Visual Analytics
Lectorial week 10: Optimisation

Ian T. Nabney

University of BRISTOL

Summary

- Understand the two-stage approach to multivariate optimisation.
- Understand the properties and drawbacks of gradient descent.
- Understand the place conjugate directions in second order optimisation methods.
- Know of conjugate gradients, quasi-Newton and scaled conjugate gradient algorithms.
- Describe the strengths and weaknesses of stochastic methods.

Further reading: Nabney chapter 2.

Optimisation

- The problem of (unconstrained) optimisation can be easily stated. You are given a function $f : \mathbb{R}^n \to \mathbb{R}$ and you want to find those $\mathbf{x} \in \mathbb{R}^n$ where f takes on a maximum or minimum value, and evaluate f at those points.
- As maximising f is equivalent to minimising $-f$, we shall just consider minimisation.
- The techniques that we use find local minima; there is little known about how to find global minima, although it is clear that stochastic methods do help.
- Note that in general, simply evaluating f' and finding a zero of this function is a bad idea: lots of critical points aren't minima.

Optimisation Strategy: $\mathbb{R}^n$

- In multi-dimensional search spaces, methods that use derivative information can be very useful to explore the geometry of the function.
- From a given point, there are infinitely many directions to search in: a single dot product $\nabla f^T \mathbf{v}$ determines whether moving in the direction $\mathbf{v}$ is uphill or downhill. This dot product involves just $O(n)$ operations.
- In one dimension, there are just two directions to move in, so evaluating f is as good as evaluating f'.
- We usually use a joint strategy:
  1. choose search direction;
  2. use one-dimensional algorithm to minimise in given direction.

One-dimensional Minimisation

Methods can be classified as

dissection e.g. golden section;

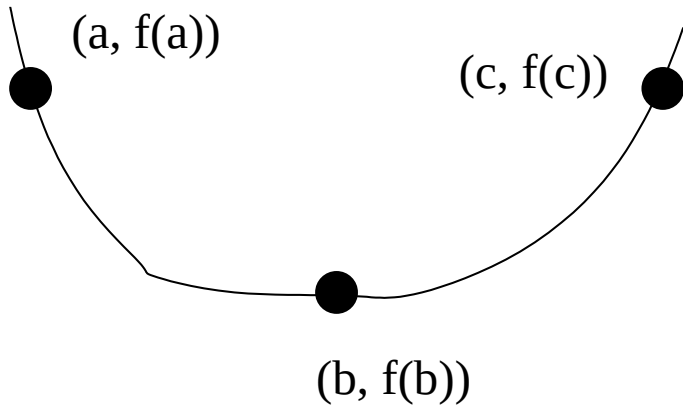
fitted polynomial e.g. quadratic interpolation and Brent's method.

The first step is to work out what precision can reasonably be attained. This will tell us when there's no point in continuing the algorithm. The answer is $\sqrt{\epsilon}$, where ϵ is machine precision. For double precision this is about $1 \cdot 5 \times 10^{-8}$.

Bracketing a Minimum

- A bracketing interval for finding a zero is a pair of values $a < b$ such that $f(a).f(b) < 0$ (i.e. the two function values have opposite sign).
- The equivalent for function minimisation is a triple $a < b < c$ such that

$$f(b) < f(a) \qquad \text{and} \qquad f(b) < f(c)$$



Multivariate Optimisation

- We shall use gradient information to select a succession of 'good' directions to search in.
- One dimensional methods can then be used to find a minimum along a particular direction. Some adaptation of these algorithms is worthwhile: for example, we have a preferred direction to move in along the line (downhill from the current search point).
- For neural networks there is another argument in favour of using analytical derivatives. In the quadratic approximation to the error function

$$E(\mathbf{w}) = E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^T \mathbf{g} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

there are $W(W+3)/2$ terms. Without using gradients, $O(W^2)$ function evaluations, each requiring $O(W)$ steps, are needed. Using backpropagation, evaluating the gradient takes $O(W)$ steps

University of BRISTOL

Algorithm Structure

- The search techniques that we consider are iterative, i.e. we proceed towards the minimum $\mathbf{x}^*$ by a sequence of steps.
- On the jth step we take a step of length α_j in the direction $\mathbf{d}_j$,

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{d}_j \tag{1}$$

  The length of the step can either be chosen using prior knowledge, or by carrying out a line search.
- It is the way that $\mathbf{d}_j$ is chosen that distinguishes the different algorithms.

Batch Gradient Descent

- Perhaps the simplest choice for $\mathbf{d}_j$ is to set it equal to $-\mathbf{g}_j$, i.e. to go down the steepest direction available at $\mathbf{x}_j$.
- We can guarantee that $f(\mathbf{x}_{j+1}) < f(\mathbf{x}_j)$, as long as the parameter α_j (called the learning rate) is chosen to be small enough.

$$f(\mathbf{x}_j + \alpha_j \mathbf{d}_j) \simeq f(\mathbf{x}_j) + \alpha_j \mathbf{g}_j^T \mathbf{d}_j. \quad (2)$$

  With $\mathbf{d}_j = -\mathbf{g}_j$ and for small α_j, this guarantees a reduction in f.
- The NETLAB function `graddesc` implements a batch gradient descent algorithm.

Drawbacks of Gradient Descent

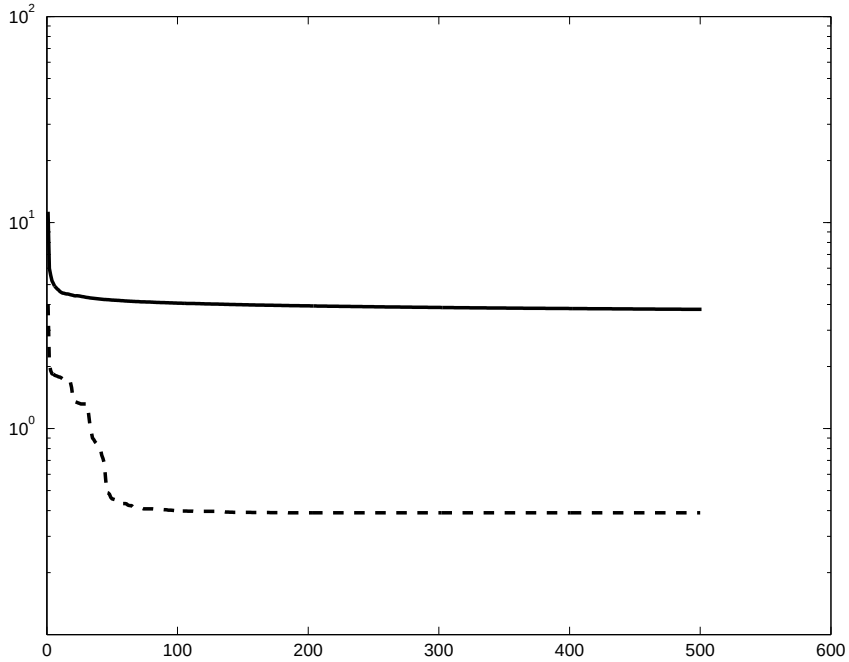
- No principled way of choosing learning rate (or momentum).
- Oscillations in search direction, particularly with line search.
- Behaviour is entirely dependent on the scaling of the variables. If we rescale each x-coordinate by a different amount, so that $X_i = \lambda_i x_i$, then

$$\frac{\partial f}{\partial X_i} = \frac{1}{\lambda_i}\frac{\partial f}{\partial x_i} \tag{3}$$

  and thus the direction of $\mathbf{g}$ can be altered arbitrarily by changing $\boldsymbol{\lambda}$.

It is included in NETLAB for purposes of comparison and for historical reasons, not for serious use.

And It's Bloody Slow



Learning curves for an MLP applied to a simple regression problem: on-line gradient descent (*solid line*) and scaled conjugate gradient (*dashed line*).

Conjugate Directions

- After a line minimisation, the new gradient is orthogonal to the line search direction. So in steepest descent, search directions will always be orthogonal.
- The solution to this problem is to choose the next search direction so that we do not 'spoil' the minimization that we have just carried out. This can be achieved by making sure that the component of the gradient parallel to the previous search direction, which has just been made zero, is unaltered (to lowest order).
- If $\mathbf{d}_j$ represents the new direction, we want

$$\left[\frac{\partial}{\partial \alpha}\left(\nabla f(\mathbf{x}_j + \alpha \mathbf{d}_j)\right)\right]^T \mathbf{d}_{j-1} = 0 \tag{4}$$

Conjugate Directions II

- Using a second order Taylor series expansion of f around $\mathbf{x}_j$, we have, for $\mathbf{x} = \mathbf{x}_j + \alpha \mathbf{d}_j$,

$$f(\mathbf{x}) \approx f(\mathbf{x}_j) + \nabla f(\mathbf{x}_j)^T(\mathbf{x} - \mathbf{x}_j) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_j)^T \mathbf{H}(\mathbf{x} - \mathbf{x}_j)$$
$$= f(\mathbf{x}_j) + \alpha \nabla f(\mathbf{x}_j)^T \mathbf{d}_j + \frac{\alpha^2}{2} \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j \quad (5)$$

- Similarly, expanding ∇f to first order around $\mathbf{x}_j$,

$$\nabla f(\mathbf{x}) = \nabla f(\mathbf{x}_j) + \mathbf{H}(\mathbf{x} - \mathbf{x}_j) = \nabla f(\mathbf{x}_j) + \alpha \mathbf{H} \mathbf{d}_j \quad (6)$$

- From this we conclude that $\frac{\partial}{\partial \alpha} (\nabla f(\mathbf{x})) = \mathbf{H}\mathbf{d}_j$ so for this to be perpendicular to $\mathbf{d}_{j-1}$ we require $\mathbf{d}_j^T \mathbf{H} \mathbf{d}_{j-1} = 0$.
- We say that $\mathbf{d}_j$ is conjugate to $\mathbf{d}_{j-1}$.

Conjugate Directions III

- If f is quadratic, then this relation holds for arbitrary values of α, since the second order Taylor series expansion is exact and $\mathbf{H}$ is constant.
- Another way of viewing this is that we want the search directions to be orthogonal not with respect to the usual Euclidean inner product, but with respect to the inner product defined by the curvature of the function at the minimum. In this light, our conclusion seems logical (not to say obvious).
- Because $\mathbf{H}$ is a symmetric positive-definite matrix, in $\mathbb{R}^n$ we can choose n mutually conjugate directions (for example, the eigenvectors of $\mathbf{H}$).

Conjugate Directions and Optimisation

- If f is quadratic and we minimise along each of these directions successively, then we will reach the minimum of f in n steps.
- Compare this with gradient descent!
- This is because the set of conjugate directions form a basis, and hence the vector from the start point to the minimum can be expressed as a linear combination of n conjugate vectors.

The Conjugate Gradients Optimisation Algorithm

Suppose that $\mathbf{H}$ is a symmetric positive-definite matrix. Define two sequences of vectors as follows:

1. $\mathbf{d}_0 = \mathbf{g}_0$ are any non-zero vectors.
2. At the jth step, set

$$\mathbf{g}_{j+1} = \mathbf{g}_j - \lambda_j \mathbf{H}\mathbf{d}_j \qquad \text{and} \qquad \mathbf{d}_{j+1} = \mathbf{g}_{j+1} + \gamma_j \mathbf{d}_j \tag{7}$$

where λ_j and γ_j are chosen to make $\mathbf{g}_{j+1}$ and $\mathbf{g}_j$ orthogonal and $\mathbf{d}_{j+1}$ and $\mathbf{d}_j$ conjugate.

We define:

$$\lambda_j = \begin{cases} 0 & \mathbf{g}_j^T H \mathbf{d}_j = 0 \\ \dfrac{\mathbf{g}_j^T \mathbf{g}_j}{\mathbf{g}_j^T H \mathbf{d}_j} & \text{otherwise} \end{cases} \tag{8}$$

$$\gamma_j = \begin{cases} 0 & \mathbf{d}_j^T \mathbf{H}\mathbf{d}_j = 0 \\ \dfrac{-\mathbf{g}_{j+1}^T \mathbf{H}\mathbf{d}_j}{\mathbf{d}_j^T \mathbf{H}\mathbf{d}_j} & \text{otherwise} \end{cases} \tag{9}$$

Newton Methods

- The basic idea behind Newton's method:

$$\nabla f(\mathbf{x}) = \nabla f(\mathbf{x}_j) + \mathbf{H}\rfloor_{\mathbf{x}_j}(\mathbf{x} - \mathbf{x}_j),$$

  which is exact if f is a quadratic form.
- If $\mathbf{x}$ is a minimum, then $\nabla f(\mathbf{x}) = 0$ and so $\mathbf{H}(\mathbf{x} - \mathbf{x}_j) = -\nabla f(\mathbf{x}_j)$ i.e. if we solve this linear equation for $\mathbf{x}$, knowing the gradient and Hessian of f at $\mathbf{x}_j$ then we can find the minimum of f. Hence

$$\mathbf{x} = \mathbf{H}^{-1}(\mathbf{H}\mathbf{x}_j - \nabla f(\mathbf{x}_j)) = \mathbf{x}_j - \mathbf{H}^{-1}\nabla f(\mathbf{x}_j)$$

  We call the vector $\mathbf{H}^{-1}\nabla f(\mathbf{x}_j)$ the Newton direction.
- If f is quadratic, then the gradient ∇f will only lie along the Newton direction if the vector joining $\mathbf{x}_j$ and the minimum of f is along an eigenvector of f. This is another explanation for the poor performance of gradient descent optimisation.

Drawbacks of Newton's Method

1. Evaluation of the Hessian can be very demanding computationally. Furthermore, if f is not a quadratic form, then the Hessian is no longer constant, and depends on the point where it is evaluated.
2. Inverting the Hessian is also computationally demanding, and can be difficult if the matrix is singular or nearly so.
3. This method is actually applying Newton's root finding method to the derivative of f: i.e. we are finding zeros of ∇f, so it is possible to find the wrong sort of critical point.
4. If the step we take is large (i.e. if $\mathbf{x}$ is a long way from $\mathbf{x}_j$), then the local quadratic approximation close to the function minimum may break down.

Quasi-Newton Methods

- An alternative is to set up the iteration

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \alpha_j \mathbf{S}_j \mathbf{g}_j. \tag{10}$$

- This is a very general algorithm; if $\mathbf{S}_j = \mathbf{H}^{-1}$ then we have Newton's method, while if $\mathbf{S}_j = \mathbf{I}$ we have steepest descent with learning rate α_j.
- Motivated by (10), we choose $\mathbf{S}_j$ to be an approximation to the inverse Hessian. It is important that $\mathbf{S}_j$ be positive definite so that for small α_j we obtain a descent method.

Scaled Conjugate Gradients

- The conjugate gradients algorithm uses line search in a very clever way to avoid having to calculate the Hessian matrix. However, the line search typically involves several additional function evaluations per step. In addition, the precision of the line search becomes another parameter which can effect the convergence of the algorithm.
- Möller's scaled conjugate gradients algorithm (Neural Networks, **6**, 525–533) provides a way of choosing 'nearly' conjugate search directions without performing a line search or calculating the Hessian.
- Practical experiments have shown that it can be an extremely efficient algorithm, outperforming both conjugate gradients and quasi-Newton algorithms when the cost of function and gradient evaluation is relatively small.

Stochastic Methods

- The methods we have discussed so far have all been concerned with finding local minima to high accuracy.
- The purpose of stochastic methods is to find global minima in high-dimensional spaces.
- The price to pay for this more ambitious goal is that the number of function evaluations is typically larger, and the level of accuracy is lower, than for the methods discussed so far.
- To find a global minimum we have to try to avoid local minima. To do this, we have to be prepared to search a large part of the space of possible solutions and we may have to move 'uphill' some of the time.
- This is in contrast to deterministic methods, where a good deal of care was taken to make sure that only downhill steps were taken.

Context and Motivation

- It is relatively easy to work with complicated functions (as the algorithms make no use of derivatives) and constraints (by restricting steps to a suitable manifold).
- Stochastic methods can also be applied to problems where the underlying space is a discrete set. In such circumstances, there may be no notion of a 'line' to search along, and derivatives may not exist.
- In neural networks, stochastic methods are particularly attractive in circumstances where accurate derivatives are hard to compute. This is the case for recurrent networks and in control problems. It is very important to use the global (i.e. full pattern set) error, and not stochastic (i.e. pattern by pattern) error.

Simulated Annealing

- At the heart of simulated annealing algorithms is an analogy with thermodynamics and the crystallization of liquids.
- At high temperatures, molecules move freely and large changes in configuration are possible. At lower temperatures, molecules move less, and there are only small changes to the configuration.
- If the liquid is cooled sufficiently slowly, the molecules can line up and form perfectly regular crystals which have the lowest energy of any possible alignment. This process is known as annealing.
- If cooling is fast (quenching), then a sub-optimal configuration may be reached.

The Metropolis Algorithm

- The simplest of the family of simulated annealing algorithms was developed by Metropolis and his team in 1953. (It was introduced as an importance sampling procedure in Monte Carlo integration.)
- Call the function f that we want to minimise, the 'energy'. Offered a change in state which results in a change of energy from E_1 to E_2, the step is taken with a probability

$$p = \exp\left[-(E_2 - E_1)/kT\right] = h(\Delta E). \qquad (11)$$

  h is known as the acceptance function.
- This formula is analogous to the Boltzmann probability distribution of the energy of a system: $\text{Prob}(E) \sim \exp(-E/kT)$, where T denotes temperature, and k is Boltzmann's constant.
- If $E_2 < E_1$, then $p > 1$ in (11), and we set $p = 1$. Thus we always take a downhill step, but take a step uphill some of the time.

Implementation

We need:

1. a method of generating random steps;
2. an annealing schedule — a method of reducing the 'temperature' T slowly.

The first of these is easy: if the underlying space is Euclidean $\mathbb{R}^D$, then random steps can be generated from a Gaussian distribution centred on the current point:

$$g(\Delta x) = (2\pi S)^{-D/2} \exp\left[-\Delta x^2/(2S)\right]. \qquad (12)$$

Combined Algorithms

- The simulated annealing algorithm can be extremely slow to converge, particularly in the later stages when close to a minimum.
- It can be useful to switch over to a local search procedure (such as quasi-Newton or conjugate gradients) when progress becomes slow (for example, due to being caught in a long narrow valley).

Constrained Optimisation

- Many practical optimisation problems place constraints on the search space. These can be of two types:
  - Equality: $g(\mathbf{x}) = 0$. These reduce the dimension of the search space.
  - Inequality: $g(\mathbf{x}) \leq 0$ or $g(\mathbf{x}) < 0$. These reduce the 'volume' of the search space, but do not affect the dimension.
- The area of search space where the constraints are satisfied is known as the feasible region. The unconstrained optimisation techniques described in this chapter cannot be applied directly to these problems.
- If the function is linear and the constraints are too, then linear programming methods can be used. If the function is quadratic and the constraints are linear, then quadratic programming methods can be used.

Summary

- Understand the two-stage approach to multivariate optimisation.
- Understand the properties and drawbacks of gradient descent.
- Understand the place conjugate directions in second order optimisation methods.
- Know of conjugate gradients, quasi-Newton and scaled conjugate gradient algorithms.
- Describe the strengths and weaknesses of stochastic methods.

Scipy optimization package:
`https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html`