

optimisation

Patrick van der Smagt

January 2012

discrete optimisation

There are certainly many discrete optimisation problems where no derivatives are known; it's a field of research in itself. Here is one example:

scheduling I want to measure arm sEMG in 0g. This is done by flying parabolas in a specially prepared airplane. I have 22s of zero-gravitation time during each parabola, for a total of 31 parabolas per flight. I want to optimally use this limited time.

How can I schedule my 6 different experiments optimally so as to minimise the effect of subject learning? (Vogel & van der Smagt, 2011)

continuous optimisation

There are certain optimisation problems, in which a continuous set of input values has to be mapped to a continuous set of output values. Here is an example:

Recording 10 values of EMG, how can we map these to the grasp force of index finger vs. thumb, or grasp force of the whole hand?

We gather data $(\mathbf{x}_i, \mathbf{z}_i)$ where \mathbf{x}_i are EMG electrode values and \mathbf{z}_i are force sensor values. We concoct a general model $\mathcal{M}(\mathbf{x}, \mathbf{w})$ with which we minimise an error $\sum_i \|\mathcal{M}(\mathbf{x}, \mathbf{w}) - \mathbf{z}_i\|$, i.e., minimise the error between the approximated output and the measured output, by varying our parameters \mathbf{w} . There must be an *optimal* \mathbf{w} ! (Castellini & van der Smagt, Biol. Cyb. 2009)

remember linear regression

suppose we have data $\mathcal{D} = \{\mathbf{x}_i, \mathbf{z}_i = \mathcal{F}(\mathbf{x}_i)\}$

Optimisation is finding the \mathbf{w} which minimise an error, typically

$$E = \sum_{(\mathbf{x}_i, \mathbf{z}_i) \in \mathcal{D}} \|\mathbf{z}_i - \mathcal{M}(\mathbf{w}, \mathbf{x}_i)\| \quad (1)$$

We modelled the data with basis functions ϕ by writing

$$\mathcal{F}(\mathbf{x}) = \mathcal{M}(\mathbf{x}) + \eta \quad (2)$$

$$= \sum_j w_j \phi_j(\mathbf{x}) + \eta \quad (3)$$

where η is the *noise, error, ...*

remember our basis functions

examples of basis functions:

(a) polynomials $\phi_i(x) = x^i$ (*why are these problematic?*)

(b) orthogonal polynomials

$$\phi_i(x) = x\phi_{i-1}(x) - \alpha_i\phi_{i-1}(x) - \beta_{i-1}\phi_{i-2}(x) \quad (4)$$

where α and β are chosen such that

$$\sum_{k=1}^n \phi_i(x_k)\phi_{j \neq i}(x_k) = 0 \quad (5)$$

which makes orthogonal basis functions (Forsythe, 1957; cf. Gram-Schmidt orthogonalisation)

remember our basis functions

(c) Gaussians $\phi_i(\mathbf{x}) = \exp(-\mathbf{x}^2)$ (why are these problematic?)

Remember the general form

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{\sqrt{2|\pi\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right) \quad (6)$$

linear optimisation does not suffice

we need to move to a more general model

$$\mathcal{M}(\mathbf{x}) = \sum_j \phi_j(\mathbf{w}_j, \mathbf{x}) \quad (7)$$

Since the parameters \mathbf{w} are no longer outside of ϕ , we cannot do linear optimisation (e.g., SVD) anymore. We have to solve a number of nonlinear equations since \mathcal{M} is nonlinear in \mathbf{w} . Why is that? Well, if you look at $\partial E / \partial \mathbf{w}$, you will find \mathbf{w} in the result (try it for some specific form of ϕ).

nonlinear example

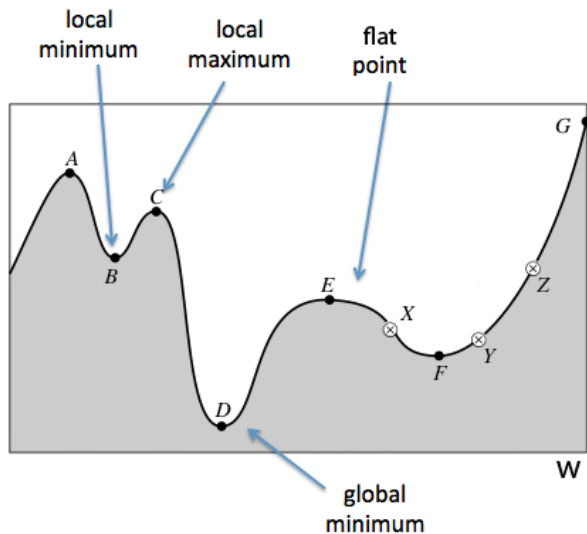
Take $\phi(w_1, w_2, x) = w_1 \exp(-w_2 x^2)$, then

$$E = \sum_{(x_i, z_i) \in \mathcal{D}} \|z_i - w_1 \exp(-w_2 x_i^2)\| \quad (8)$$

How do we optimise this? We are looking for $\arg \min_{w_1, w_2} E$

the value of E

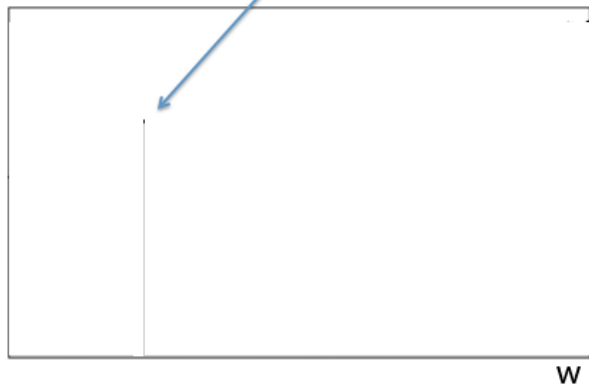
we are interested in finding $\arg \min_w E$



using local information $E(\mathbf{w})$

we are interested in finding $\arg \min_{\mathbf{w}} E$

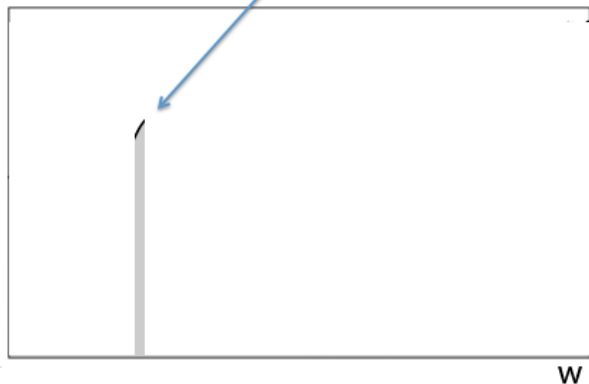
we usually only have local information



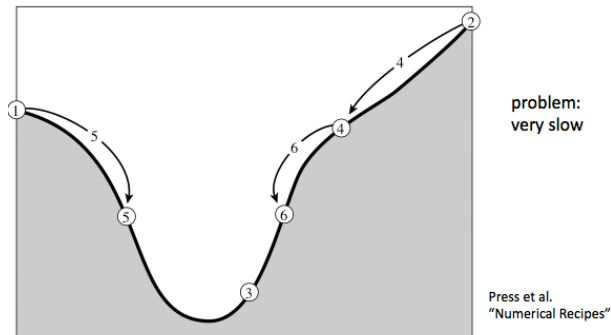
using local information $E(\mathbf{w})$ as well as $E'(\mathbf{w})$

we are interested in finding $\arg \min_{\mathbf{w}} E$

we usually only have local information



a simple method of finding a minimum

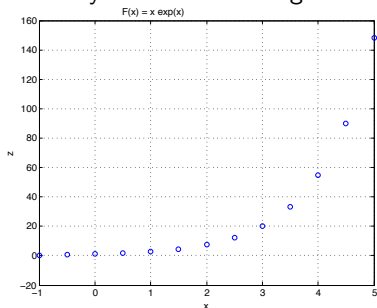


"Successive bracketing of a minimum. The minimum is originally bracketed by points 1,3,2. The function is evaluated at 4, which replaces 2; then at 5, which replaces 1; then at 6, which replaces 4. The rule at each stage is to keep a center point that is lower than the two outside points. After the steps shown, the minimum is bracketed by points 5,3,6."

using the gradient \mathbf{g} of E

method # 1: use everything you have. You may want to use a high-order (2nd, 3rd, ...) polynomial to match your E and E' data points, and then analytically compute the minimum.

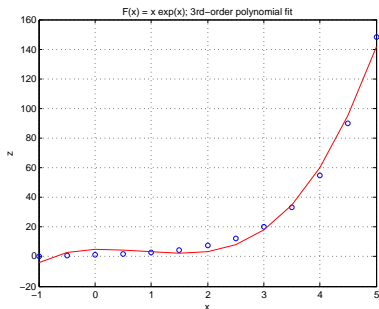
But: remember how unstable such higher-order polynomials can be! Close to the minimum, such errors become apparent, and noise and insufficient approximation can spoil your meal.



using the gradient \mathbf{g} of E

method # 1: use everything you have. You may want to use a high-order (2nd, 3rd, ...) polynomial to match your E and E' data points, and then analytically compute the minimum.

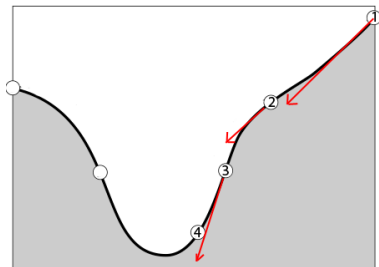
But: remember how unstable such higher-order polynomials can be! Close to the minimum, such errors become apparent, and noise and insufficient approximation can spoil your meal.



method # 2: use derivatives to find directions. We are more conservative: we use the direction of the derivative to know in which direction to look

using the gradient \mathbf{g} of E

The direction in which to optimise is given by the gradient $-\mathbf{g}$



Searching the minimum by repeated evaluation of E and $\mathbf{g} \equiv E'$. The value of $-\mathbf{g}$ gives us a direction and in which we want to optimise. We change the parameter vector as follows:

$$\mathbf{u}_i = -\mathbf{g}_i \quad (9)$$

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha \mathbf{u}_i \quad (10)$$

we call α the **learning parameter**

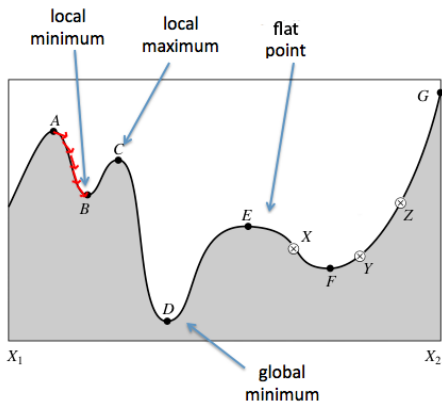
we call this method **steepest descent** or **gradient descent**

it belongs to the class of **greedy algorithms**

the value of α

a *too small* value for α has two drawbacks:

- ▶ we find the minimum more slowly
- ▶ we end up in local minima or saddle/flat points

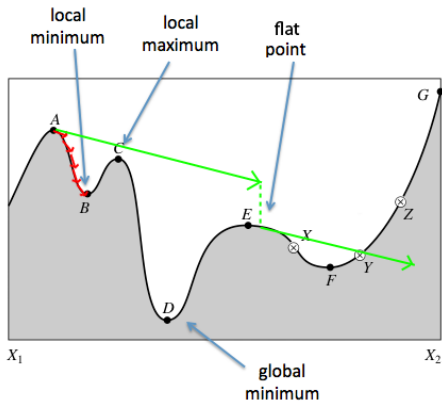


we need 5 steps to get stuck!

the value of α

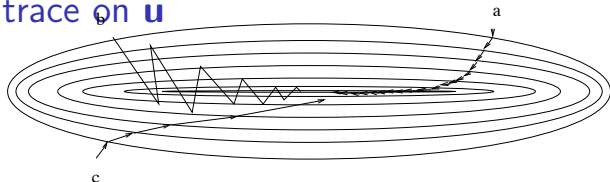
a *too large* value for α has one drawback:

- ▶ good chance you will never find a minimum; oscillations usually occur



we only need 2 steps to overshoot!

putting a trace on \mathbf{u}



a: small α ; b: large α ; c: the best of both worlds

$$\mathbf{u}_0 = -\mathbf{g}_0 \quad (11)$$

$$\mathbf{u}_i = -\mathbf{g}_i + \beta \mathbf{u}_{i-1} \quad (12)$$

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha \mathbf{u}_i \quad (13)$$

we call α the **learning rate**

we call β the **momentum**

We have previously seen: $\beta = \alpha^{-1} - 1$; we like to keep α small (below 0.1).

The good news: practice shows that this is *so much faster*

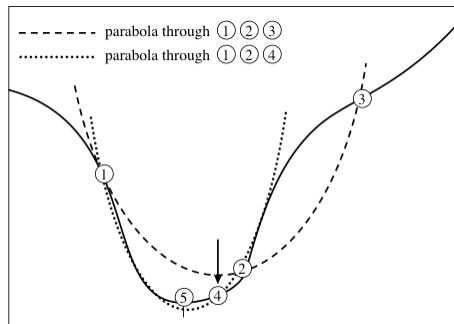
The bad news: remember the rule for $\beta = \alpha^{-1} - 1$? It assumes that ∇E does not change. That is a bad assumption!

can't we do a *bit* better than that?

Something that appears to be true (seeing is believing): many error functions have a smooth, quadratic-like form, especially as we get closer to the minimum.

We can exploit that “rule of thumb” to not just use first order information, but also include second order information.

Brent's method



problem:
quadratic model
of E is not always
realistic

Brent's method
combines the best
of both worlds

Press et al.
"Numerical Recipes"

"Convergence to a minimum by inverse parabolic interpolation. A parabola (dashed line) is drawn through the three original points 1,2,3 on the given function (solid line). The function is evaluated at the parabola's minimum, 4, which replaces point 3. A new parabola (dotted line) is drawn through points 1,4,2. The minimum of this parabola is at 5, which is close to the minimum of the function."

Brent again?

how we generalise Brent to multiple dimensions? The idea of “bracketing a minimum” does, of course, not generalise to multiple dimensions!

The “downhill simplex method” does something similar in multiple dimensions by minimising in hypertriangles. This appears to be inefficient and imprecise (but quick to program).

What about using derivatives? The “parabola” in Brent can be generalised to higher dimensions, but that is, in fact, the second-order approximation of our function E .

some formalism

we use the Taylor expansion of function E around \mathbf{w}_0 :

$$E(\mathbf{w}) = E(\mathbf{w}_0) + \overbrace{\sum_i \frac{\partial E}{\partial w_i} \Big|_{\mathbf{w}_0}}^{\text{gradient } -\mathbf{g}_i \text{ at } \mathbf{w}_0} (w_i - w_{0i}) + \frac{1}{2} \underbrace{\sum_{i,j} \frac{\partial^2 E}{\partial w_i \partial w_j} \Big|_{\mathbf{w}_0}}_{\text{Hessian } H_{ij} \text{ at } \mathbf{w}_0} (w_i - w_{0i})(w_j - w_{0j}) + \dots \quad (14)$$

$$\hat{E}(\mathbf{w}) := c - \mathbf{g}^T (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_0)^T H (\mathbf{w} - \mathbf{w}_0) \quad (15)$$

The **Hessian** is the matrix of second derivatives. **Can you grasp H ?**

So there you go: we have a closed form in which we can still do very efficient optimisation in many dimensions by approximating E by a parabola. We just need to compute and inverse H , and then we can directly compute where the minimum is supposed to be. An iterative approach can take care of rounding errors.

knowing the Hessian



Starting at point \mathbf{w}_0 , we want to find the point \mathbf{w} where a minimum is located, i.e., where $\mathbf{g} \equiv \nabla E(\mathbf{w}) = 0$. We find that

$$\hat{E}(\mathbf{w}) = \hat{E}(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)\nabla E(\mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T H(\mathbf{w} - \mathbf{w}_0)$$



when differentiating to \mathbf{w} we get

$$\nabla \hat{E}(\mathbf{w}) = \nabla \hat{E}(\mathbf{w}_0) + H(\mathbf{w} - \mathbf{w}_0) \quad (16)$$

We are looking for the point when $\nabla \hat{E}(\mathbf{w}) = 0$, so

$$(\mathbf{w} - \mathbf{w}_0) = -H^{-1}\nabla \hat{E}(\mathbf{w}_0) \quad (17)$$

knowing the Hessian

We are looking for the point when $\nabla \hat{E}(\mathbf{w}) = 0$, so

$$(\mathbf{w} - \mathbf{w}_0) = -H^{-1} \nabla \hat{E}(\mathbf{w}_0) \quad (18)$$

Ergo, we need to compute H^{-1}

We all know we do not want to invert such matrices, though. Apart from the fact that it is no fun, it is numerically unstable, leading to numerical errors which goof up the whole problem. It appears that H is usually badly conditioned (remember the vanishing gradient!), i.e., the largest eigenvalue is several (5, 10, 30, ...) orders of magnitude larger than the lowest, hindering you from a good inverse of H .

What can we do better?

not knowing the Hessian

quasi-Newton methods approximate the inverse Hessian by

$$\lim_{i \rightarrow \infty} B_i = H^{-1} \quad (19)$$



the equation according to Davidon-Fletcher-Powell (DFP)

$$B_{i+1} = B_i + \frac{(\mathbf{w}_{i+1} - \mathbf{w}_i) \times (\mathbf{w}_{i+1} - \mathbf{w}_i)}{(\mathbf{w}_{i+1} - \mathbf{w}_i) \cdot (\nabla E(\mathbf{w}_{i+1}) - \nabla E(\mathbf{w}_i))} \\ - \frac{[B_i \cdot (\nabla E(\mathbf{w}_{i+1}) - \nabla E(\mathbf{w}_i))] \times [B_i \cdot (\nabla E(\mathbf{w}_{i+1}) - \nabla E(\mathbf{w}_i))]}{(\nabla E(\mathbf{w}_{i+1}) - \nabla E(\mathbf{w}_i)) \cdot B_i \cdot (\nabla E(\mathbf{w}_{i+1}) - \nabla E(\mathbf{w}_i))}$$
$$B_0 = I$$



the additional

term according to Broyden-Fletcher-Goldfarb-Shanno (BFGS):

$$\dots + (\nabla \hat{E}(\mathbf{w}_{i+1}) - \nabla \hat{E}(\mathbf{w}_i)) \cdot B_i \cdot (\nabla \hat{E}(\mathbf{w}_{i+1}) - \nabla \hat{E}(\mathbf{w}_i)) \mathbf{v} \times \mathbf{v}$$

where $\mathbf{v} = \dots$

Ref: Polak, E. 1971, Computational Methods in Optimization (NY: Academic Press)

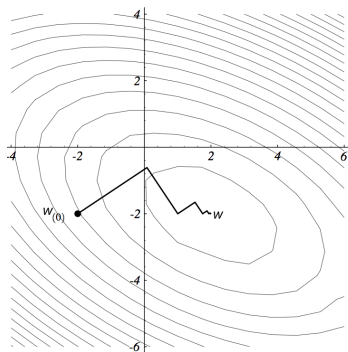
time for a short summary

- ▶ Newton's method needs to invert H , that is problematic
- ▶ quasi-Newton methods approximate H^{-1} , of which various methods exist (DFB and BFGS are most prominent)
- ▶ how many elements does H carry?

think of computational cost $O(N^3)$ and storage cost $O(N^2)$ where N is *the number of weights*.

last trick in the box

look at this steepest-descent optimisation



If we make sure that

- ▶ when moving along \mathbf{u}_i we reach a minimum *in that direction*;
- ▶ subsequent gradients $\mathbf{g}_i, \mathbf{g}_{i+1}$ are perpendicular

then we will reach the minimum of \hat{E} in $\text{length}(\mathbf{w})$ steps!



how is that done?

remember our optimisation direction \mathbf{u}

we want the current search direction perpendicular to the next gradient:

$$\mathbf{u}_i \cdot \mathbf{g}_{i+1} = 0 \quad (20)$$

and the one after that:

$$\mathbf{u}_i \cdot \mathbf{g}_{i+2} = 0 \quad (21)$$

we can now require that

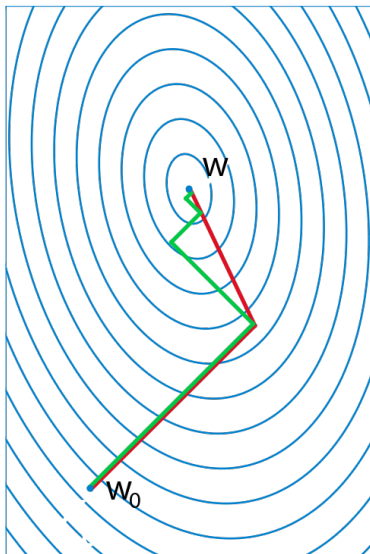
$$0 = \mathbf{u}_i \cdot (\mathbf{g}_{i+1} - \mathbf{g}_{i+2}) \quad (22)$$

we have seen that $\mathbf{g}_{i+2} = \mathbf{g}_{i+1} + H \cdot (\mathbf{w}_{i+2} - \mathbf{w}_{i+1})$ therefore

$$0 = \mathbf{u}_i \cdot H \cdot (\mathbf{w}_{i+2} - \mathbf{w}_{i+1}) = \mathbf{u}_i \cdot H \cdot \mathbf{u}_{i+1} \quad (23)$$

in this case we call \mathbf{u}_i and \mathbf{u}_{i+1} **conjugate**

it looks like this





we're almost there

remember also that we defined the next minimisation direction as

$$\mathbf{u}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{u}_i \quad (24)$$

now, to make \mathbf{u}_i and \mathbf{u}_{i+1} conjugate:

$$0 = \mathbf{u}_i \cdot H \cdot \mathbf{u}_{i+1} \quad (25)$$

$$= \mathbf{u}_i \cdot H \cdot (\beta_i \mathbf{u}_i - \alpha_i \mathbf{g}_{i+1}) \quad (26)$$

we take



$$\beta_i = \frac{\mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i} \quad (27)$$

which is called Fletcher-Reeves conjugate gradient.

Can you qualitatively understand what it does?



We can also compute the optimal α as

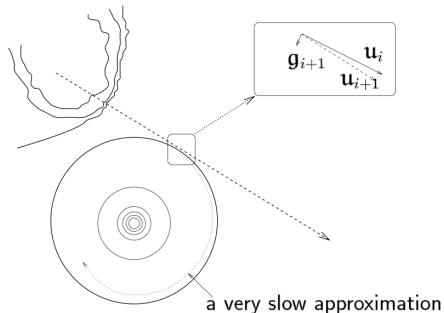
$$\alpha_i = \frac{\mathbf{g}_i \cdot \mathbf{g}_i}{\mathbf{u}_i \cdot H \cdot \mathbf{u}_i} \quad (28)$$

yet: we never have to compute α : use line minimisation instead!

this is extremely cool!

we have found a first-order method to implement a second-order minimisation approach. By using *gradient descent* with carefully chosen learning and momentum parameters, we get results as good as when we had to compute the (inverse) Hessian.

did we forget something?
ah, the Taylor approximation... so there is Polak-Ribière CG, Powell restarts, ... a whole bag of tricks from the 1960's and 1970's



Levenberg-Marquardt

often, the second-order information is so poor that it better be ignored. Marquardt introduced a simple method to take care of ignoring the Hessian when it gives us bad luck.

Define

$$H'_{jj} = H_{jj}(1 + \lambda) \quad (29)$$

If λ is large, the Hessian will be diagonally dominant, and we add a “disturbance” in the direction of the gradient.

Now an algorithm can be

- ▶ if $E(\mathbf{w} + \mathbf{u}') > E(\mathbf{w})$, increase λ by a factor 10 or so;
- ▶ if $E(\mathbf{w} + \mathbf{u}') < E(\mathbf{w})$, decrease λ by a similar large factor.

a derived approach exists called *scaled conjugate gradient* (Møller, 1993), which implements this trick in cg and thus avoids the line search.

it works pretty well, but. . .

- ▶ what about global optima
- ▶ what about saddle or flat points

simulated annealing

relates to a physics principle called *annealing*: by slowly decreasing the temperature of a fluid, the molecules slowly regularise towards a crystalline state

algorithm:

1. choose a random \mathbf{w}_0 and evaluate $E_0 \equiv E_{\mathbf{w}_0}$; $i = 1$
2. choose a random \mathbf{u}_i and set $\mathbf{w}_i = \mathbf{w}_{i-1} + \mathbf{u}_i$
3. if $E_i \leq E_{i-1}$ (**improvement**) then accept the new point: $i = i + 1$
4. if $E_i > E_{i-1}$ (**worse**) then accept the new point with probability $P = \exp[-(E_i - E_{i-1})/T]$: $i = i + 1$
5. $T \leftarrow f(T)$; goto 2

global minimisation has been proven, provided the decrease of temperature T is infinitesimal.

S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, Science, 220(4598): 671-680, 1983

other methods

- ▶ genetic methods
- ▶ exhaustive search
- ▶ random search
- ▶ many conferences on (global) optimisation

In summary, we can greatly improve on optimisation methods by some careful analysis and algebra. However, that does not solve our underlying problem—if \mathcal{M} is badly chosen, we are still very inefficient!

references

books

Press WH, Teukolsky SA, Vetterling WT, Flannery BP, *Numerical Recipes*, Cambridge University Press, 1988–

Stoer J, Bulirsch J, *Introduction to Numerical Analysis*, Springer, 1980–

Polak E, *Computational Methods in Optimization*, Academic Press, 1971

papers

Forsythe GE, “Generation and use of orthogonal polynomials for data-fitting with a digital computer”, *J. Soc. Indust. Appl. Math.* 5 (2), 1957

Møller M, “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning”, *Neural Networks* 6:525–533, 1993

van der Smagt P, “Minimisation methods for training feedforward neural networks”, *Neural Networks* 7:1–11, 1994