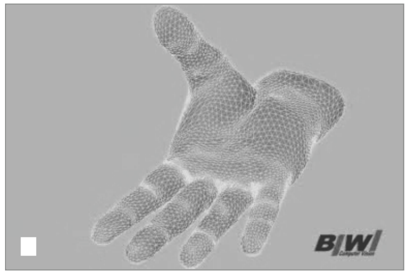
Here SMD
- outperformed Matlab toolbox
- able to train generic model

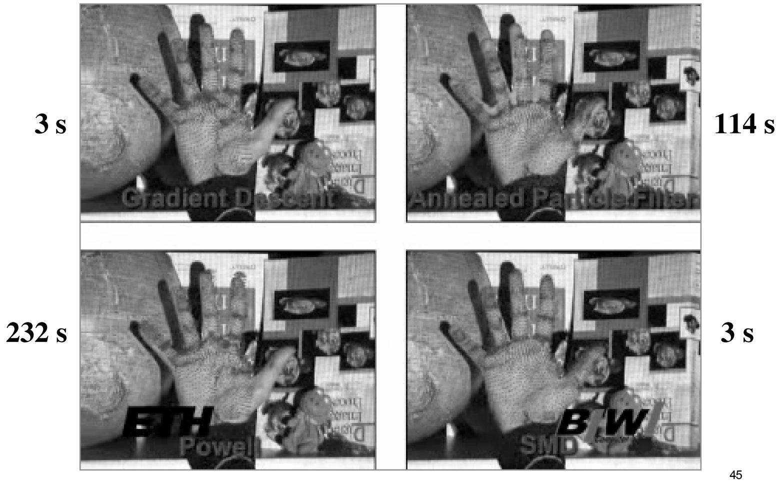


Application II: Hand Tracking

(with M. Bray & L. van Gool, Computer Vision Lab, ETH Zürich)

- Detailed hand model (10k vertices, 23 d.o.f.)
- Randomly sample a few points on model surface
- Project them to image
- Compare with camera image at these points
- Use resulting stoch. gradient to adjust model via SMD





45

MLSS Canberra 2005

Hand Tracking: Results

- SMD: 40-fold speed-up over state of the art
- Speed-up due to stochastic approximation
- Stochasticity helps escape local minima
  $\Rightarrow$ better tracking performance

Work continues at ETH, Oxford, and NICTA:

- Multiple, ordinary video cameras, occlusions
- Real-time tracking of hands, face, body, …

46

MLSS Canberra 2005

SA, SMD: Summary

- Data-rich ML problems need SA for efficiency
- Classical gradient methods don’t work with SA
- Like CG, SMD combines
  - Extreme scalability: cheap $O(n)$ iterations
  - Efficiency: rapid (superlinear) convergence
- Unlike CG, SMD designed to work with SA
- 2nd order without the cost (fast Hv product)

47

MLSS Canberra 2005

ANGie Project

ANGie project will explore SMD at NICTA:

- Mathematical analysis (stability, convergence)
- Further development of the core algorithm
- Development of AD tools & techniques
- Use of SMD in different ML settings
  (kernels, graphical models, RL, control, …)
- Reference applications (computer vision, …)
- **Jobs available** (postdoc, Ph.D.)

48

MLSS Canberra 2005

Algorithmic Differentiation (AD)

- *a.k.a.* automatic differentiation (www.autodiff.org)
- Given (code for) a diff'able function, produces (code for) derivative function(s) automatically
- Solves major software engineering problem by ensuring correctness of derivative code
- Textbook (SIAM 2000): Griewanck, Evaluating Derivatives: Principles & Techniques of AD

49

MLSS Canberra 2005

Differentiation Strategies

- Symbolic: $\sin' = \cos$ $\quad d(M^{-1}) = -M^{-1}(dM)M^{-1}$
  - Transformation of symbolic algebraic expressions
  - Knowledge-intensive; goal is mathematical insight
- Numeric: $\frac{\partial}{\partial x_i} f(\vec{x}) = \lim_{h \to 0} [f(\vec{x} + h\vec{e}_i) - f(\vec{x})]/h$
  - **Knowledge-free**; goal is just numerical result
  - **Approximate**; choice of step h is problematic
  - **Inefficient** for calculating high-dim. gradients (approximates only forward mode of AD)

50

MLSS Canberra 2005

Differentiation Strategies

- Algorithmic:
  - Low-level symbolic diff. for numeric purposes
  - Transformation & evaluation of algebraic code
  - Differentiate high-level constructs by way of their implementation in terms of lower-level primitives
  - Exact and efficient. Two modes: given $\vec{y} = f(\vec{x})$
    - Forward mode calculates $d\vec{y} = J_f \, d\vec{x}$ (perturbation)
    - Reverse mode calculates $\frac{\partial}{\partial \vec{x}} = J_f^T \frac{\partial}{\partial \vec{y}}$ (gradient)

51

MLSS Canberra 2005

Forward Mode

- Propagates perturbations forward through the tangent linear system: $d\vec{y} = J_f \, d\vec{x}$
- Basic rules:
  - Sums: $c = a + b \;\Rightarrow\; \mathrm{d}c = \mathrm{d}a + \mathrm{d}b$
  - Products: $c = a \cdot b \;\Rightarrow\; \mathrm{d}c = a \cdot \mathrm{d}b + \mathrm{d}a \cdot b$
  - Chain rule: $c = f(a) \;\Rightarrow\; \mathrm{d}c = f'(a) \cdot \mathrm{d}a$
- Higher-level (math library, linear algebra, …) rules can be added to increase efficiency

52

MLSS Canberra 2005

Forward Mode: Implementation

Straightforward since control flow is unchanged.

Many ways to do it -

- Source transformation:

```
a*=b;  ⇒  a*=b;
           da*=b;
           da+=a*db;
```

- Byte code transformation
- Augmented byte code interpreter
- Overloaded C++ class (available on request)
- Or just use complex number library…

53

MLSS Canberra 2005

Forward Mode: Implementation

Complex arithmetic can perform forward AD!

Consider complex $(x, \varepsilon \cdot dx)$, where $\varepsilon = 10^{-150}$:

- Sums: $(a, \varepsilon \cdot \mathrm{d}a) + (b, \varepsilon \cdot \mathrm{d}b) = (a+b, \varepsilon \cdot (\mathrm{d}a + \mathrm{d}b))$ √
- Products:
  $(a, \varepsilon \cdot \mathrm{d}a)(b, \varepsilon \cdot \mathrm{d}b) = (ab - \varepsilon^2 \cdot \mathrm{d}a \cdot \mathrm{d}b, \varepsilon \cdot (a \cdot \mathrm{d}b + \mathrm{d}a \cdot b))$
  √ $\nwarrow 10^{-300} \approx 0$ √

Very fast and usually accurate enough - but can't use this trick for complex numbers…

54

MLSS Canberra 2005

Calculating Gradients

- Gradient of scalar fn. = transposed Jacobian: $\vec{g} = \frac{\partial f(\vec{w})}{\partial \vec{w}} = J_f^T$
- Can calculate individual elements by forward AD: $(\vec{g})_i = (J_f^T)_i = J_f \vec{e}_i$
- n iterations for gradient
  ⇒ inefficient for high-dim. systems (large n)
- Reverse mode AD obtains gradient efficiently (single iteration) but is harder to implement

55

MLSS Canberra 2005

Reverse Mode

- Propagates gradients back through the adjoint system: $\frac{\partial}{\partial \vec{x}} = J_f^T \frac{\partial}{\partial \vec{y}}$
- Gradient of scalar fn.: $\frac{\partial f(\vec{x})}{\partial \vec{x}} = J_f^T \frac{\partial f(\vec{x})}{\partial \vec{y}} = J_f^T 1$
- For neural nets, known as backprop(agation)
- Requires reversal of fn.'s dataflow. Need to
  - Memorize dataflow & intermediate results
  - Unroll loops, overwrites, *etc.* Hard to do well!

56

MLSS Canberra 2005

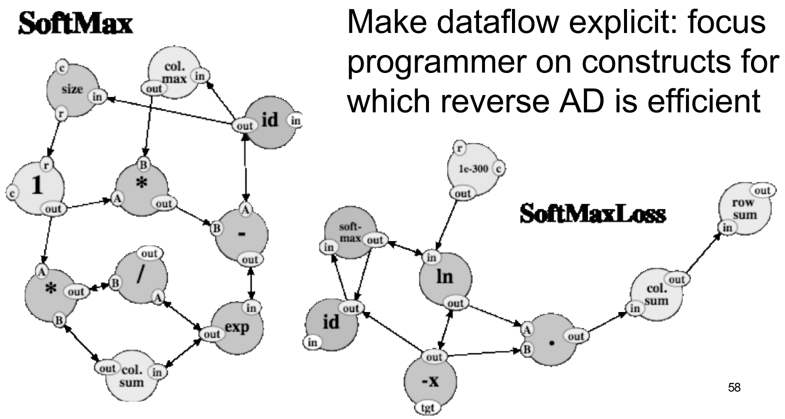
Reverse Mode: Rules

- Similar to forward mode:
  - Sums: $c = a + b \Rightarrow \frac{\partial}{\partial a} = \frac{\partial}{\partial c}; \frac{\partial}{\partial b} = \frac{\partial}{\partial c}$
    Forks: $b = a; c = a \Rightarrow \frac{\partial}{\partial a} = \frac{\partial}{\partial b} + \frac{\partial}{\partial c}$
  - Products: $c = a \cdot b \Rightarrow \frac{\partial}{\partial a} = \frac{\partial}{\partial c} b; \frac{\partial}{\partial b} = a \frac{\partial}{\partial c}$
  - Chain rule: $c = f(a) \Rightarrow \frac{\partial}{\partial a} = f'(a) \frac{\partial}{\partial c}$
- Higher-level rules essential for efficiency
- Ongoing research, *e.g.* reverse AD of fixed-point iterations without unrolling (Pearlmutter 2004)

57

MLSS Canberra 2005

Visual Dataflow Programming



Make dataflow explicit: focus programmer on constructs for which reverse AD is efficient

58

MLSS Canberra 2005

Fast Hessian-Vector Product

Applying forward mode to gradient code:

$$J_{\vec g}\vec v = \frac{\partial \vec g(\vec x)}{\partial \vec x}\vec v = \frac{\partial^2 f(\vec x)}{(\partial \vec x)^2}\vec v = H_f \vec v$$

gives product of H_f with arbitrary vector v.

- As fast as 2-3 gradient evaluations; usually $O(n)$ - even though H_f is $n \times n$ matrix!
- Similar trick for Gauss-Newton approximation:

$$G_f \vec v = \underline{J_{\vec m}^T \underline{H_l \underline{J_{\vec m}\vec v}}}$$

forward mode
Hv product
reverse mode

59

MLSS Canberra 2005

Course Summary

1. **Classical Gradient Methods**
   Direct (gradient-free), Steepest Descent, Newton, Levenberg-Marquardt, BFGS, Conjugate Gradient
2. **Stochastic Approximation** (SA)
   Why necessary, why difficult. Step size adaptation.
3. **Stochastic Meta-Descent** (SMD)
   Advanced stochastic step size adaptation method.
4. **Algorithmic Differentiation** (AD)
   Forward/reverse mode. Fast Hessian-vector products.

60

MLSS Canberra 2005