Optimization

Machine Learning and Pattern Recognition

Chris Williams

School of Informatics, University of Edinburgh

October 2015

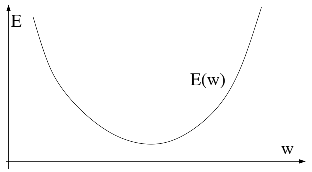
(These slides have been adapted from previous versions by Charles Sutton, Amos Storkey, David Barber, and from Sam Roweis (1972-2010))

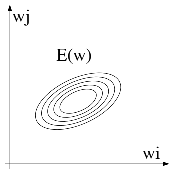
Outline

- Unconstrained Optimization Problems
  - Gradient descent
  - Second order methods
- Constrained Optimization Problems
  - Linear programming
  - Quadratic programming
- Non-convexity
- Reading: Murphy 8.3.2, 8.3.3, 8.5.2.3, 7.3.3.
  Barber A.3, A.4, A.5 up to end A.5.1, A.5.7, 17.4.1 pp 379-381.

Why Numerical Optimization?

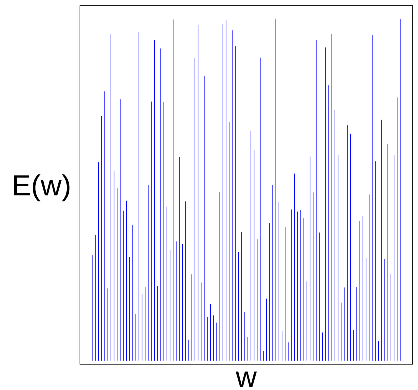
- Logistic regression and neural networks both result in likelihoods that we cannot maximize in closed form.
- End result: an "error function" $E(\mathbf{w})$ which we want to minimize.
- Note $\operatorname{argmin} f(\mathbf{x}) = \operatorname{argmax} - f(\mathbf{x})$
- e.g., $E(\mathbf{w})$ can be the negative of the log likelihood.
- Consider a fixed training set; think in weight (not input) space. At each setting of the weights there is some error (given the fixed training set): this defines an error surface in weight space.
- Learning $\equiv$ descending the error surface.





Role of Smoothness

If E completely unconstrained, minimization is impossible.



All we could do is search through all possible values $\mathbf{w}$.

Key idea: If E is continuous, then measuring $E(\mathbf{w})$ gives information about E at many nearby values.

Role of Derivatives

- Another powerful tool that we have is the gradient

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_D})^T.$$

- Two ways to think of this:
  - Each $\frac{\partial E}{\partial w_k}$ says: If we wiggle w_k and keep everything else the same, does the error get better or worse?
  - The function

    $$f(\mathbf{w}) = E(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)^\top \nabla E|_{\mathbf{w}_0}$$

    is a linear function of $\mathbf{w}$ that approximates E well in a neighbourhood around $\mathbf{w}_0$. (Taylor's theorem)
- Gradient points in the direction of steepest error ascent in weight space.

Numerical Optimization Algorithms

- **Numerical optimization** algorithms try to solve the general problem

$$\min_{\mathbf{w}} E(\mathbf{w})$$

- Different types of optimization algorithms expect different inputs
  - Zero-th order: Requires only a procedure that computes $E(\mathbf{w})$. These are basically search algorithms.
  - First order: Also requires the gradient ∇E
  - Second order: Also requires the Hessian matrix $\nabla\nabla E$
  - High order: Uses higher order derivatives. Rarely useful.
  - Constrained optimization: Only a subset of $\mathbf{w}$ values are legal.
- Today we'll discuss first order, second order, and constrained optimization

Optimization Algorithm Cartoon

- Basically, numerical optimization algorithms are iterative. They generate a sequence of points

$$\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \ldots$$
$$E(\mathbf{w}_0), E(\mathbf{w}_1), E(\mathbf{w}_2), \ldots$$
$$\nabla E(\mathbf{w}_0), \nabla E(\mathbf{w}_1), \nabla E(\mathbf{w}_2), \ldots$$

- Basic optimization algorithm is

initialize $\mathbf{w}$
while $E(\mathbf{w})$ is unacceptably high
    calculate $\mathbf{g} = \nabla E$
    Compute direction $\mathbf{d}$ from $\mathbf{w}$, $E(\mathbf{w})$, $\mathbf{g}$
        (can use previous gradients as well...)
    $\mathbf{w} \leftarrow \mathbf{w} - \eta\, \mathbf{d}$
end while
return $\mathbf{w}$

Gradient Descent

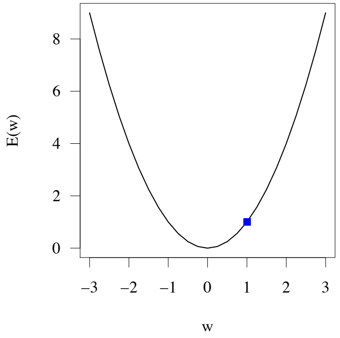
- Locally the direction of steepest descent is the gradient.
- Simple gradient descent algorithm:

initialize $\mathbf{w}$
while $E(\mathbf{w})$ is unacceptably high
    calculate $\mathbf{g} \leftarrow \frac{\partial E}{\partial \mathbf{w}}$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta\, \mathbf{g}$
end while
return $\mathbf{w}$

- η is known as the *step size* (sometimes called *learning rate*)
  - We must choose $\eta > 0$.
  - η too small $\rightarrow$ too slow
  - η too large $\rightarrow$ instability

Effect of Step Size

Goal: Minimize
$E(w) = w^2$

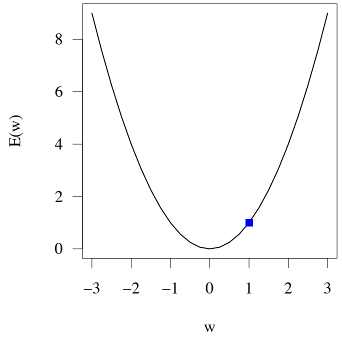


- Take $\eta = 0.1$. Works well.

$$
\begin{aligned}
w_0 &= 1.0 \\
w_1 &= \mathbf{w}_0 - 0.1 \cdot 2w_0 = 0.8 \\
w_2 &= \mathbf{w}_1 - 0.1 \cdot 2w_1 = 0.64 \\
w_3 &= \mathbf{w}_2 - 0.1 \cdot 2w_2 = 0.512 \\
&\cdots \\
w_{25} &= 0.0047
\end{aligned}
$$

Effect of Step Size

Goal: Minimize
$E(w) = w^2$



- Take $\eta = 1.1$. Not so good. If you step too far, you can leap over the region that contains the minimum

$$
\begin{aligned}
w_0 &= 1.0 \\
w_1 &= \mathbf{w}_0 - 1.1 \cdot 2w_0 = -1.2 \\
w_2 &= \mathbf{w}_1 - 1.1 \cdot 2w_1 = 1.44 \\
w_3 &= \mathbf{w}_2 - 1.1 \cdot 2w_2 = -1.72 \\
&\cdots \\
w_{25} &= 79.50
\end{aligned}
$$

- Finally, take $\eta = 0.000001$. What happens here?

Batch vs online

- So far all the objective functions we have seen look like:

$$E(\mathbf{w}; D) = \sum_{n=1}^{n} E^n(\mathbf{w}; y^n, \mathbf{x}^n).$$

$D = \{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots (\mathbf{x}^n, y^n)\}$ is the training set.
- Each term sum depends on only one training instance
- The gradient in this case is always

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial \mathbf{w}}$$

- The algorithm on slide 8 scans *all* the training instances before changing the parameters.
- Seems dumb if we have millions of training instances. Surely we can get a gradient that is "good enough" from fewer instances, e.g., a couple thousand? Or maybe even from just one?

Batch vs online

- **Batch** learning: use all patterns in training set, and update weights after calculating

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial \mathbf{w}}$$

- **On-line** learning: adapt weights after each pattern presentation, using $\frac{\partial E^n}{\partial \mathbf{w}}$
- **Batch** more powerful optimization methods
- **Batch** easier to analyze
- **On-line** more feasible for huge or continually growing datasets
- **On-line** may have ability to jump over local optima

Algorithms for Batch Gradient Descent

- Here is batch gradient descent.

  initialize $\mathbf{w}$
  while $E(\mathbf{w})$ is unacceptably high
  calculate $\mathbf{g} \leftarrow \sum_{n=1}^{N} \frac{\partial E^n}{\partial \mathbf{w}}$
  $\mathbf{w} \leftarrow \mathbf{w} - \eta\,\mathbf{g}$
  end while
  return $\mathbf{w}$

- This is just the algorithm we have seen before. We have just "substituted in" the fact that $E = \sum_{n=1}^{N} E^n$.

Algorithms for Online Gradient Descent

- Here is (a particular type of) online gradient descent algorithm

  initialize $\mathbf{w}$
  while $E(\mathbf{w})$ is unacceptably high
  Pick j as uniform random integer in $1 \ldots N$
  calculate $\mathbf{g} \leftarrow \frac{\partial E^j}{\partial \mathbf{w}}$
  $\mathbf{w} \leftarrow \mathbf{w} - \eta\,\mathbf{g}$
  end while
  return $\mathbf{w}$

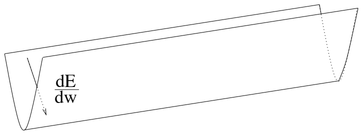
- This version is also called "stochastic gradient ascent" because we have picked the training instance randomly.
- There are other variants of online gradient descent.

Problems With Gradient Descent

- Setting the step size η
- Shallow valleys
- Highly curved error surfaces
- Local minima

Shallow Valleys

- Typical gradient descent can be fooled in several ways, which is why more sophisticated methods are used when possible. One problem:



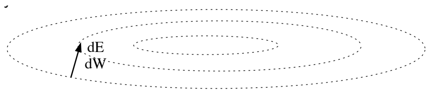
- Gradient descent goes very slowly once it hits the shallow valley.
- One hack to deal with this is *momentum*

$$\mathbf{d}_t = \beta \mathbf{d}_{t-1} + (1-\beta)\eta \nabla E(\mathbf{w}_t)$$

- Now you have to set both η and β. Can be difficult and irritating.

Curved Error Surfaces

- A second problem with gradient descent is that the gradient might not point towards the optimum. This is because of curvature



- Note: gradient is the *locally* steepest direction. Need not directly point toward local optimum.
- Local curvature is measured by the Hessian matrix: $H_{ij} = \partial^2 E / \partial w_i w_j$.
- By the way, do these ellipses remind you of anything?

Second Order Information

- Taylor expansion

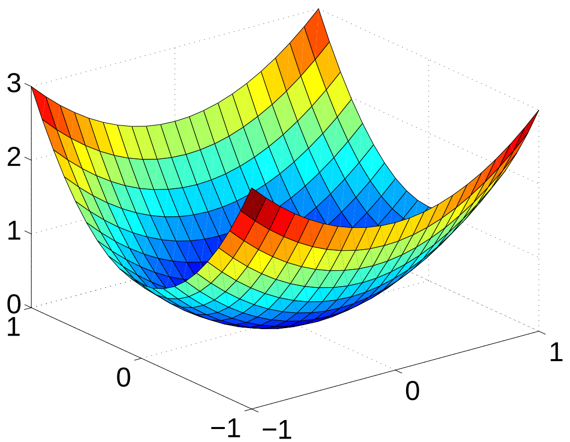
$$E(\mathbf{w} + \boldsymbol{\delta}) \simeq E(\mathbf{w}) + \boldsymbol{\delta}^T \nabla_{\mathbf{w}} E + \frac{1}{2} \boldsymbol{\delta}^T H \boldsymbol{\delta}$$

where

$$H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$$

- H is called the Hessian.
- If H is positive definite, this models the error surface as a quadratic bowl.

Quadratic Bowl



Direct Optimization

- A quadratic function

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

can be minimised directly using

$$\mathbf{w} = -H^{-1} \mathbf{b}$$

but this requires
  - Knowing/computing H, which has size $O(D^2)$ for a D-dimensional parameter space
  - Inverting H, $O(D^3)$

Newton's Method

- Use the second order Taylor expansion

$$E(\mathbf{w} + \boldsymbol{\delta}) \simeq E(\mathbf{w}) + \boldsymbol{\delta}^T \nabla_{\mathbf{w}} E + \frac{1}{2} \boldsymbol{\delta}^T H \boldsymbol{\delta}$$

- From the last slide, the minimum of the approximation is $\boldsymbol{\delta}^* = -H^{-1} \nabla_{\mathbf{w}} E$
- Use that as the direction in steepest descent
- This is called *Newton's method*.
- You may have heard of Newton's method for finding a root, i.e., a point x such that $f(x) = 0$. Similar thing, we are finding zeros of ∇f.
- Compare Newton step to gradient descent $\boldsymbol{\delta} = -\eta \nabla_{\mathbf{w}} E$

Advanced First Order Methods

- Newton's method is fast in that once you are close enough to a minimum.
- What we mean by this is that it needs very few iterations to get close to the optimum (You can actually prove this if you take an optimization course)
- If you have a not-too-large number of parameters and instances, this is probably method of choice.
- But for most ML problems, it is slow. Why? How many second derivatives are there?
- Instead we use "fancy" first-order methods that try to approximate second order information using only gradients.
- These are the state of the art *for batch methods*
  - One type: Quasi-Newton methods (I like one called *limited memory BFGS*).
  - Conjugate gradient
  - We won't discuss how these work, but you should know that they exist so that you can use them.

Constrained problems

- Constraints: e.g. $f(\mathbf{w}) < 0$.
- Example: Observe the points $\{0.5, 1.0\}$ from a Gaussian with known mean $\mu = 0.8$ and unknown standard deviation σ. Want to estimate σ by maximum likelihood.
- Constraint: σ must be positive.
- In this case to find the maximum likelihood solution, the optimization problem is

$$\max_{\sigma} \sum_{i=1}^{2} [-\frac{1}{2\sigma^2}(x^i - \mu)^2 - \frac{1}{2}\log(2\pi\sigma^2)]$$

subject to $\sigma > 0$

- In this case: solution can be done analytically. More complex cases require a numerical method for constrained optimization.

Constrained problems

Either remove constraints by re-parameterization. E.g. $\mathbf{w} > 0$. Set $\phi = \log(\mathbf{w})$. Now ϕ unconstrained.

Or use a constrained optimization method, e.g. for linear programming, quadratic programming.

Linear Programming

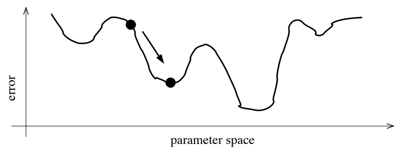
- Find optimum, within a (potentially unbounded) polytope, of a linear function
- Polytope = polygon or higher dimensional generalization thereof.
- Easy: maximum (if it exists) must be at vertex of polytope (or on a convex set containing such a vertex). Hill climb on vertices using an adjacency walk (Simplex algorithm)

Quadratic Programming

- Find optimum, within a (potentially unbounded) polytope, of a quadratic form
- Interior point methods, Active set methods.
- Second order methods for convex quadratic functions Newton-Raphson, Conjugate Gradient variants.
- A number of machine learning methods are cast as quadratic programming problems (e.g. Support Vector Machines).

Non-convexity and local minima

- If you follow the gradient, where will you end up? Once you hit a local minimum, gradient is 0, so you stop.



- Certain nice functions, such as the likelihood for linear and logistic regression are *convex*, meaning that the second derivative is always positive. This implies that any local minimum is global.

- Dealing with local minima: Train multiple models from different starting places, and then choose best (or combine in some way).
- No guarantees. Unrealistic to believe this will find global mimimum.
- Local minima occur, e.g. for neural networks
- Bayesian interpretation, where $E(\mathbf{w}) = -\log p(\mathbf{w}|D)$
- Finding local minima of $E(\mathbf{w})$ as a way of approximating integration over the posterior by finding local maxima of $p(\mathbf{w}|D)$

Convex Functions

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for $\alpha \in [0,1]$

$$f(\alpha \mathbf{x} + (1-\alpha)\mathbf{y}) \le \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y})$$

  Essentially "bowl shaped"
- Examples:

$$f(x) = x^2 \qquad f(x) = -\log x \qquad f(\mathbf{x}) = \log\left(\sum_d \exp\{x_d\}\right)$$

- If f differentiable, this implies

$$f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^\top \nabla f|_{\mathbf{x}_0} \le f(\mathbf{x})$$

  for all $\mathbf{x}$ and $\mathbf{x}_0$. (To see this: take limit of above as $\mathbf{x} \to \mathbf{y}$.)
- This implies that any local minimum is a global one!

Convex Optimization Problems

- A convex optimization problem is one that can be written as

$$\begin{aligned} &\min \ f_0(\mathbf{x}) \\ \text{subject to } & f_i(\mathbf{x}) \le 0 \qquad i \in \{1 \dots N\} \end{aligned}$$

  for some choice of functions $f_0 \dots f_N$ where each f_i is convex
- Optimise convex function over a convex set...
- Unconstrained problems: Use methods from before. You'll find a global optimum!
- Convexity means any local optimum is also global optimum.
- Constrained convex problems: Interior point methods, Active set methods.
- Most convex optimization problems can be solved efficiently in practice.
- (How high a scale you can reach depends on the type of problem you have)

Optimization: Summary

- Complex mathematical area. Do not implement your own optimization algorithms if you can help it!
- My advice: For unconstrained problems
  - Batch is less hassle than online. But if you have big data, you must use online. Batch is too slow
  - (For neural networks, typically online methods are method of choice.)
  - If online, you use gradient descent. Forget about second order stuff.
  - If batch, use one of the fancy first-order methods (quasi-Newton or conjugate gradients). DO NOT implement either of these yourself!
- If you have a constrained problem
  - Linear programs are easy. Use off the shelf tools.
  - More than that: Try to convert into unconstrained problem.
- Convex problems: Global minimum. Non-convex: *Local* optima.

What you should take away

- Complex mathematical area. Do not implement your own optimization algorithms if you can help it!
- Stuff you should understand:
  - How and why we convert learning problems into optimization problems
  - *Modularity* between modelling and optimization
  - Gradient descent
  - Why gradient descent can run into problems
  - Especially local minima
- Methods of choice: Fancy first-order methods (e.g., quasi-Newton, CG) for moderate amounts of data. Stochastic gradient for large amounts of data.