Optimisation[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.

The need for optimisation

Machine learning often requires fitting a model to data. Often this means finding the parameters $\boldsymbol{\theta}$ of the model that 'best' fit the data.

Regression

For example, for regression based on training data $(\mathbf{x}^n, y^n)$ we might have a model $y(x|\boldsymbol{\theta})$ and wish to set $\boldsymbol{\theta}$ by minimising

$$E(\theta) = \sum_n \left(y^n - y(\mathbf{x}^n|\theta)\right)^2$$

Complexity

In all but very simple cases, it is extremely difficult to find an algorithm that will guarantee to find the optimal $\boldsymbol{\theta}$.

A simple case: Linear regression

For example for a linear predictor

$$y(\mathbf{x}|\theta) \equiv \mathbf{x}^{\mathsf{T}}\boldsymbol{\theta}$$

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

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

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

$$\underbrace{\sum_n y^n x_i^n}_{b_i} = \sum_j \underbrace{\sum_n x_i^n x_j^n}_{X_{ij}} \theta_j$$

Hence, in matrix form, this is

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

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

Quadratic functions

A class of simple functions to optimise is, for symmetric $\mathbf{A}$:

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

This has a unique minimum if and only if $\mathbf{A}$ is positive definite. In this case, at the optimimum

$$\nabla f = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \rightarrow \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

For $\mathbf{x} + \boldsymbol{\delta}$, the new value of the function is

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \underbrace{\boldsymbol{\delta}^\mathsf{T}\nabla f}_{=0} + \frac{1}{2}\underbrace{\boldsymbol{\delta}^\mathsf{T}\mathbf{A}\boldsymbol{\delta}}_{\geq 0}$$

Hence $\mathbf{A}^{-1}\mathbf{b}$ is a minimum.

Gradient Descent and First Order Methods

Gradient Descent

We wish to find $\mathbf{x}$ that minimises $f(\mathbf{x})$. For general f there is no closed-form solution to this problem and we typically resort to iterative methods.
For $\mathbf{x}_{k+1} \approx \mathbf{x}_k$,

$$f(\mathbf{x}_{k+1}) \approx f(\mathbf{x}_k) + (\mathbf{x}_{k+1} - \mathbf{x}_k)^\mathsf{T} \nabla f(\mathbf{x}_k)$$

setting

$$\mathbf{x}_{k+1} - \mathbf{x}_k = -\epsilon \nabla f(\mathbf{x}_k)$$

gives

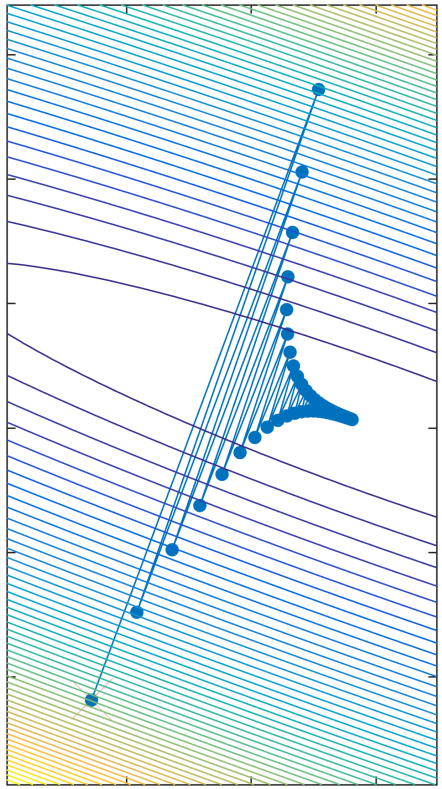
$$f(\mathbf{x}_{k+1}) \approx f(\mathbf{x}_k) - \epsilon \left|\nabla f(\mathbf{x}_k)\right|^2$$

Hence, for a small ϵ, the algorithm

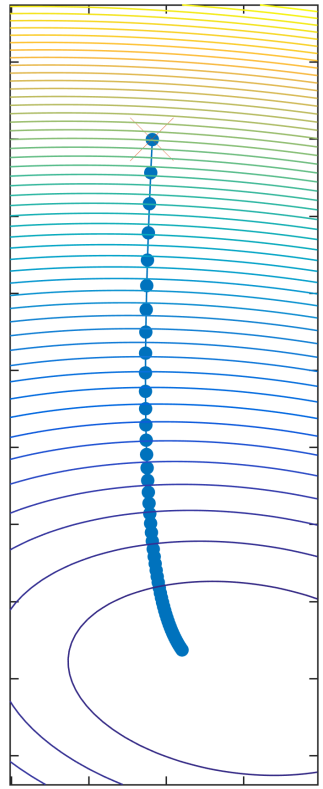
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon \nabla f(\mathbf{x}_k)$$

decreases f. We iterate until convergence.

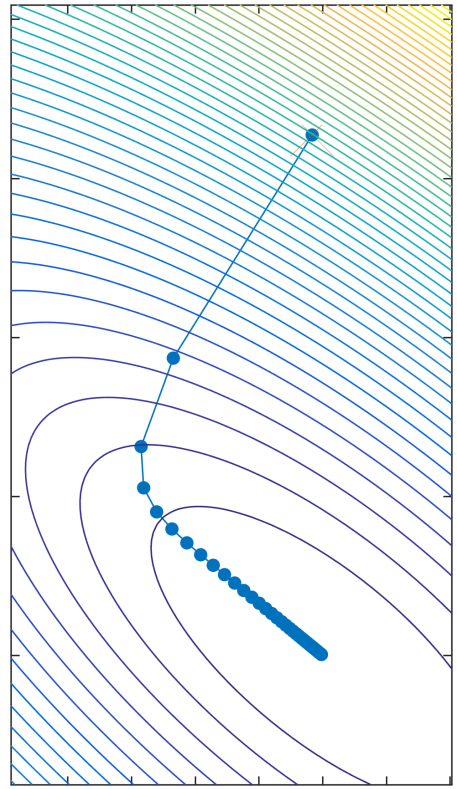
Gradient Descent



(a) Learning rate too large



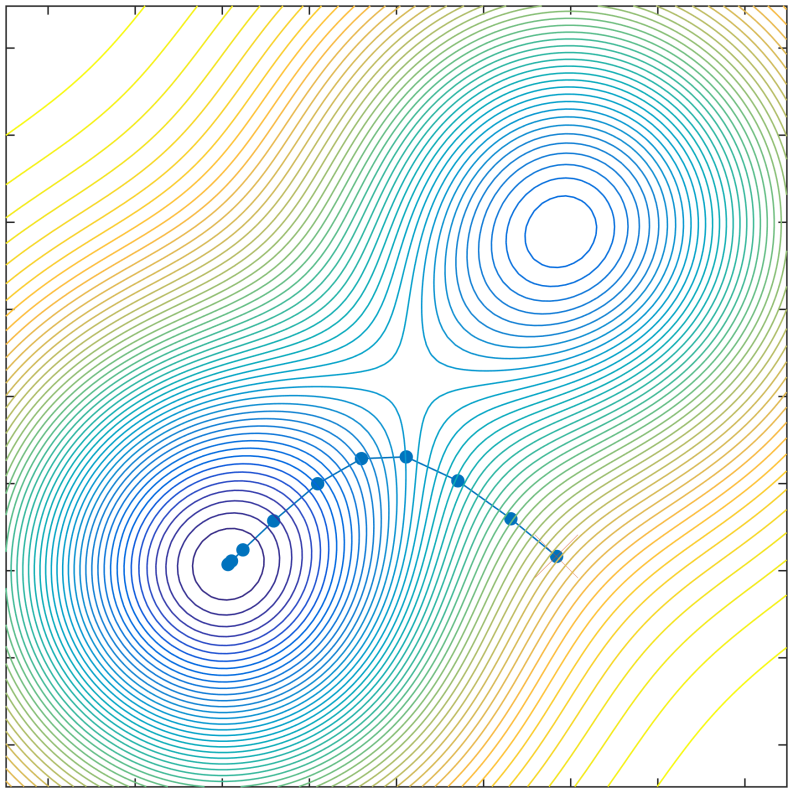
(b) Learning rate too small



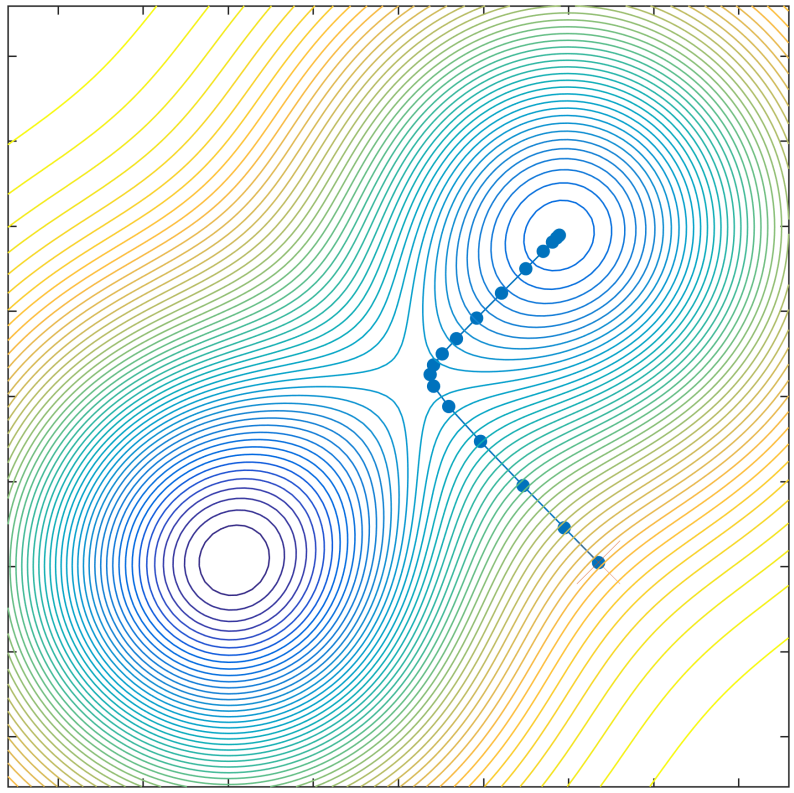
(c) Learning rate OK

Note the difference between converging to the minimal function value, and the optimum parameters; we might have almost converged to the minimal value but still be a long way from the optimum parameters. Common to consider adapting the learning rate. This is usually determined by experimenting with different values or learning schedules.

Gradient Descent



(d) Found global minimum



(e) Found local minimum

Depending on the initial point, even for the same function and learning rate, we can converge to different solutions.

Generalised gradient update

- Consider an update of the form using a 'curvature matrix' $\mathbf{C}$,

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \epsilon \mathbf{C}^{-1}\mathbf{g}$$

where $\mathbf{C}$ is symmetric and positive definite, $\epsilon > 0$.

$$f(\mathbf{x}_{t+1}) \approx f(\mathbf{x}_t) - \epsilon \mathbf{g}^{\mathsf{T}}\mathbf{C}^{-1}\mathbf{g} + \frac{\epsilon^2}{2}\mathbf{g}^{\mathsf{T}}\mathbf{C}^{-1}\mathbf{H}\mathbf{C}^{-1}\mathbf{g}$$

- The first term $-\epsilon \mathbf{g}^{\mathsf{T}}\mathbf{C}^{-1}\mathbf{g}$ reduces the function value (the inverse of a positive definite matrix is positive definite). Provided ϵ is small, the second term will be small and not affect the function reduction.
- Up to second order, we would get a reduction in the function value provided

$$\epsilon < \frac{2\mathbf{g}^{\mathsf{T}}\mathbf{C}^{-1}\mathbf{g}}{\mathbf{g}^{\mathsf{T}}\mathbf{C}^{-1}\mathbf{H}\mathbf{C}^{-1}\mathbf{g}}$$

- Will discuss later good ways to find suitable 'curvature' matrices.

Convergence Rate for Convex Functions

Let's assume:

Convexity f is convex and finite over the path $\mathbf{x}_1, \ldots, \mathbf{x}_T$ that the gradient descent algorithm will take.

Finite Solution The optimum $\mathbf{x}^*$ exists and is finite.

∇f Lipschitz continuous The gradient is Lipschitz with constant L. We further assume f is twice differentiable. This means

$$\mathbf{H}(\mathbf{x}) \preceq L\mathbf{I}$$

so that all the eigenvalues of the Hessian are less than or equal to L.

Learning Rate At each step, the learning rate $\epsilon \leq 1/L$.

Convergence:

$$f(\mathbf{x}_T) - f(\mathbf{x}^*) \leq \frac{1}{2\epsilon T} \left(\mathbf{x}_1 - \mathbf{x}^*\right)^2$$

This means that the error goes down as $O\left(1/T\right)$ where T is the number of iterations in the gradient descent algorithm.

Proof (1)

Note that I'm using t here instead of k for the iteration index.

- $\mathbf{x}_{t+1} = \mathbf{x}_t - \epsilon \nabla f(\mathbf{x}_t)$
- Taylor's theorem (with residual term) says

$$f(\mathbf{y}) = f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^\mathsf{T} \nabla f(\mathbf{x}) + \frac{1}{2}(\mathbf{y} - \mathbf{x})^\mathsf{T} \mathbf{H}(\mathbf{z})(\mathbf{y} - \mathbf{x})$$

for some $\mathbf{z}$ between $\mathbf{x}$ and $\mathbf{y}$. Since $\mathbf{H} \preceq L\mathbf{I}$ then (writing $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$)

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \epsilon \mathbf{g}^2(\mathbf{x}_t) + \frac{L\epsilon^2}{2}\mathbf{g}^2(\mathbf{x}_t) = f(\mathbf{x}_t) - \epsilon\left(1 - \frac{L\epsilon}{2}\right)\mathbf{g}^2(\mathbf{x}_t)$$

If $\epsilon \leq 2/L$, then $f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t)$. Optimising over ϵ the biggest decrease is obtained using a learning rate $\epsilon = 1/L$. We assume throughout that $\epsilon \leq 1/L$, for which

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \frac{\epsilon}{2}\mathbf{g}^2(\mathbf{x}_t)$$

- Hence, provided ϵ is small enough, we will decrease the error with each update.

Proof (2)

- Since f is convex,

$$f(\mathbf{y}) \geq f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^{\mathsf{T}} \nabla f(\mathbf{x})$$

  Which means

$$f(\mathbf{x}_t) \leq f(\mathbf{x}^*) + (\mathbf{x}_t - \mathbf{x}^*)^{\mathsf{T}} \mathbf{g}(\mathbf{x}_t)$$

- Hence

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \frac{\epsilon}{2}\mathbf{g}^2(\mathbf{x}_t) \leq f(\mathbf{x}^*) + (\mathbf{x}_t - \mathbf{x}^*)^{\mathsf{T}} \mathbf{g}(\mathbf{x}_t) - \frac{\epsilon}{2}\mathbf{g}^2(\mathbf{x}_t)$$

- Write $\mathbf{d} = \mathbf{x}_t - \mathbf{x}^*$. Then

$$\mathbf{d}^{\mathsf{T}}\mathbf{g} - \frac{\epsilon}{2}\mathbf{g}^2 = -\frac{1}{2\epsilon}\left[(\epsilon\mathbf{g} - \mathbf{d})^2 - \mathbf{d}^2\right]$$

- Hence, using $\mathbf{d} - \epsilon\mathbf{g} = \mathbf{x}_t - \epsilon\mathbf{g} - \mathbf{x}^* = \mathbf{x}_{t+1} - \mathbf{x}^*$

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}^*) + \frac{1}{2\epsilon}\left[(\mathbf{x}_t - \mathbf{x}^*)^2 - (\mathbf{x}_{t+1} - \mathbf{x}^*)^2\right]$$

Proof (3)

- Rearranging the above,

$$f(\mathbf{x}_{t+1}) - f(\mathbf{x}^*) \leq \frac{1}{2\epsilon}\left[(\mathbf{x}_t - \mathbf{x}^*)^2 - (\mathbf{x}_{t+1} - \mathbf{x}^*)^2\right]$$
$$f(\mathbf{x}_{t+2}) - f(\mathbf{x}^*) \leq \frac{1}{2\epsilon}\left[(\mathbf{x}_{t+1} - \mathbf{x}^*)^2 - (\mathbf{x}_{t+2} - \mathbf{x}^*)^2\right]$$

- Hence the sum 'telescopes' with terms cancelling:

$$\sum_{t=1}^{T}(f(\mathbf{x}_{t+1}) - f(\mathbf{x}^*)) \leq \frac{1}{2\epsilon}\left[(\mathbf{x}_1 - \mathbf{x}^*)^2 - (\mathbf{x}_{T+1} - \mathbf{x}^*)^2\right]$$
$$\leq \frac{1}{2\epsilon}(\mathbf{x}_1 - \mathbf{x}^*)^2$$

Since $f(\mathbf{x}_T) \leq f(\mathbf{x}_t)$,

$$\sum_{t=1}^{T}(f(\mathbf{x}_T) - f(\mathbf{x}^*)) \leq \sum_{t=1}^{T}(f(\mathbf{x}_t) - f(\mathbf{x}^*))$$
$$T\,(f(\mathbf{x}_T) - f(\mathbf{x}^*)) \leq \frac{1}{2\epsilon}(\mathbf{x}_1 - \mathbf{x}^*)^2 \qquad \blacksquare$$

Momentum

One simple idea to limit the zig-zag behaviour is to make an update in the average direction of the previous updates.

Moving average

Consider a set of numbers $x_1, \ldots, x_t$. Then the average a_t is given by

$$a_t = \frac{1}{t}\sum_{\tau=1}^{t} x_\tau = \frac{1}{t}\left(x_t + (t-1)a_{t-1}\right) = \epsilon_t x_t + \mu_t a_{t-1}$$

for suitably chosen ϵ_t and $0 \leq \mu_t \leq 1$. If μ_t is small then the more recent x contribute more strongly to the moving average.

Momentum

Idea is to use a form of moving average to the updates:

$$\tilde{\mathbf{g}}_{k+1} = \mu_k \tilde{\mathbf{g}}_k - \epsilon \mathbf{g}_k(\mathbf{x}_k)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \tilde{\mathbf{g}}_{k+1}$$

Hence, instead of using the update $-\epsilon \mathbf{g}_k(\mathbf{x}_k)$, we use the moving average of the update $\tilde{\mathbf{g}}_{k+1}$ to form the new update.

Momentum

- Momentum can increase the speed of convergence since, for smooth objectives, as we get close to the minimum the gradient decreases and standard gradient descent starts to slow down.
- If the learning rate is too large, standard gradient descent may oscillate, but momentum may reduce oscillations by going in the average direction.
- However, the momentum parameter μ may need to be reduced with the iteration count to ensure convergence.
- Particularly useful when the gradient is noisy. By averaging over previous gradients, the noise 'averages' out and the moving average direction can be much less noisy.
- Momentum is also useful to avoid saddles (a point where the gradient is zero, but the objective function is not a minimum, such as the function x^3 at the origin) since typically the momentum will carry you over the saddle.

Nesterov's Accelerated Gradient

Nesterov

- This looks similar to momentum but has a slightly different update

$$\tilde{\mathbf{g}}_{k+1} = \mu_k \tilde{\mathbf{g}}_k - \epsilon \mathbf{g}(\mathbf{x}_k + \mu_k \tilde{\mathbf{g}}_k)$$

  That is, we use the gradient of the point we will move to, rather than the current point.

- Need to choose a schedule for μ_k. Nesterov suggests

$$\mu_k = 1 - 3/(k+5)$$

- For convex functions NAG has rate of convergence to the optimum

$$f(\mathbf{x}_k) - f^* \leq \frac{c}{k^2}$$

  for some constant c, compared to $1/k$ convergence for gradient descent.

'Understanding' Nesterov's Accelerated Gradient

Let's imagine that we've arrived at our current parameter $\mathbf{x}_k$ based on an update

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{v}_{k-1} \tag{1}$$

where $\mathbf{v}_{k-1}$ is the update. We now (retrospectively) ask: 'What would have been a better update than the one we actually made?'. Well, using the update we arrived at a function value

$$f(\mathbf{x}_k) = f(\mathbf{x}_{k-1} + \mathbf{v}_{k-1})$$

We could have got to a better value by changing $\mathbf{v}_{k-1}$ to

$$\mathbf{v}_k = \mathbf{v}_{k-1} - \epsilon \frac{\partial}{\partial \mathbf{v}} f(\mathbf{x}_k) = \mathbf{v}_{k-1} - \epsilon \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_{k-1} + \mathbf{v}_{k-1})$$

Hence, we define

$$\mathbf{v}_k = \mathbf{v}_{k-1} - \epsilon \mathbf{g}(\mathbf{x}_{k-1} + \mathbf{v}_{k-1})$$

and make the update (note the difference with (1))

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{v}_k$$

'Understanding' Nesterov's Accelerated Gradient

- The basic update

$$\mathbf{v}_k = \mathbf{v}_{k-1} - \epsilon \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_{k-1} + \mathbf{v}_{k-1})$$

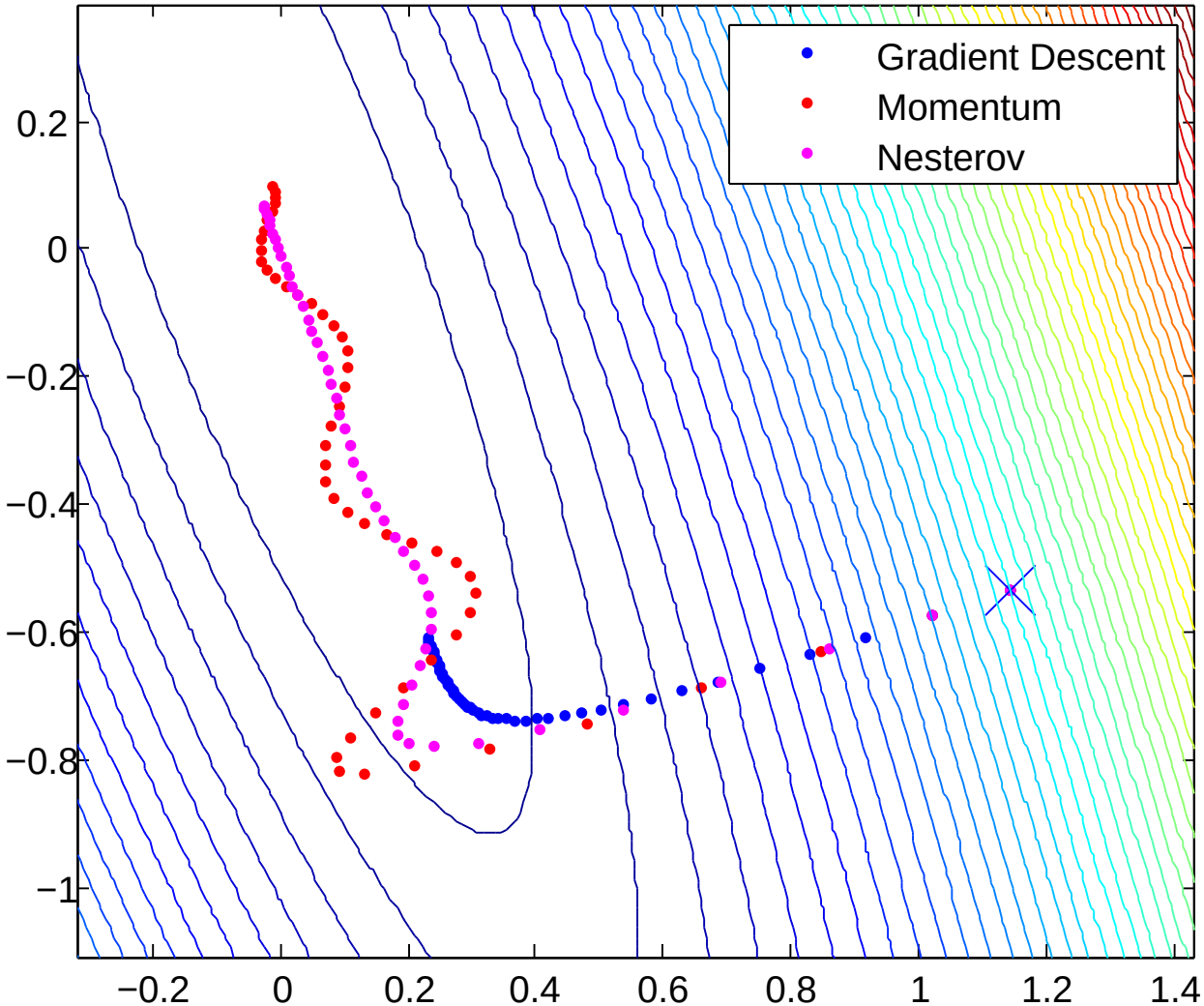
  essentially is a form of momentum that pushes $\mathbf{x}$ along the direction it was previously going.
- Even when we reach a point where the gradient is zero, then $\mathbf{v}_k$ will be the same as $\mathbf{v}_{k-1}$.
- Without modification, this vanilla algorithm will diverge.
- We therefore introduce a term to dampen oscillations and ensure convergence. One can show that this gives the standard Nesterov update:

$$\mathbf{v}_k = \mu_{k-1}\mathbf{v}_{k-1} - \epsilon \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_{k-1} + \mu_{k-1}\mathbf{v}_{k-1})$$

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{v}_k$$

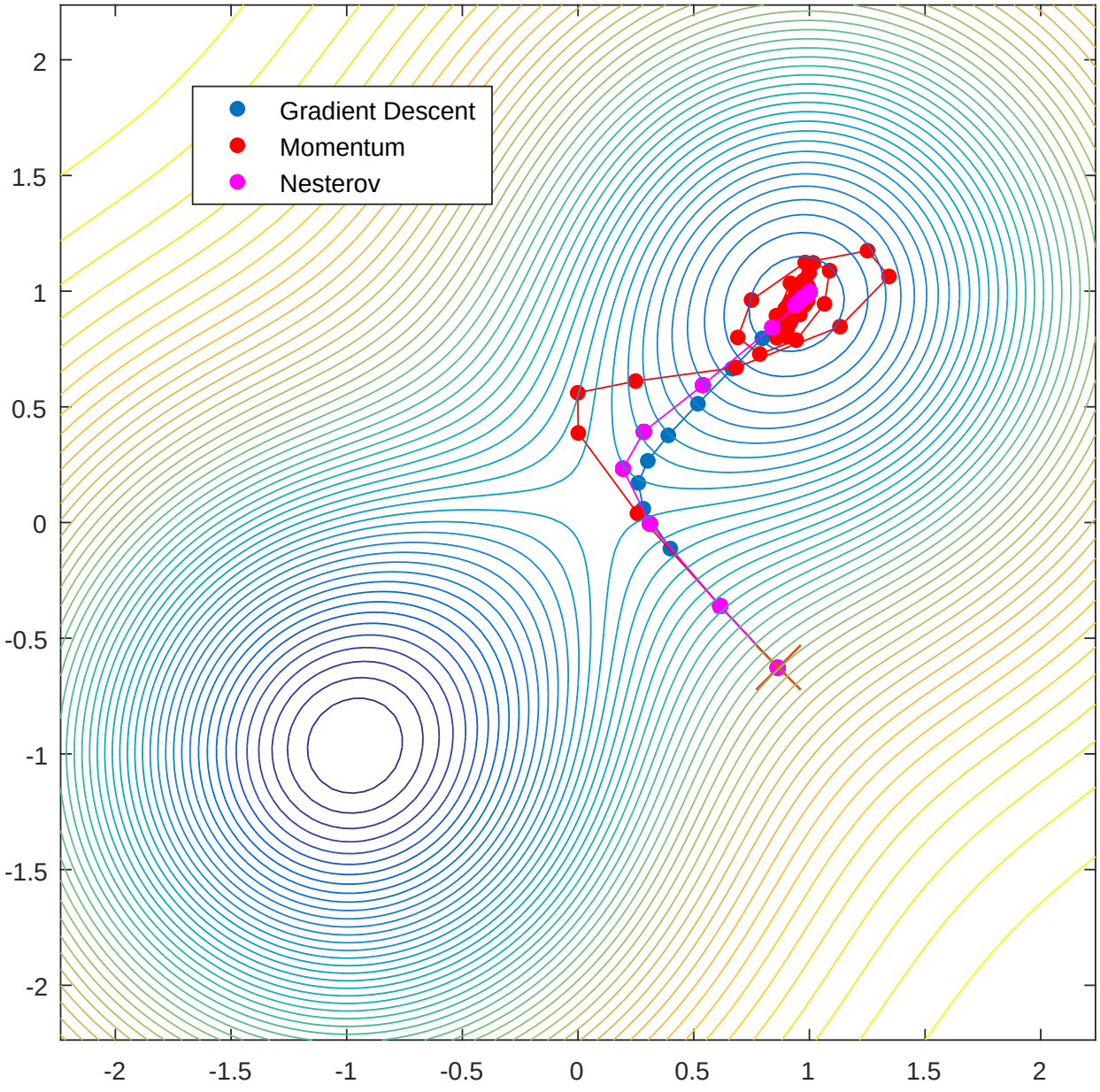
  Since $\mu < 1$, $\mathbf{v}$ will quickly converge to zero around the minimum.
- One can also view this as a 'trust' region factor (see later) that says that we only trust the Taylor expansion provided that the update is not too large.

Demo



$\epsilon = 0.1$. $\mu_t = 1 - 3/(t+5)$ for both momentum and Nesterov. All trajectories start at the same point and have 50 updates. Nesterov oscillates much less than momentum. See `demoGradDescent.m`

Demo



$\epsilon = 0.1$. $\mu_t = 1 - 3/(t+5)$ for both momentum and Nesterov.

Comments on gradient descent

Good

This is a very simple algorithm that is easy to implement.

Bad

Only improves the solution at each stage by a small amount. If ϵ is not small enough, the function may not decrease in value. In practice one needs to find a suitably small ϵ to guarantee convergence.

Ugly

The method is coordinate system dependent. Let $\mathbf{x} = \mathbf{M}\mathbf{y}$ and define

$$\hat{f}(\mathbf{y}) = f(\mathbf{x})$$

We now perform gradient descent on $\hat{f}(\mathbf{y})$:

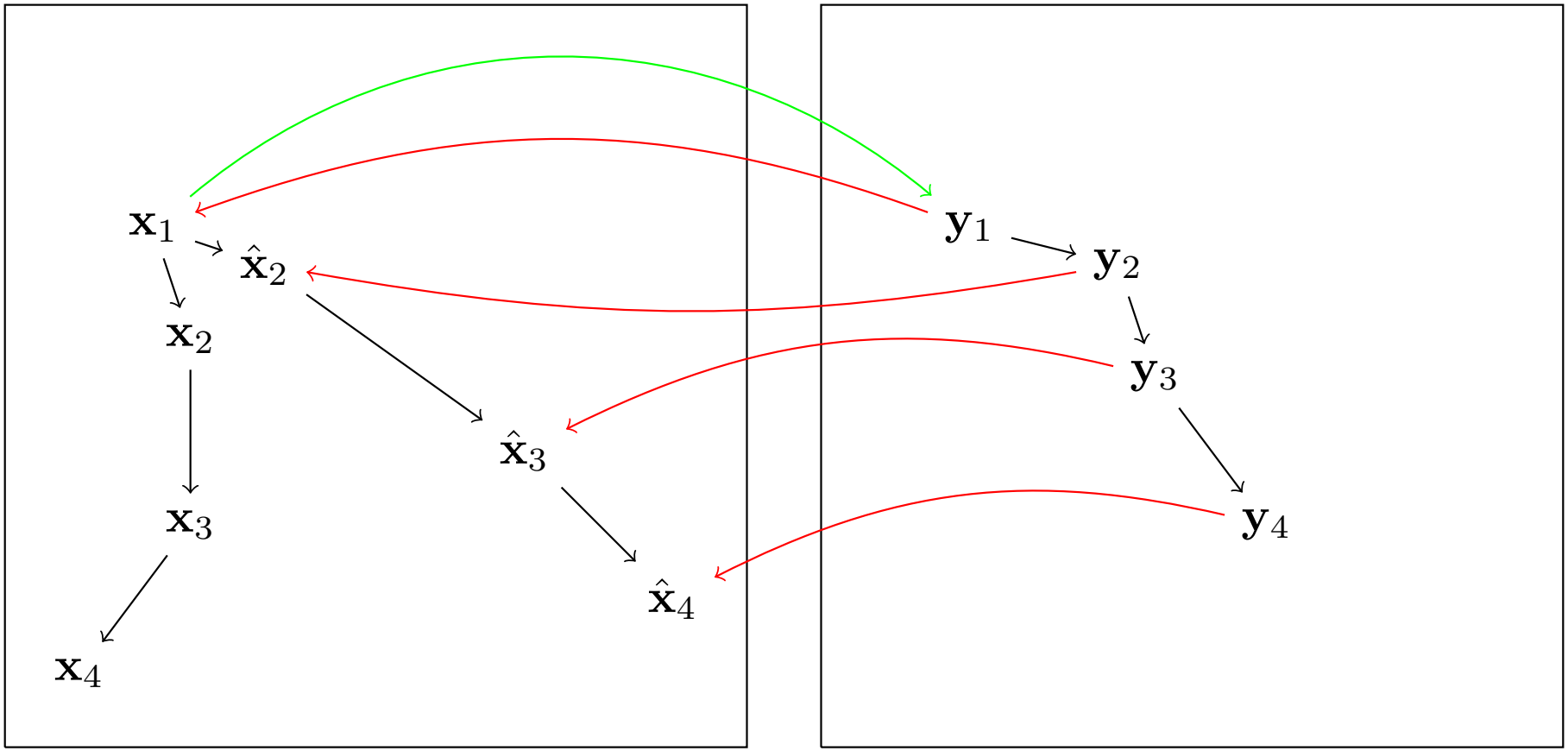
$$[\mathbf{y}_{k+1}]_i = [\mathbf{y}_k]_i - \epsilon \sum_j \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial y_i} \rightarrow \mathbf{y}_{k+1} = \mathbf{y}_k - \epsilon \mathbf{M}^\mathsf{T} \nabla f(\mathbf{x}_k)$$

Hence

$$\mathbf{M}\mathbf{y}_{k+1} = \mathbf{M}\mathbf{y}_k - \epsilon \mathbf{M}\mathbf{M}^\mathsf{T} \nabla f(\mathbf{x}_k) \rightarrow \mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon \mathbf{M}\mathbf{M}^\mathsf{T} \nabla f(\mathbf{x}_k)$$

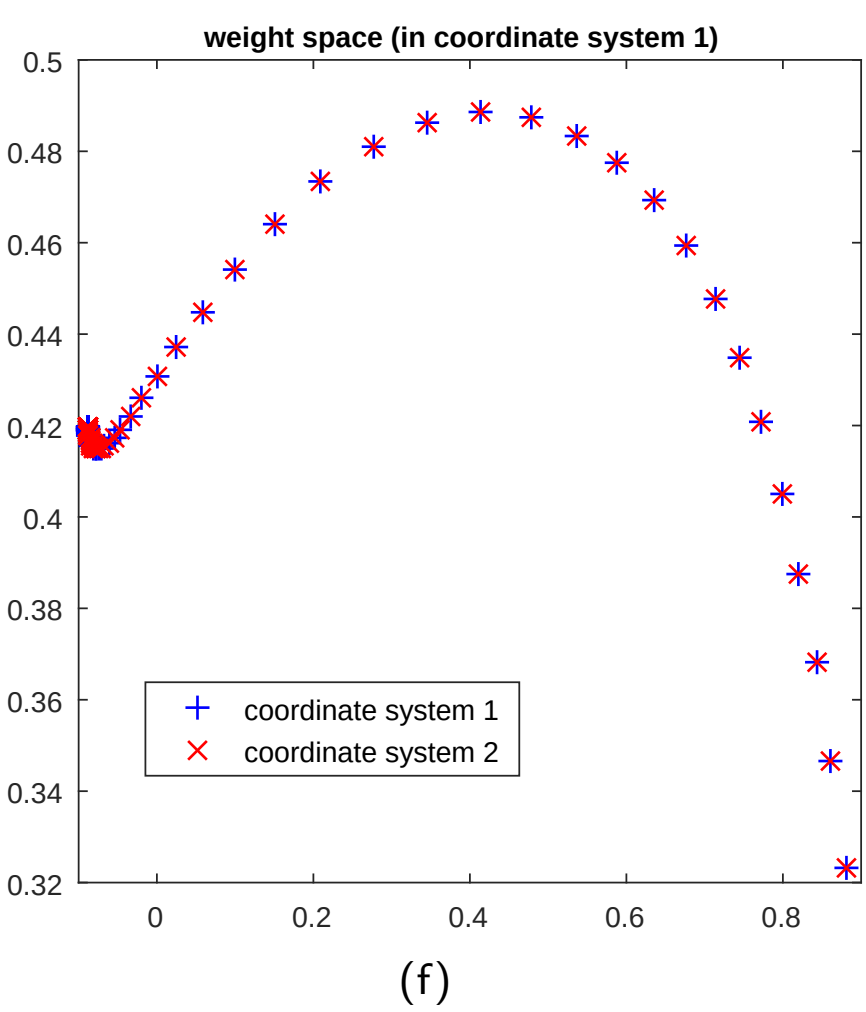
The algorithm is coordinate system dependent (except for orthogonal transformations).

Coordinate System Dependence

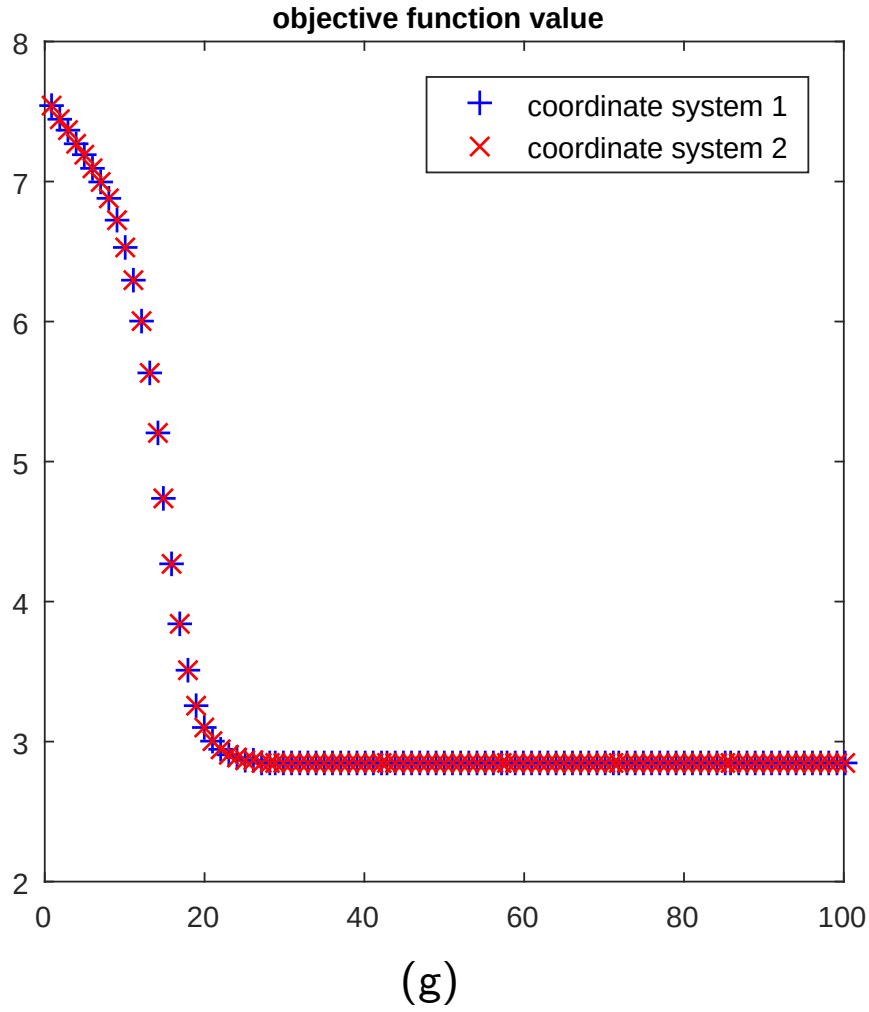


The $\mathbf{x}_k$ points are the standard gradient descent vectors in the x-coordinate system. We then map the initial point $\mathbf{x}_1$ to the corresponding point $\mathbf{y}_1 = \mathbf{M}^{-1}\mathbf{x}_1$ and begin gradient descent in the $\mathbf{y}$ space. We can then map each point $\mathbf{y}_k$ back to the corresponding point in the x-space using $\hat{\mathbf{x}}_k = \mathbf{M}\mathbf{y}_k$. In general (unless $\mathbf{M}$ is orthogonal) the $\mathbf{x}$ and $\hat{\mathbf{x}}$ trajectories are different. See `demoGradDescentCoordTransform.m`.

Demo



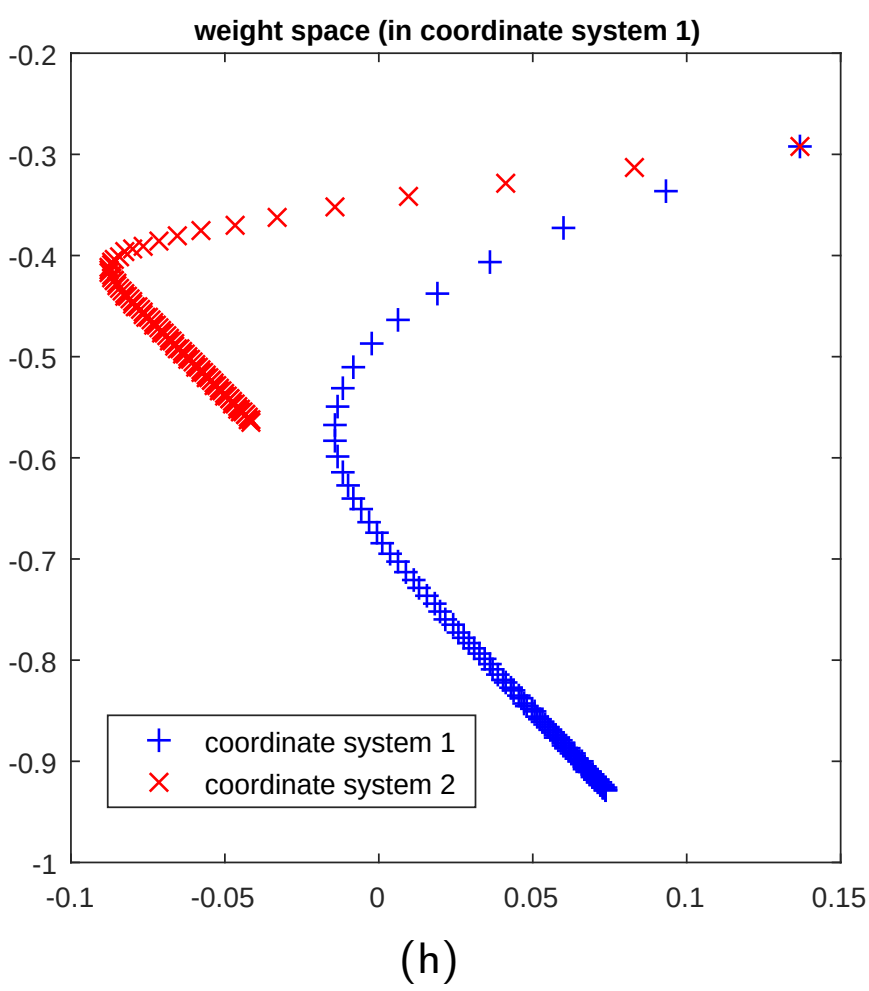
(f)



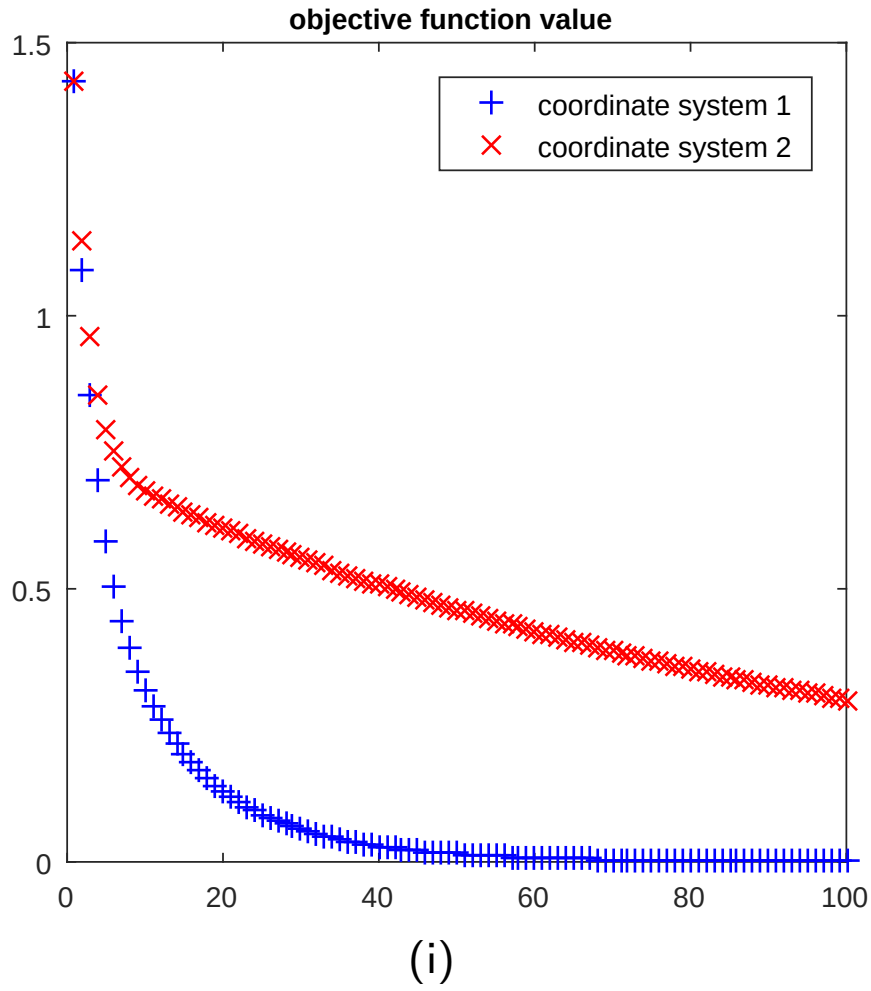
(g)

If we use an orthogonal M the gradient descent is invariant to the coordinate transformation.

Demo



(h)



(i)

If we use an non-orthogonal M the gradient descent depends on the coordinate transformation.

Line Search and Conjugate Gradients

Line Search

One way to potentially improve on gradient descent is choose a particular direction $\mathbf{p}_k$ and search along there. We then find the minimum of the one dimensional problem

$$F(\lambda) = f(\mathbf{x}_k + \lambda \mathbf{p}_k)$$

Finding the optimal λ^* can be achieved using a standard one-dimensional optimisation method. For example if we identify a bracket such that $f(a) > f(b) < f(c)$ and then fit a polynomial to estimate a new x with $f(x) < f(b)$, this identifies a new bracket. One then repeats until convergence. Once found we set

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda^* \mathbf{p}_k$$

and then choose another search direction $\mathbf{p}_{k+1}$ and iterate.

Good

By reducing the problem to a sequence of one dimensional optimisation problems, at each stage the potential change in f is greater than would be typically achievable by gradient descent.

Search directions

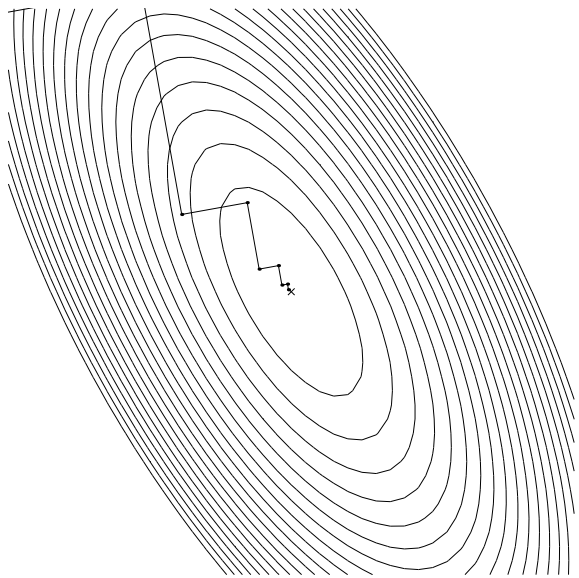
How do we choose a good search direction?

Choosing the search directions

It would seem reasonable to choose a search direction that points 'maximally downhill' from the current point $\mathbf{x}_k$. That is, to set

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

However, at least for quadratic functions, this is not optimal and leads to potentially zig-zag behaviour:



This zig-zag behaviour can occur for non isotropic surfaces.

Philosophy

Other ways to avoid problems of basic gradient descent:

Use Quadratic functions to gain insight

Much of the theory of optimisation is based on the following: Even though the problem is not quadratic, let's pretend the problem is quadratic and see what we would do in that case. This will inspire the solution for the general case.

Quadratic functions

In the quadratic function case, we get zig-zag behaviour if $\mathbf{A}$ is not diagonal. If we could find a coordinate transform $\mathbf{x} = \mathbf{P}\hat{\mathbf{x}}$

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

with

$$\mathbf{P}^{\mathsf{T}}\mathbf{A}\mathbf{P}$$

being diagonal, then we can perform line-search independently along each axis of $\hat{\mathbf{x}}$ and find the solution efficiently. This is achieved by the conjugate gradient algorithm.

Conjugate vectors

The vectors $\mathbf{p}_i$, $i = 1, \ldots, n$ are conjugate for the $n \times n$ matrix $\mathbf{A}$ if

$$\mathbf{p}_i^\mathsf{T} \mathbf{A} \mathbf{p}_j = \delta_{ij} \mathbf{p}_i^\mathsf{T} \mathbf{A} \mathbf{p}_i, \qquad i, j = 1, \ldots, n$$

Finding the optimum of a quadratic

Searching along these conjugate directions effectively 'diagonalises' the problem. If we write $\mathbf{x} = \sum_i \alpha_i \mathbf{p}_i$, then

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\mathsf{T} \mathbf{A} \mathbf{x} - \mathbf{b}^\mathsf{T} \mathbf{x} = \sum_{i=1}^{n} \left(\frac{1}{2} \alpha_i^2 \mathbf{p}_i^\mathsf{T} \mathbf{A} \mathbf{p}_i - \alpha_i \mathbf{b}^\mathsf{T} \mathbf{p}_i \right)$$

which gives a set of independent optimisation problems, one for each α_i. Hence, if we find the optimum along n conjugate directions, we must have found the optimum $\mathbf{x}$.

Conjugate Gradient

- The idea of conjugate gradients is to perform the independent optimisations along conjugate directions, but do these optimisations sequentially, building up the set of conjugate vectors as we do so.
- To keep the argument simple, set $\mathbf{x}_1 = \mathbf{0}$ and in general

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

  At each iteration we seek the optimum α_k.
- Using this, we then build up the solution

$$\mathbf{x}_{k+1} = \sum_{i=1}^{k} \alpha_i \mathbf{p}_i$$

  so that after n iterations, we will have found the overall optimum.
- It's enough at each iteration to search along a direction $\mathbf{p}_k$ that is conjugate to all previous search directions. Writing $\mathbf{a} \perp \mathbf{b}$ if $\mathbf{a}$ is conjugate to $\mathbf{b}$. For example for $n = 4$, choose $\mathbf{p}_1$. Then find $\mathbf{p}_2 \perp \mathbf{p}_1$. Then find $\mathbf{p}_3$ such that $\mathbf{p}_3 \perp \mathbf{p}_2, \mathbf{p}_3 \perp \mathbf{p}_1$. Then find $\mathbf{p}_4$ such that $\mathbf{p}_4 \perp \mathbf{p}_3, \mathbf{p}_4 \perp \mathbf{p}_2, \mathbf{p}_4 \perp \mathbf{p}_1$. We now have a full conjugate set.

Conjugate Gradients

The question now is how to find the conjugate directions. As we saw, we only require that $\mathbf{p}_k$ is conjugate to $\mathbf{p}_1, \ldots, \mathbf{p}_{k-1}$. Defining $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$, and

$$\beta_k = \mathbf{g}_{k+1}^\mathsf{T} \mathbf{g}_{k+1} / (\mathbf{g}_k^\mathsf{T} \mathbf{g}_k)$$

one can show that for our quadratic function,

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k$$

is conjugate to all previous $\mathbf{p}_i$, $i = 1, \ldots, k$.
A more common alternative is the Polak-Ribière formula.

$$\beta_k = \frac{\mathbf{g}_{k+1}^\mathsf{T} (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^\mathsf{T} \mathbf{g}_k}$$

For quadratic functions, with $\dim \mathbf{x} = n$, conjugate gradients is guaranteed to find the optimum in n iterations, each iteration taking $O\left(n^2\right)$ operations. In a more general non-quadratic problem, no such guarantee exists.

Conjugate Gradients

This gives rise to the conjugate gradients for minimising a function $f(\mathbf{x})$

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

Higher Order Methods

Newton's method

Consider a function $f(\mathbf{x})$ that we wish to find the minimum of. A Taylor expansion up to second order gives

$$f(\mathbf{x}+\boldsymbol{\Delta}) = f(\mathbf{x}) + \boldsymbol{\Delta}^{\mathsf{T}}\nabla f + \frac{1}{2}\boldsymbol{\Delta}^{\mathsf{T}}\mathbf{H}_f\boldsymbol{\Delta} + O(|\boldsymbol{\Delta}|^3)$$

The matrix $\mathbf{H}_f$ is the Hessian.
Differentiating the right hand side with respect to $\boldsymbol{\Delta}$ (or, equivalently, completing the square), we find that the right hand side has its lowest value when

$$\nabla f = -\mathbf{H}_f\boldsymbol{\Delta} \Rightarrow \boldsymbol{\Delta} = -\mathbf{H}_f^{-1}\nabla f$$

Hence, an optimisation routine to minimise f is given by the Newton update

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon\mathbf{H}_f^{-1}\nabla f$$

With this update, the change in the function is

$$\boldsymbol{\Delta}^{\mathsf{T}}\nabla f + \frac{1}{2}\boldsymbol{\Delta}^{\mathsf{T}}\mathbf{H}_f\boldsymbol{\Delta} = \epsilon\left(-1+\frac{\epsilon}{2}\right)\nabla f^{\mathsf{T}}\mathbf{H}_f^{-1}\nabla$$

For quadratic functions, Newton's method converges in one step (for $\epsilon = 1$). More generally one uses a value $\epsilon < 1$ to avoid overshooting effects.

Comments on Newton's method

Good

A benefit of the Newton method over gradient descent is that the decrease in the objective function is invariant under a linear change of co-ordinates, $\mathbf{x} = \mathbf{M}\mathbf{y}$. Defining $\hat{f}(\mathbf{y}) \equiv f(\mathbf{x})$, the change in $\mathbf{y}$ under a Newton update is

$$-\mathbf{H}_{\hat{f}}^{-1}\nabla_{\mathbf{y}}\hat{f}$$

where $\nabla_{\mathbf{y}}\hat{f} = \mathbf{M}^{\mathsf{T}}\nabla_{\mathbf{x}}f$, $\mathbf{H}_{\hat{f}} = \mathbf{M}^{\mathsf{T}}\mathbf{H}_f\mathbf{M}$. In terms of the $\mathbf{x}$ coordinate system the change is

$$\mathbf{M}\Delta\mathbf{y} = -\mathbf{M}\mathbf{H}_{\hat{f}}^{-1}\nabla_{\mathbf{y}}\hat{f} = -\mathbf{M}\left(\mathbf{M}^{\mathsf{T}}\mathbf{H}_f\mathbf{M}\right)^{-1}\mathbf{M}^{\mathsf{T}}\nabla_{\mathbf{x}}f = -\mathbf{H}_f^{-1}\nabla_{\mathbf{x}}f = \Delta\mathbf{x}$$

so that the change is independent of the coordinate system (up to linear transformations of the coordinates).

Bad

Storing the Hessian and solving the linear system $\mathbf{H}_f^{-1}\nabla f$ is very expensive.

More comments on Newton's method

- Newton's method is not guaranteed to produce a downhill step!
- This only happens (for sure) if ϵ is small and $\mathbf{H}$ is positive definite.
- If $\mathbf{H}$ is positive definite, one can use a line search in the direction $\mathbf{H}^{-1}\nabla f$ to ensure we go downhill and make a non-trivial jump.

Using Conjugate gradient

When we solve the linear system $\mathbf{H}\boldsymbol{\Delta} = \nabla f$, a practical approach is to find $\boldsymbol{\Delta}$ by minimising

$$\Psi(\boldsymbol{\Delta}) \equiv \frac{1}{2}\boldsymbol{\Delta}^{\mathsf{T}}\mathbf{H}\boldsymbol{\Delta} - \boldsymbol{\Delta}^{\mathsf{T}}\nabla f$$

This is typically much faster than calling standard linear solvers (such as Gaussian elimination).

Quasi-Newton

- In Quasi-Newton methods such as Broyden-Fletcher-Goldfarb-Shannon, an approximate inverse Hessian is formed iteratively.
- LBFGS is a popular practical method that limits the memory requirement of BFGS.

Gauss Newton

Consider objectives of the form

$$E(\mathbf{w}) = \sum_{n=1}^{N} \left[y^n - f(\mathbf{x}^n, \mathbf{w})\right]^2$$

where n is for example a data index in a set of N datapoints. This is typical of square loss functions in regression for predictor function f.

Hessian

$$\frac{\partial^2 E}{\partial w_i \partial w_j} = 2\sum_n \left(-\left[y^n - f(\mathbf{x}^n, \mathbf{w})\right] \frac{\partial^2 f^n}{\partial w_i \partial w_j} + \frac{\partial f^n}{\partial w_i}\frac{\partial f^n}{\partial w_j} \right), \quad f^n \equiv f(\mathbf{x}^n, \mathbf{w})$$

- Close to the minimum of E, provided the function f is fitting the data well, then the residuals $y^n - f(\mathbf{x}^n, \mathbf{w})$ will be small and we can ignore the first term.
- The second term is positive definite. We therefore define the matrix $\mathbf{C} = 2\sum_n \nabla f^n \left(\nabla f^n\right)^\mathsf{T}$ and use this in place of $\mathbf{H}$ in the Newton update.

Generalised Gauss Newton

Consider an objective function

$$f(\mathbf{x}) = L(\mathbf{h}(\mathbf{x}))$$

where $\mathbf{h}(\mathbf{x})$ is a vector-valued function and L is a convex loss function. Then

$$\frac{\partial^2 f}{\partial x_i x_j} = \sum_k \frac{\partial L}{\partial h_k} \frac{\partial^2 h_k}{\partial x_i \partial x_j} + \sum_{k,l} \frac{\partial h_k}{\partial x_i} \frac{\partial^2 L}{\partial h_k \partial h_l} \frac{\partial h_l}{\partial x_j}$$

By neglecting the first term above, the matrix with elements

$$G_{i,j} = \sum_{k,l} \frac{\partial h_k}{\partial x_i} \frac{\partial^2 L}{\partial h_k \partial h_l} \frac{\partial h_l}{\partial x_j}$$

is positive semidefinite:

$$\sum_{i,j} z_i G_{i,j} z_j = \sum_{k,l} \underbrace{\sum_{i=1} \frac{\partial h_k}{\partial x_i}}_{\gamma_k} \underbrace{\frac{\partial^2 L}{\partial h_k \partial h_l}}_{M_{k,l}} \underbrace{\sum_j \frac{\partial h_l}{\partial x_j}}_{\gamma_l} = \sum_{k,l} \gamma_k M_{k,l} \gamma_l \geq 0$$

Generalised Gauss Newton

- Since the Generalised Gauss-Newton matrix is positive semidefinite, the update

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \epsilon \mathbf{G}^{-1} \mathbf{g}$$

  where

$$g_i \equiv \frac{\partial f}{\partial x_i}$$

  is guaranteed to lower the function value f (provided ϵ is sufficiently small).
- Since $\frac{\partial L}{\partial h_k}$ will be small close to a minimum, this update will approach that of the Newton method as we converge.

Levenberg-Marquardt algorithm

In the Gauss-Newton method the update requires us to solve the linear system

$$\mathbf{C}\boldsymbol{\Delta} = \nabla E$$

- However, if we don't have many training datapoints, then $\mathbf{C}$ will be rank deficient (non-invertible).
- Even then, in practice, $\mathbf{C}$ may be ill-conditioned (it has a large range of eigenvalues) which makes solving the system numerically unstable.

LM algorithm

The idea is to regularise the problem and solve (using conjugate gradients)

$$(\mathbf{C} + \lambda \mathbf{I})\,\boldsymbol{\Delta} = \nabla E$$

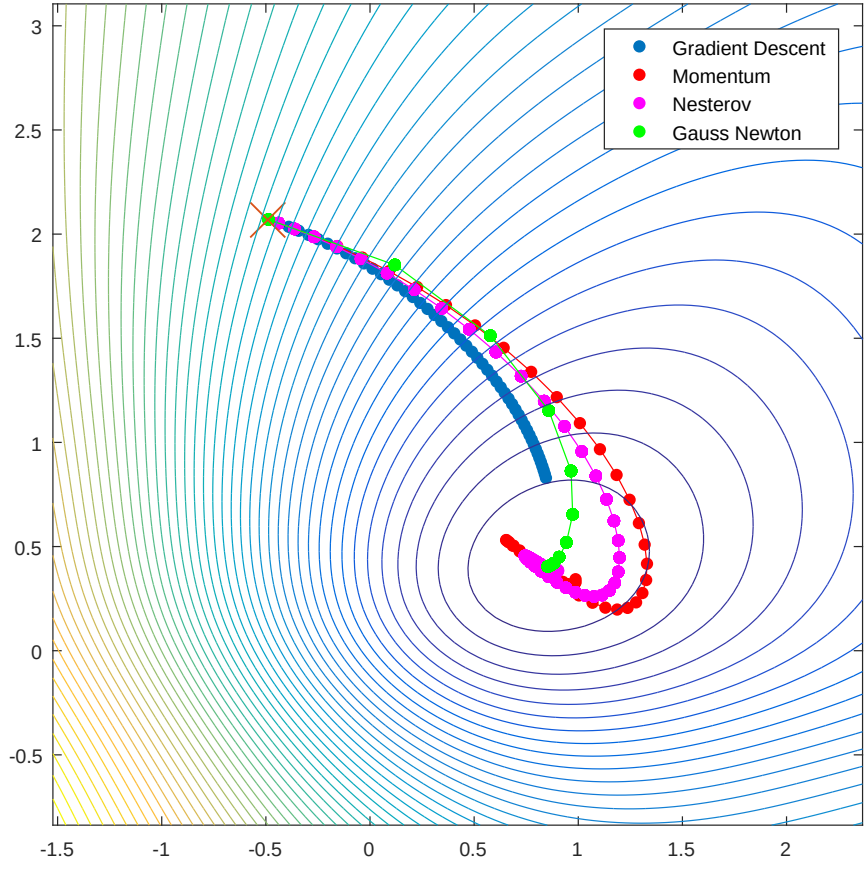
for a suitably chosen positive scalar λ.

- One can show that this is effectively defining a 'trust region' since this is equivalent to minimising (with respect to $\boldsymbol{\Delta}$)

$$E(\mathbf{w}) + \boldsymbol{\Delta}^{\mathsf{T}}\nabla E + \frac{1}{2}\boldsymbol{\Delta}^{\mathsf{T}}\mathbf{C}\boldsymbol{\Delta} + \frac{\lambda}{2}\boldsymbol{\Delta}^{\mathsf{T}}\boldsymbol{\Delta}$$

  where the final regularising terms constrains the scale of the update $\boldsymbol{\Delta}$.
- This makes sense since we only trust the Taylor expansion in the vicinity of $\mathbf{w}$.

Demo: Gauss Newton on a sum of squares loss



Gauss Newton using a learning rate of $\epsilon = 0.95$ and regulariser $\lambda = 0.05$. The Gauss Newton method converges in fewer iterations, but each iteration is more costly.

Natural Gradient (Gauss-Newton style for KL loss)

Consider an objective that we wish to minimise wrt θ of the form

$$E(\theta) \equiv KL(p(x)|q(x|\theta)) = -\langle \log q(x|\theta) \rangle + \text{const.}$$

where expectation is wrt $p(x)$. The gradient wrt θ is

$$g_i = -\left\langle \frac{\partial \log q(x|\theta)}{\partial \theta_i} \right\rangle$$

We define the matrix positive definite matrix curvature matrix

$$C_{i,j} = \left\langle \frac{\partial \log q(x|\theta)}{\partial \theta_i} \frac{\partial \log q(x|\theta)}{\partial \theta_j} \right\rangle$$

Then the update is $C^{-1}g$ so that

$$\theta(t+1) = \theta(t) - \epsilon C^{-1} g$$

This is analogous to the Gauss-Newton method and guarantees a reduction in the KL (for small ϵ).

Natural Gradient Invariance

Consider a transformation $\theta_i = f_i(\phi)$. Then

$$g_i^\phi = -\frac{\partial \langle \log q(x|f(\phi)) \rangle}{\partial \phi_i} = -\left\langle \sum_j \frac{\partial \log q(x|\theta)}{\partial \theta_j} \frac{\partial \theta_j}{\partial \phi_i} \right\rangle = \left[J^\mathsf{T} \left\langle g^\theta \right\rangle \right]_i$$

where $J_{i,j} = \frac{\partial \theta_i}{\partial \phi_j}$. Similarly,

$$C_{i,j}^\phi = \left\langle \frac{\partial \log q(x|f(\phi)}{\partial \phi_i} \frac{\partial \log q(x|f(\phi))}{\partial \phi_j} \right\rangle = \left\langle \left[J^\mathsf{T} g^\theta \right]_i \left[J^\mathsf{T} g^\theta \right]_j \right\rangle$$

In matrix notation

$$C^\phi = J^\mathsf{T} \left\langle g^\theta \left(g^\theta \right)^\mathsf{T} \right\rangle J$$

Hence the update

$$\delta^\phi = \left(C^\phi \right)^{-1} g^\phi = \left(J^\mathsf{T} \left\langle g^\theta \left(g^\theta \right)^\mathsf{T} \right\rangle J \right)^{-1} J^\mathsf{T} \left\langle g^\theta \right\rangle = J^{-1} \left(\left\langle g^\theta \left(g^\theta \right)^\mathsf{T} \right\rangle \right)^{-1} \left\langle g^\theta \right\rangle$$

- The ϕ update corresponds to a new point in the θ space

$$\begin{aligned}
\theta_i^{new} &= f_i(\phi + \epsilon\delta^\phi) \\
&\approx f_i(\phi) + \epsilon \sum_j \delta_j^\phi \frac{\partial f_i}{\partial \phi_j} \\
&= \theta_i + \epsilon \sum_j J_{ij} \left[J^{-1}\delta^\theta\right]_j \\
&= \theta_i + \epsilon\delta_i^\theta
\end{aligned}$$

- The final expression is the update to θ that we would have made in the θ coordinate system.
- Hence, *up to first order* the parameter update is independent of the coordinate system.
- This is not the case for Newton's method (see next page).

In Newton's method, the update is (just for a scalar for simplicity)

$$\begin{aligned}\phi(t+1) &= \phi(t) - \epsilon \left(\frac{\partial^2 E}{\partial \phi^2}\right)^{-1} \frac{\partial E}{\partial \phi} \\ &= \phi(t) - \epsilon \left(\frac{\partial}{\partial \phi}\left(\frac{\partial E}{\partial \theta}\frac{\partial \theta}{\partial \phi}\right)\right)^{-1} \frac{\partial E}{\partial \theta}\frac{\partial \theta}{\partial \phi} \\ &= \phi(t) - \epsilon \left(\frac{\partial E}{\partial \theta}\frac{\partial^2 \theta}{\partial \phi^2} + \left(\frac{\partial \theta}{\partial \phi}\right)^2 \frac{\partial^2 E}{\partial \theta^2}\right)^{-1} \frac{\partial E}{\partial \theta}\frac{\partial \theta}{\partial \phi}\end{aligned}$$

The term $\frac{\partial E}{\partial \theta}\frac{\partial^2 \theta}{\partial \phi^2}$ prevents the invariance. If this were not present, we would have

$$\phi(t+1) = \phi(t) - \epsilon \frac{\partial \phi}{\partial \theta}\left(\frac{\partial^2 E}{\partial \theta^2}\right)^{-1} \frac{\partial E}{\partial \theta}$$

and

$$f(\phi(t+1)) \approx f(\phi(t)) - \epsilon \frac{\partial f}{\partial \phi}\frac{\partial \phi}{\partial \theta}\left(\frac{\partial^2 E}{\partial \theta^2}\right)^{-1} \frac{\partial E}{\partial \theta}$$

which gives (to first order) the Newton update in the θ coordinate system

$$\theta(t+1) \approx \theta(t) - \epsilon \left(\frac{\partial^2 E}{\partial \theta^2}\right)^{-1} \frac{\partial E}{\partial \theta}$$

Note that the invariance still holds for linear transformations and as we approach the optimum since then the term $\frac{\partial E}{\partial \theta}\frac{\partial^2 \theta}{\partial \phi^2}$ becomes small.