Role of Smoothness

If $E$ completely unconstrained, minimization is impossible.

All we could do is search through all possible values $w$.

Key idea: If $E$ is continuous, then measuring $E(w)$ gives information about $E$ at many nearby values.
Role of Derivatives

- Another powerful tool that we have is the gradient
  \[ \nabla E = \left( \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_D} \right)^T. \]

- 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 \( f(w) = E(w_0) + (w - w_0)^T \nabla E|_{w_0} \)
    is a linear function of \( w \) that approximates \( E \) well in a neighbourough around \( w_0 \). (Taylor's theorem)

- Gradient points in the direction of steepest error ascent in weight space.

Optimization Algorithm Cartoon

- Basically, numerical optimization algorithms are iterative. They generate a sequence of points
  \[ w_0, w_1, w_2, \ldots \]
  \[ E(w_0), E(w_1), E(w_2), \ldots \]
  \[ \nabla E(w_0), \nabla E(w_1), \nabla E(w_2), \ldots \]

- Basic optimization algorithm is

```
initial w
while E(w) is unacceptably high
    calculate g = \nabla E
    Compute direction d from w, E(w), g
    (can use previous gradients as well...)
    w ← w − \eta d
end while
return w
```

Numerical Optimization Algorithms

- **Numerical optimization** algorithms try to solve the general problem
  \[ \min_w E(w) \]

- Different types of optimization algorithms expect different inputs:
  - Zero-th order: Requires only a procedure that computes \( E(w) \).
  - These are basically search algorithms.
  - First order: Also requires the gradient \( \nabla E \)
  - Second order: Also requires the Hessian matrix \( \nabla \nabla E \)
  - High order: Uses higher order derivatives. Rarely useful.
  - Constrained optimization: Only a subset of \( w \) values are legal.

- Today we’ll discuss first order, second order, and constrained optimization.

Gradient Descent

- Locally the direction of steepest descent is the gradient.

- Simple gradient descent algorithm:

```
initialize w
while E(w) is unacceptably high
    calculate g ← \frac{\partial E}{\partial w}
    w ← w − \eta g
end while
return w
```

- \( \eta \) is known as the step size (sometimes called learning rate)
  - We must choose \( \eta > 0 \).
  - \( \eta \) too small → too slow
  - \( \eta \) too large → instability
Effect of Step Size

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

- Take \( \eta = 0.1 \). Works well.
  \[
  \begin{align*}
  w_0 &= 1.0 \\
  w_1 &= w_0 - 0.1 \cdot 2w_0 = 0.8 \\
  w_2 &= w_1 - 0.1 \cdot 2w_1 = 0.64 \\
  w_3 &= w_2 - 0.1 \cdot 2w_2 = 0.512 \\
  \vdots \\
  w_{25} &= 0.0047
  \end{align*}
  \]

Batch vs online

- So far all the objective functions we have seen look like:
  \[
  E(w; D) = \sum_{n=1}^{N} E^n(w; y^n, x^n).
  \]
- \( D = \{(x^1, y^1), (x^2, y^2), \ldots (x^n, y^n)\} \) is the training set.
- Each term sum depends on only one training instance
- The gradient in this case is always
  \[
  \frac{\partial E}{\partial w} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial w}
  \]
- 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
  \[
  \frac{\partial E}{\partial w} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial w}
  \]
- **On-line** learning: adapt weights after each pattern presentation, using
  \[
  \frac{\partial E^n}{\partial w}
  \]
- **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 w
  while $E(w)$ is unacceptably high
    calculate $g \leftarrow \sum_{n=1}^{N} \frac{\partial E^n}{\partial w}$
    $w \leftarrow w - \eta g$
  end while
  return w

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

Problems With Gradient Descent

- Setting the step size $\eta$
- Shallow valleys
- Highly curved error surfaces
- Local minima

Algorithms for Online Gradient Descent

- Here is (a particular type of) online gradient descent algorithm
  initialize w
  while $E(w)$ is unacceptably high
    Pick $j$ as uniform random integer in $1 \ldots N$
    calculate $g \leftarrow \frac{\partial E^j}{\partial w}$
    $w \leftarrow w - \eta g$
  end while
  return 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.

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
  $d_t = \beta d_{t-1} + (1 - \beta) \eta \nabla E(w_t)$

- Now you have to set both $\eta$ and $\beta$. 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 towards the optimum.

- Local curvature is measured by the Hessian matrix: 
  \[ H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \]

- By the way, do these ellipses remind you of anything?

Quadratic Bowl

Second Order Information

- Taylor expansion
  \[ E(w + \delta) \simeq E(w) + \delta^T \nabla_w E + \frac{1}{2} \delta^T H \delta \]

  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.

Direct Optimization

- A quadratic function
  \[ E(w) = \frac{1}{2} w^T H w + b^T w \]

  can be minimised directly using
  \[ w = -H^{-1} b \]

  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
  \[ E(w + \delta) \simeq E(w) + \delta^T \nabla_w E + \frac{1}{2} \delta^T H \delta \]
- From the last slide, the minimum of the approximation is
  \[ \delta^* = -H^{-1} \nabla_w E \]
- 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 \( \nabla f \).

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. \( f(w) < 0 \).
- Example: Observe the points \( \{0.5, 1.0\} \) from a Gaussian with known mean \( \mu = 0.8 \) and unknown standard deviation \( \sigma \). Want to estimate \( \sigma \) by maximum likelihood.
- Constraint: \( \sigma \) must be positive.
- In this case to find the maximum likelihood solution, the optimization problem is
  \[
  \max_{\sigma} \sum_{i=1}^{2} \left[ -\frac{1}{2\sigma^2}(x^i - \mu)^2 - \frac{1}{2} \log(2\pi\sigma^2) \right]
  \]
  subject to \( \sigma > 0 \)
- In this case: solution can be done analytically. More complex cases require a numerical method for constrained optimization.

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

Or use a constrained optimization method, e.g. for linear programming, quadratic programming.
Conjugate Gradients
• Observation: at the end of a line search, the new gradient is (almost) orthogonal to the direction we just searched in.
• The other way is to explicitly incorporate the constraints into our cost function.

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: e.g. Newton-Raphson, Conjugate Gradient variants.
• A number of machine learning methods are cast as quadratic programming problems (e.g. Support Vector Machines).

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).

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 minimum.
• Local minima occur, e.g. for neural networks
• Bayesian interpretation, where $E(w) = -\log p(w|D)$
• Finding local minima of $E(w)$ as a way of approximating integration over the posterior by finding local maxima of $p(w|D)$
Convex Functions

- A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if for $\alpha \in [0, 1]$
  $$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

Essentially “bowl shaped”

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

- If $f$ differentiable, this implies
  $f(x_0) + (x - x_0)^\top \nabla f|_{x_0} \leq f(x)$

for all $x$ and $x_0$. (To see this: take limit of above as $x \rightarrow y$.)

- This implies that any local minimum is a global one!

Convex Optimization Problems

- A convex optimization problem is one that can be written as
  $\min f_0(x)$
  subject to $f_i(x) \leq 0 \quad i \in \{1 \ldots N\}$

for some choice of functions $f_0 \ldots f_N$ 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.


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.