Basic optimization methods, part 2

Aapo Hyvärinen

for the NNDL course, 19 March 2025

- Stochastic gradient method
- Newton's method
- Conjugate gradient
- Comparisons, conclusions

[There will be more about NN specific optimization in later lectures]

Literature:
Prince's book Sections 6.2–6.3 (only part of these slides);
Taylor expansion in multivariate case is explained under this link

Stochastic gradient method: theoretical definition

- Consider $\mathbf{x}$, a *random* vector
- Assume we want to maximize some *expectation*

$$\max_{\mathbf{w}} f(\mathbf{w}) := \max_{\mathbf{w}} \mathrm{E}\{g(\mathbf{w}, \mathbf{x})\} \tag{1}$$

- Basic gradient method

$$\mathbf{w} \leftarrow \mathbf{w} + \mu \mathrm{E}\{\nabla_{\mathbf{w}} g(\mathbf{w}, \mathbf{x})\} \tag{2}$$

(Expectation is a linear operation, so we can take gradient inside the expectation: $\nabla \mathrm{E} g = \mathrm{E} \nabla g$)

- Idea in stochastic gradient method: we don't (need to) compute the expectation before taking the gradient step.
- For *one single observation* $\tilde{\mathbf{x}}$ of $\mathbf{x}$, do gradient iteration for $\tilde{\mathbf{x}}$ only, omitting expectation:

$$\mathbf{w} \leftarrow \mathbf{w} + \mu \nabla_{\mathbf{w}} g(\mathbf{w}, \tilde{\mathbf{x}}) \tag{3}$$

- Works because update will be *on the average* equal to original gradient ("stochastic / Monte Carlo approximation")

Stochastic gradient method: definition for sample

- In practice, we have sample (dataset) $\mathbf{x}_1, \ldots, \mathbf{x}_N$
- Expectation is computed as *sample average*; our problem is

$$\max_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{w}, \mathbf{x}_i) \tag{4}$$

- Practical formulation of stochastic gradient:
  For *each data point* $\mathbf{x}_i$, do the stochastic gradient iteration

$$\mathbf{w} \leftarrow \mathbf{w} + \mu \nabla_{\mathbf{w}} g(\mathbf{w}, \mathbf{x}_i) \tag{5}$$

  randomly choosing the sample index i at every iteration.
- To reduce randomness, "mini-batches" are often used:
  Compute gradient for a small "subsample" (say, 100 points)

$$\mathbf{w} \leftarrow \mathbf{w} + \mu \frac{1}{\text{size}(M)} \sum_{i \in M} \nabla_{\mathbf{w}} g(\mathbf{w}, \mathbf{x}_i) \tag{6}$$

  where the set of indices M is redrawn for each step.
  - Similar to reducing μ, but often faster (e.g. parallelizable)

Stochastic gradient method: simple example

- Log-likelihood for estimating mean of Gaussian 1D variable:

$$\sum_{i=1}^{N} -\frac{1}{2}(x_i - \alpha)^2 \tag{7}$$

  from sample $x_1, \ldots, x_N$.
- Defining $g(\alpha, x) = -\frac{1}{2}(x - \alpha)^2$, we optimize

$$\max_{\alpha} \sum_{i=1}^{N} g(\alpha, x_i) \tag{8}$$

  (Could divide by N to comply with the notation above.)
- Taking derivative: $\nabla_\alpha g(\alpha, x) = x - \alpha$ we get iteration:

$$\alpha \leftarrow \alpha + \mu(x_i - \alpha) \tag{9}$$

  for an index i, randomly chosen at each iteration
- Intuitively, each x_i "pulls" α towards itself
  $\rightarrow$ converges to their mean

Stochastic gradient method: pros and cons

- Basic justification: stochastic gradient faster to compute
  - No need to sum over data points $\Rightarrow$ reduction in complexity
  - Particularly useful in ML/DL, when you have a big data set
- Further utility: weaker tendency to get stuck in local maxima
  - Kind of randomly exploring the space
- But: step size has to be much smaller than in ordinary gradient to compensate for the random fluctuations
  - Need *many* more steps $\rightarrow$ speedup reduced
- Further problems with stochastic gradient:
  1. Not clear how to diagnose convergence
     - Above, we looked at e.g. norm of gradient, but norm of stochastic gradient is too random
  2. More difficult to adapt step size
     - objective function only given as stochastic approximation, cannot really compare value of objective for different step sizes
- (In DL, often called SGD, stochastic gradient descent)

Convergence theory of stochastic gradient method

- For fixed μ:
  - cannot converge to a single point
  - only *expectation* of stochastic gradient zero at maximum
  - will have random fluctuations around the maximum
  - can only converge to a *distribution* around the maximum
  - (ordinary gradient method does converge with fixed, well-chosen μ)
- But: We can use step size "schedules", i.e. *time-dependent* μ_t
- For suitable step size schedules:
  - can, at least in theory, converge to the optimal point
  - essential to fulfill the two conditions (Robbins-Monro):

$$\sum_{t=0}^{\infty} \mu_t = \infty, \quad \sum_{t=0}^{\infty} \mu_t^2 < \infty, \tag{10}$$

  - e.g.: if step size decreased as $\sim 1/t$ where t is iteration count
  - (this theory only considers local optimization, global optima another question)

Practical tricks for making convergence better

- In practice, systematic reduction of μ as in Robbins-Monro conditions is not too often done
- Simple related tricks to reduce randomness in the result:
  1. Reducing the step size at some point(s) (e.g. half-way through) is a good idea and often done
     - Equivalently (kind of), increase minibatch size
  2. Take average over the final iterations: denoting final iteration by T, average $\mathbf{w}_T, \mathbf{w}_{T-1}, \ldots \mathbf{w}_{T-\tau}$ for some number τ (e.g. 100) and give this as output (i.e. as the optimizing $\mathbf{w}$)
- Note: here we are talking about *predefined* schedules of μ_t, an *adaptive* μ is a different thing (not considered in this lecture)

Using momentum to improve stochastic gradient

- Widely used method to reduce random fluctuations
- Idea: Update **w** based on an average of current stochastic gradient and recent ones
- Computationally simple approach is to take moving average:

$$\mathbf{s} \leftarrow (1-\alpha)\mathbf{s} + \alpha \nabla_{\mathbf{w}} g(\mathbf{w}, \mathbf{x}_i) \tag{11}$$

  for a small positive constant α.
  Then: use **s** instead of gradient in gradient update
- Effect is similar to using minibatches, but now taking moving average "smoothly" over recent history
- Many variants exist; above is only a rough sketch
- Can be combined with minibatches: moving average of gradients in minibatches

Newton's method: starting point

Maths background under this link (same as on title page)

- Classic alternative ("improvement"?) to gradient method
- Starting point: the optima of an objective function are found in points where the gradient is zero
- So, we can try to solve the system of equations given by

$$\frac{\partial f}{\partial w_1}(\mathbf{w}) = 0$$

$$\vdots$$

$$\frac{\partial f}{\partial w_n}(\mathbf{w}) = 0$$

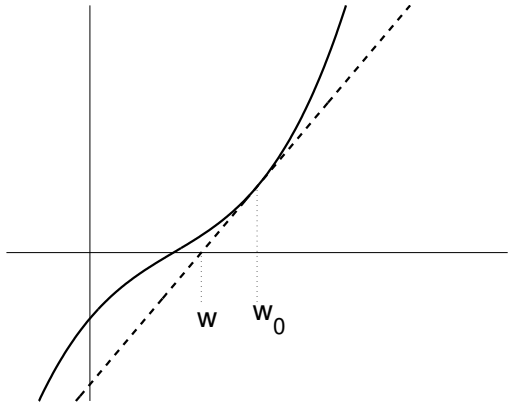
- Newton's method is originally a method for solving any system of equations
- We use it here for solving the optimization problem by solving the above system of equations

Newton's method (for solving an equation)

- Consider case of solving a nonlinear equation, just 1D for illustration
- Idea: approximate the function linearly using its derivative

$$g(w) \approx g(w_0) + g'(w_0)(w - w_0) \tag{12}$$

- Find point w where g is zero in this approximation
- Geometrically, approximate the graph of the function using its tangent, whose slope is given by the derivative.



- Iteratively apply this on new w, and iterate until convergence

Some background: linear approximations by differentials

- General principle: linear approximation of function $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^d$ by the Jacobian

$$\mathbf{g}(\mathbf{w}) \approx \mathbf{g}(\mathbf{w}_0) + \mathbf{J}\mathbf{g}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{13}$$

where $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J}_{ij}(\mathbf{w}) = \frac{\partial g_i}{\partial w_j}(\mathbf{w}) \tag{14}$$

- We apply the above on the gradient
- We get

$$\nabla f(\mathbf{w}) \approx \nabla f(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{15}$$

where $\mathbf{H}$, called the Hessian matrix, is the matrix of partial derivatives of gradient, or second partial derivatives of f:

$$\mathbf{H}_{ij}(\mathbf{w}) = \frac{\partial [\nabla f]_i}{\partial w_j}(\mathbf{w}) = \frac{\partial^2 f}{\partial w_i \partial w_j}(\mathbf{w}) \tag{16}$$

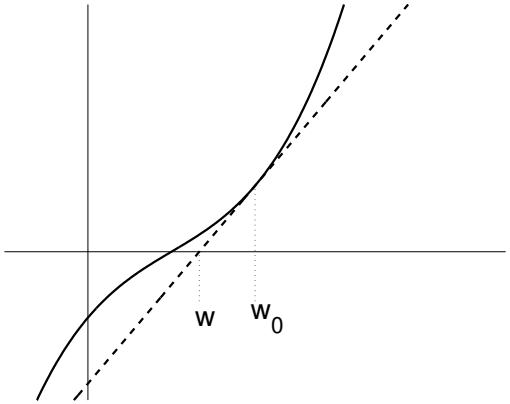
and it is equal to the *Jacobian of the gradient*

Deriving Newton's method for optimization

- On previous slide, we found approximation:

$$\nabla f(\mathbf{w}) \approx \nabla f(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{17}$$

  for $\mathbf{w}$ around $\mathbf{w}_0$.
- Now we can formalize the intuition above in high dimensions:



- Given some point $\mathbf{w}_0$, find the new point as the one for which this linear approximation is zero:

$$\nabla f(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) = \mathbf{0} \tag{18}$$

  which leads to

$$\mathbf{w} = \mathbf{w}_0 - \mathbf{H}(\mathbf{w}_0)^{-1} \nabla f(\mathbf{w}_0) \tag{19}$$

Resulting Newton's method for optimization

- Start from an initial point $\mathbf{w}_0$
  - typically randomly generated, just like in gradient method
- Iteratively update $\mathbf{w}$ according to

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}(\mathbf{w})^{-1} \nabla f(\mathbf{w}) \tag{20}$$

- Run until convergence
  - Same kind of diagnostics as in gradient method

Newton's method: simple example

- Recall formula:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}(\mathbf{w})^{-1} \nabla f(\mathbf{w}) \tag{21}$$

- Consider a 1D example

$$f(w) = 1 - w^4 \tag{22}$$

- We have easily

$$\nabla f(w) = f'(w) = -4w^3 \tag{23}$$

$$\mathbf{H} f(w) = f''(w) = -12w^2 \tag{24}$$

- And Newton's method:

$$w \leftarrow w - \frac{4w^3}{12w^2} = w - \frac{1}{3}w = \frac{2}{3}w \tag{25}$$

- Clearly, works very well: converges to zero which is optimum

More background: Quadratic approximation by Taylor

- Elementary case: For a function $f : \mathbb{R} \to \mathbb{R}$, well-known:

$$f(w) \approx f(w_0) + f'(w_0)(w - w_0) + \frac{1}{2} f''(w_0)(w - w_0)^2 \quad (26)$$

- Multivariate case: important to note there are different cases
  1. For a function $f : \mathbb{R}^d \to \mathbb{R}$: Second-order Taylor expansion is

$$f(\mathbf{w}) \approx f(\mathbf{w}_0) + \nabla f(\mathbf{w}_0)^T (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_0)^T \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0)$$

     where ∇f is gradient, $\mathbf{H}$ is Hessian matrix.
  2. For a function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$: Second-order Taylor expansion is

$$\mathbf{f}(\mathbf{w}) \approx \mathbf{f}(\mathbf{w}_0) + \mathbf{J}\mathbf{f}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) + \text{something too complicated}$$

     where $\mathbf{J}$ is Jacobian (generalizes previous slides, Eq. 13)

Alternative approach to Newton: second-order optimization

- We saw 2nd-order Taylor expansion of objective function:

$$f(\mathbf{w}) \approx f(\mathbf{w}_0) + \nabla f(\mathbf{w}_0)^T(\mathbf{w}-\mathbf{w}_0) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_0)^T\mathbf{H}(\mathbf{w}_0)(\mathbf{w}-\mathbf{w}_0)$$

- To derive the algorithm: Given the (current) point $\mathbf{w}_0$, find point $\mathbf{w}$ where this is optimized
- We recall simple formulae for the gradient:

$$\nabla_{\mathbf{w}}(\mathbf{z}^T\mathbf{w}) = \mathbf{z} \tag{27}$$

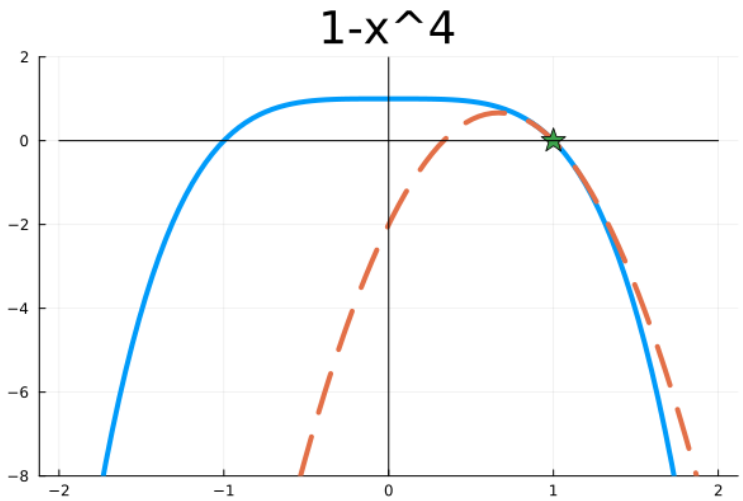
$$\nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{M}\mathbf{w}) = 2\mathbf{M}\mathbf{w} \tag{28}$$

- So, gradient of the above with respect to $\mathbf{w}$ is

$$\nabla f(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{29}$$

- Set gradient to zero $\rightarrow$ exactly same equation (18) and thus same method as above!

Illustration of Newton method as quadratic approximation



Quadratic approximation (red) of objective (blue) at the point given by the star.

Comparison of gradient method and Newton's method

Gradient method	Newton's method
$\mathbf{w} \leftarrow \mathbf{w} + \mu \nabla f(\mathbf{w})$	$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}(\mathbf{w})^{-1} \nabla f(\mathbf{w})$

- Similarity: adding some function of the gradient to $\mathbf{w}$
- Difference comes from (small) scalar μ vs. matrix $-\mathbf{H}(\mathbf{w}_0)^{-1}$
  1. In Newton's method the direction where $\mathbf{w}$ "moves" is not given by the gradient directly, but the gradient multiplied by the inverse of the Hessian.
  2. Resulting step in Newton is not necessarily small, can be large
  3. There is no *choice* of step size in Newton
     - Can be a good thing or a bad thing, see below
  4. In gradient method, one can choose between minimization and maximization by choosing the sign in the algorithm.
     In Newton's method, no such choice is possible:
     it just tries to find a local extremum in which the gradient is zero, can be either a minimum or a maximum.

Advantages of gradient method vs. Newton's method

- Newton's method needs computation of Hessian, and computing $\mathbf{H}(\mathbf{w})^{-1}\nabla f(\mathbf{w})$
  $\Rightarrow$ each step computationally much more expensive
  - one step of Newton method *cubic* in $\dim(\mathbf{w})$
  - one step of gradient method linear in $\dim(\mathbf{w})$, at least for simple gradients
  - (sidenote: to compute $\mathbf{M}^{-1}\mathbf{z}$, never compute the inverse explicitly, solve equation $\mathbf{M}\mathbf{w} = \mathbf{z}$ which is much faster)
- Due to the clever choice of the "step size", Newton's method promises to converge in much fewer iterations
- With gradient method, we have some guarantee that the objective is increased if the step size is small enough
- Newton's method can completely diverge and there is nothing we can do about it (next slide)

Divergence of Newton's method

- Consider the function

$$f(w) = \exp(-\frac{1}{2}w^2) \tag{30}$$

- A single maximum at $w = 0$.
- Easy to calculate:

$$f'(w) = -w \exp(-\frac{1}{2}w^2) \tag{31}$$

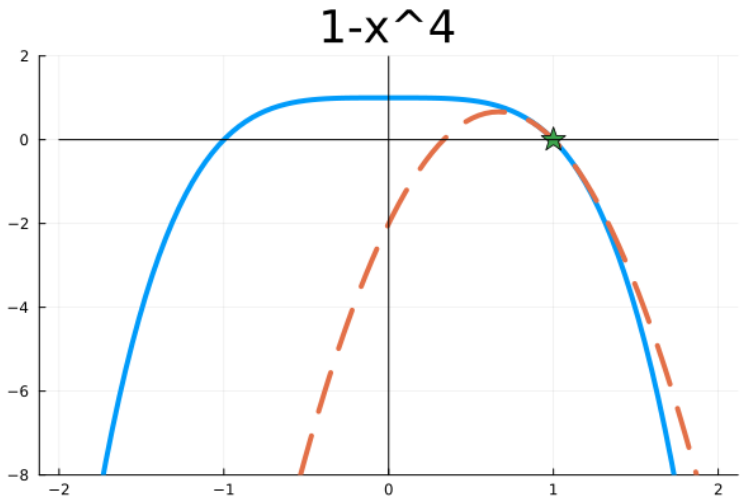
$$f''(w) = (w^2 - 1) \exp(-\frac{1}{2}w^2) \tag{32}$$

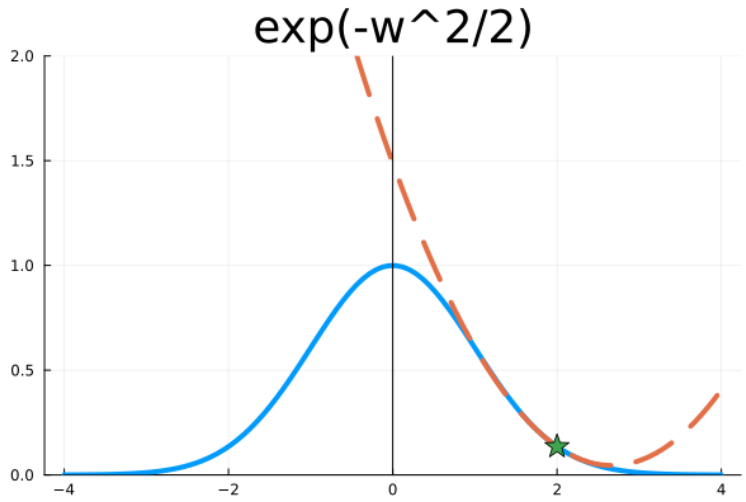
- Newton iteration is

$$w \leftarrow w + \frac{w}{w^2 - 1} \tag{33}$$

- Assume that we start the iteration at any point where $w > 1$.
- Then, the change $\frac{w}{w^2-1}$ is positive, which means that w is increased and it moves further and further away from zero!

Illustration of Newton method's success and failure





Quadratic approximation (red) of objective (blue)
at the point given by the star.

On the left, things go well. On the right, they go wrong.
Quadratic approximation can be very useful or completely useless.

Final word on applicability of Newton's method in DL

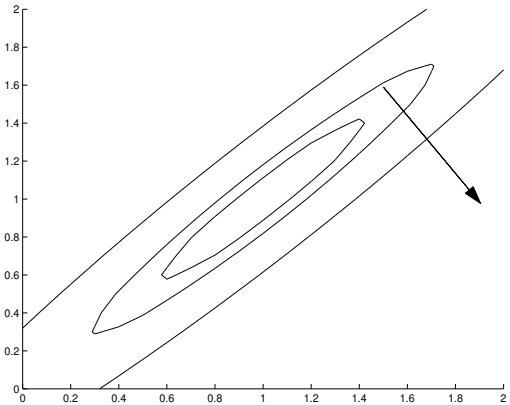
- In its *basic* form, Newton's method is *useless* in deep learning
  - Each step is very expensive to compute if $\dim(\mathbf{w})$ is large
  - Often diverges in practice
  - Not clear how to build stochastic version of method
- However, the theory is interesting when cleverly adapted
  - The method promises to find a really good step size
  - ... and even change the gradient direction to a better one
  - Methods which do some kind of *hybrid* of gradient method and Newton's method have proven useful.

  $\rightarrow$ "Second-order optimization methods" in deep learning (Arto will talk about this more)

  $\rightarrow$ Conjugate gradient method (next)

Conjugate gradient (CG) methods

- Often considered as the most efficient general-purpose optimization approach *in the non-stochastic case*
- Conjugate gradient methods try to find a direction which is better than the gradient, similarly to Newton's method
- Recall that the gradient direction is good for a very small step size, but not for a moderately large step size.



Here, gradient goes rather completely in the wrong direction due to the strongly "non-spherical" objective function.

- CG improves direction for such "badly-conditioned" functions

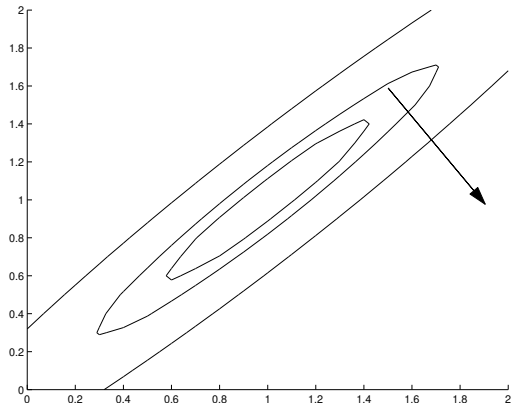
Conjugate gradient in detail (just to give you the flavor)

- Basic points:
  - store a "conjugate direction" $\mathbf{s}$
  - update it separately
  - do line search in that direction
- Algorithm:
  1. Take initial step ($t = 0$) like in ordinary gradient method
  2. $t \leftarrow t + 1$
  3. Calculate gradient: $\Delta \mathbf{w}_t = -\nabla_{\mathbf{w}} f(\mathbf{w}_t)$
  4. Compute β_t according to (e.g.) one of the following
     - Fletcher–Reeves: $\beta_t^{FR} = \frac{\Delta \mathbf{w}_t^T \Delta \mathbf{w}_t}{\Delta \mathbf{w}_{t-1}^T \Delta \mathbf{w}_{t-1}}$
     - Polak–Ribière: $\beta_t^{PR} = \frac{\Delta \mathbf{w}_t^T (\Delta \mathbf{w}_t - \Delta \mathbf{w}_{t-1})}{\Delta \mathbf{w}_{t-1}^T \Delta \mathbf{w}_{t-1}}$
  5. Update conjugate direction: $\mathbf{s}_t = \Delta \mathbf{w}_t + \beta_t \mathbf{s}_{t-1}$
  6. Perform a line search: optimize $\alpha_t = \arg\min_\alpha f(\mathbf{w}_t + \alpha \mathbf{s}_t)$,
  7. Finally, conjugate gradient iteration: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \mathbf{s}_t$
  8. If not converged, go back to 2

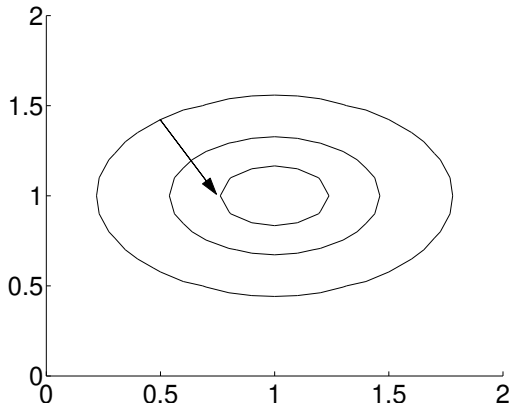
Conditioning of optimization problem

- Actually, why is it so difficult to optimize a) but not b) ?

a)



b)



- The problem in a) is that the contours are "elongated"
  - Must take very small steps to not go wrong
- Measured by *condition number* of the Hessian (at optimum)
  - Condition number $= \frac{|\text{largest eigenvalue}|}{|\text{smallest eigenvalue}|}$, (for symmetric matrix)
- In purely spherical case b), condition number is small (=1)
- In "elongated" case a), condition number is large (maybe 10)
- Large condition number $\rightarrow$ *"Badly conditioned problem"*
  - one reason why an optimization problem is particularly difficult

Badly conditioned problems: Newton and conjugate grad

- Consider a quadratic function as objective
  - Near optimum, any function is approximately quadratic
  - Typical approach to enable theoretical analysis
- Newton's method will directly find the optimum in a single step! (by construction)
- Conjugate gradient method will find optimum in max n steps in n-dimensional space
- In principle, this shows the superiority of those methods
  - but real objective functions are not quadratic, so...
  - the relevance of this analysis is limited;
  - still, empirically, conjugate gradient works well...
    - unless you have a huge data set, so that the evaluation of the true gradient (non-stochastic, for the whole data) is too slow

Alternating variables

- In some cases, the objective function is such that it depends on two "different" groups of variables, say $\mathbf{x}$ and $\mathbf{y}$.
- Then, it may happen that it is easy to optimize the objective with respect to $\mathbf{x}$ when *keeping* $\mathbf{y}$ *fixed*, or vice versa.
- Alternating variables method: alternate between maximization with respect to $\mathbf{x}$ and with respect to $\mathbf{y}$

$$\mathbf{x}(i+1) \leftarrow \arg\max_{\mathbf{x}} g(\mathbf{x}(i), \mathbf{y}(i)) \tag{34}$$

$$\mathbf{y}(i+1) \leftarrow \arg\max_{\mathbf{y}} g(\mathbf{x}(i+1), \mathbf{y}(i)) \tag{35}$$

  where i is the iteration count, and arg max denotes the argument which maximizes the objective function.
- Special case: each "group" could be just one variable $\rightarrow$ "coordinate descent/ascent"
- Kind of special case: Expectation-Maximization (EM) algorithm (not in this course)

Conclusion

- *Optimization in $\mathbb{R}^n$ is a very difficult problem!*
- Basic gradient method is reasonable and widely used in ML
- Stochastic gradient popular in DL, but use with caution!
- Choice of step size eternal problem
  - Adaptation rather straightforward in case of ordinary gradient
  - Adaptation more difficult with stochastic gradient
- Diagnosis of convergence important
  - Easy with ordinary gradient
  - Very difficult with stochastic gradient
- Newton & conjugate gradient can be very useful ... or not
- Local optima ubiquitous problem in deep learning
- Bad conditioning another problem
- I would prefer ordinary (conjugate?) gradient to stochastic
  - ... unless ordinary gradient is very expensive to compute, which only happens with huge datasets (millions of points?)
  - Unfortunately, 90% of deep learning people disagree with me ;)
  - On the other hand: stochastic gradient may help in generalization in supervised learning (see Arto's lectures)