# Large Scale Learning[1]

David Barber

[1] These slides accompany the book *Bayesian Reasoning and Machine Learning*. The book and demos can be downloaded from `www.cs.ucl.ac.uk/staff/D.Barber/brml`. Feedback and corrections are also available on the site. Feel free to adapt these slides for your own purposes, but please include a link the above website.

## Large Scale Problems

### Stochastic Gradient Descent

SGD is useful in the situations:

- The dataset is too big to store all of it.
- The dataset is so big that it will take a long time to take an update

---

### Exploiting Sparsity

- We will also consider some special linear models that we can solve efficiently by exploiting any sparsity that might exist in the problem.

## Batch vs Online

### Batch

- Compute the gradient by first summing over all the training data inputs.
- Then do a gradient update.

---

### Online

- Datapoints arrive in a stream
- Compute approximate gradient by summing over a single datapoint.
- Then do a gradient update immediately for this datapoint.

## Linear Models

## Linear regression

Least squares error:

$$E(\boldsymbol{\theta}) = \sum_n \left(y^n - \boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n\right)^2 + \lambda\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta}$$

The optimum is given when the gradient wrt $\boldsymbol{\theta}$ is zero:

$$\frac{\partial E}{\partial \theta_i} = -2\sum_n \left(y^n - \boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n\right) x_i^n + 2\lambda\theta_i = 0$$

$$\underbrace{\sum_n y^n x_i^n}_{b_i} = \sum_j \left(\underbrace{\lambda\delta_{i,j} + \sum_n x_i^n x_j^n}_{A_{ij}}\right)\theta_j$$

Hence, in matrix form, this is

$$\mathbf{b} = \mathbf{A}\boldsymbol{\theta}, \rightarrow \boldsymbol{\theta} = \mathbf{A}^{-1}\mathbf{b}$$

which is a simple linear system that can be solved in $O\left((\dim\boldsymbol{\theta})^3\right)$ time.

## Sparse Data

- Let $D = \dim \boldsymbol{\theta}$.
- Storing $\mathbf{A}$ scales $O\left(D^2\right)$ – very expensive for large $D$.
- Note that the gradient

$$\frac{\partial E}{\partial \theta_i} = -2 \sum_n \left( y^n - \boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^n \right) x_i^n + 2\lambda\theta_i$$

  Can be computed without storing $\mathbf{A}$.
- Suggests that we can use gradient based methods to find $\boldsymbol{\theta}$.
- If each $\mathbf{x}$ is itself sparse, with density factor $0 \leq d \leq 1$ (so that on average each $\mathbf{x}$ contains only $dD$ non zero entries), computing the gradient takes $O\left(dDN\right)$ time and order $O\left(N + D\right)$ storage.

## Online Stochastic Gradient Descent

We can consider the equivalent rescaled objective

$$\frac{1}{N}\sum_n \left(y^n - \boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n\right)^2 + \gamma\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta}$$

where $\gamma = \lambda/N$, with gradient

$$\mathbf{g} = -2\frac{1}{N}\sum_n \left(y^n - \boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n\right)\mathbf{x}^n + 2\gamma\boldsymbol{\theta}$$

- We can view the first term as the expectation wrt the empirical distribution.
- By approximating the empirical distribution by a limited number of samples $N'$, we can form the approximation

$$\mathbf{g} \approx -2\frac{1}{N'}\sum_{n'} \left(y^{n'} - \boldsymbol{\theta}^\mathsf{T}\mathbf{x}^{n'}\right)\mathbf{x}^{n'} + 2\gamma\boldsymbol{\theta}$$

- Then do a gradient update.
- Only requires that $N'$ samples from the dataset $\mathbf{X}$ are available for each gradient calculation.

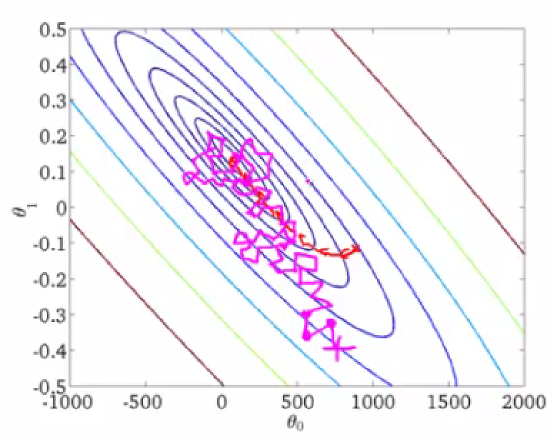
## Online Stochastic Gradient Descent

- In the extreme case we can assume that only a stream of input-output pairs $\mathbf{x}^n, y^n$ is available (we don't store these pairs – they just 'appear').
- Then as each datapoint $n$ arrives, we can compute a stochastic approximation to the gradient

$$\mathbf{g} \approx 2\left(y^n - \boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n\right)\mathbf{x}^n + 2\gamma\boldsymbol{\theta}$$

- Then do a gradient update, $\theta^{new} = \theta - \epsilon\mathbf{g}$.
- Provided that the gradient step is small, on average we will tend to take a gradient in roughly the same direction as the batch update.
- Many very large (Google scale) 'Big Data' algorithms use Stochastic Gradient Descent with data stored on different machines.
- The stochastic nature makes the algorithm robust to synchronisation issues in parallel implementations.
- Vowpal Wabbit and similar tools use SGD to train very large linear predictors.

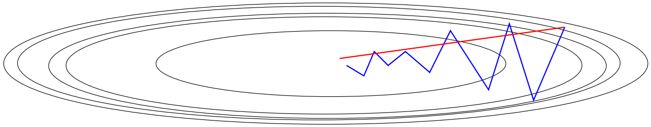
# Stochastic Gradient Descent



In red is the path of standard gradient descent, in magenta the path of stochastic gradient descent. Eventually they both converge to the true minimum. Thanks to Alex Holehouse for the picture.

## Higher Order Methods

- Gradient Descent only makes small local moves and is thus slow to converge.
- Higher order methods combine better search directions with line search to improve convergence.
- Efficient ways to exploit sparsity in the feature (input) vector $\mathbf{x}$ to perform regression on very large datasets with very high dimensional $\mathbf{x}$.



- In blue is the standard gradient descent path and in red is the conjugate gradients (with line search) path.
- For a quadratic surface, conjugate gradients is much faster than standard gradient descent.

## Conjugate Gradients

Conjugate gradients is an efficient way to minimise a function $f(\boldsymbol{\theta})$

```
 1: $k = 1$
 2: Choose $\boldsymbol{\theta}_1$.
 3: $\mathbf{p}_1 = -\mathbf{g}_1$
 4: **while** $\mathbf{g}_k \neq \mathbf{0}$ **do**
 5:     $\alpha_k = \underset{\alpha_k}{\operatorname{argmin}}\ f(\boldsymbol{\theta}_k + \alpha_k \mathbf{p}_k)$        ▷ Line Search
 6:     $\boldsymbol{\theta}_{k+1} := \boldsymbol{\theta}_k + \alpha_k \mathbf{p}_k$
 7:     $\beta_k := \mathbf{g}_{k+1}^\mathsf{T} \mathbf{g}_{k+1} / (\mathbf{g}_k^\mathsf{T} \mathbf{g}_k)$
 8:     $\mathbf{p}_{k+1} := -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k$
 9:     $k = k + 1$
10: **end while**
```

The algorithm is particularly useful in the case of quadratic functions $f$ since then the line search can be carried out exactly.

## Line Search for a Quadratic

Given a quadratic

$$f(\mathbf{x}) \equiv \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{b}$$

we wish to find $\lambda$ that minimises along the line direction $\mathbf{p}$:

$$\min_{\lambda} f(\mathbf{x} + \lambda\mathbf{p})$$

$$\begin{aligned} f(\mathbf{x} + \lambda\mathbf{p}) &= \frac{1}{2}\left(\mathbf{x} + \lambda\mathbf{p}\right)^\mathsf{T}\mathbf{A}\left(\mathbf{x} + \lambda\mathbf{p}\right) - \mathbf{b}^\mathsf{T}\left(\mathbf{x} + \lambda\mathbf{p}\right) \\ &= \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + \frac{1}{2}\lambda^2\mathbf{p}^\mathsf{T}\mathbf{A}\mathbf{p} + \lambda\left(\mathbf{p}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{p}\right) \end{aligned}$$

Differentiating wrt $\lambda$ and setting to 0, we obtain

$$\lambda = -\frac{\mathbf{p}^\mathsf{T}\left(\mathbf{A}\mathbf{x} - \mathbf{b}\right)}{\mathbf{p}^\mathsf{T}\mathbf{A}\mathbf{p}}$$

## Conjugate Gradients for the quadratic form

```
function [x, val]=conjgrad(A,b,x0,opts)
%CONJGRAD conjugate gradients for minimising 0.5*x'*A*x-x'*b
% for positive definite A
% [x, val]=conjgrad(A,b,x0,opts)
% x0 is the initial starting solution
x=x0;
g=A*x-b;
p=-g;
valOld=realmax;
for loop=1:opts.maxits
    alpha=-(p'*g)/(p'*A*p);
    x=x+alpha*p;
    gnew=A*x-b;
    beta=(gnew'*gnew)/(g'*g);
    p=-gnew+beta*p;
    g=gnew;
    val = 0.5*x'*(g-b);
    if loop>1; if valOld-val<opts.tol; break; end;  end
    valOld=val;
end
```

## Conjugate Gradients for Linear Regression

$$\frac{1}{2}\sum_{n}\left(y^{n}-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^{n}\right)^{2}+\frac{1}{2}\lambda\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{\theta}=\frac{1}{2}\boldsymbol{\theta}^{\mathsf{T}}\mathbf{A}\boldsymbol{\theta}-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{b}$$

where

$$\mathbf{A}=\lambda\mathbf{I}+\sum_{n}\mathbf{x}^{n}\left(\mathbf{x}^{n}\right)^{\mathsf{T}}, \qquad \mathbf{b}=\sum_{n}y^{n}\mathbf{x}^{n}$$

- Note that only place the problematic $\mathbf{A}$ appears in the CG algorithm is in the term

$$\mathbf{p}^{\mathsf{T}}\mathbf{A}\mathbf{p}=\lambda\mathbf{p}^{2}+\sum_{n}\left(\mathbf{p}^{\mathsf{T}}\mathbf{x}^{n}\right)^{2}$$

- This means that we can run (batch) CG without computing $\mathbf{A}$.

## Conjugate Gradients for Sparse Linear Regression

```
function [w, valCG]=conjgradSparseLinReg(X,y,w0,lambda,opts)
%CONJGRADSPARSELINREG conjudate gradients for minimising:
% sum_n [(y(n)-w'*X(:,n)]^2 +lambda*w'*w
% (Trivial Initialisation code lines removed)
% MAIN LOOP:
for loop=1:opts.maxits
    tmp=lambda*sum(p.*p);
    for n=1:N
        tmp=tmp+(sum(X(:,n).*p))^2;
    end
    alpha=-(p'*g)/tmp;
    w=w+alpha*p;
    gnew=-b+lambda*w;
    for n=1:N
        gnew=gnew+X(:,n)*(sum(X(:,n).*w));
    end
    beta=(gnew'*gnew)/(g'*g);
    p=-gnew+beta*p;
    g=gnew;
    val = 0.5*w'*(g-b);
    if loop>1; if valOld-val<opts.tol; break; end; end
    valOld=val;
```

## Sparse Linear Learning

- `demoLargeScaleLinearRegression.m` shows how to quickly perform linear regression for very high dimensional sparse data.
- Sparse Linear Learning can also easily be applied to classification, for example logistic regression ●
- An advantage of these simple models is that they are easily parallelisable ●
- Distributed data can be handled easily ●
- Stochastic Gradient Descent is a popular and simple mechanism to make parallel algorithms robust to synchronistation issues ●