IAML: Optimization

Nigel Goddard
School of Informatics

Semester 1

Outline

- Why we use optimization in machine learning
- The general optimization problem
- Gradient descent
- Problems with gradient descent
- Batch versus online
- Second-order methods
- Constrained optimization

Many illustrations, text, and general ideas from these slides are taken from Sam Roweis (1972-2010).

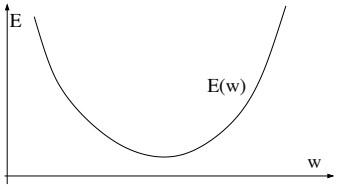
Why Optimization

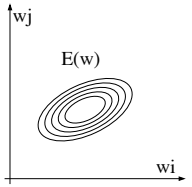
- A main idea in machine learning is to convert the learning problem into a continuous optimization problem.
- Examples: Linear regression, logistic regression (we have seen), neural networks, SVMs (we will see these later)
- One way to do this is *maximum likelihood*

$$\begin{aligned}\ell(\mathbf{w}) &= \log p(y_1, \mathbf{x}_1, y_2, \mathbf{x}_2, \ldots, y_n, \mathbf{x}_n|\mathbf{w}) \\ &= \log \prod_{i=1}^{n} p(y_i, \mathbf{x}_i|\mathbf{w}) \\ &= \sum_{i=1}^{n} \log p(y_i, \mathbf{x}_i|\mathbf{w})\end{aligned}$$

- Example: Linear regression

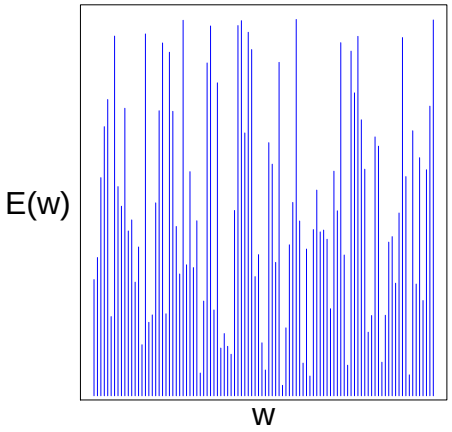
- End result: an "error function" $E(\mathbf{w})$ which we want to minimize.
- e.g., $E(\mathbf{w})$ can be the negative of the log likelihood.
- Consider a fixed training set; think in weight (not input) space. At each setting of the weights there is some error (given the fixed training set): this defines an error surface in weight space.
- Learning == descending the error surface.
- If the data are IID, the error function E is a sum of error function E_i for each data point





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

- If we wiggle w_k and keep everything else the same, does the error get better or worse?
- Calculus has an answer to exactly this question: $\frac{\partial E}{\partial w_k}$
- So: use a differentiable cost function E and compute partial derivatives of each parameter
- The vector of partial derivatives is called the gradient of the error. It is written $\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n})$. Alternate notation $\frac{\partial E}{\partial \mathbf{w}}$.
- It points in the direction of steepest error descent in weight space.
- Three crucial questions:
  - How do we compute the gradient ∇E efficiently?
  - Once we have the gradient, how do we minimize the error?
  - Where will we end up in weight space?

Numerical Optimization Algorithms

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

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

- Most commonly, a numerical optimization procedure takes two inputs:
  - A procedure that computes $E(\mathbf{w})$
  - A procedure that computes the partial derivative $\frac{\partial E}{\partial w_j}$
- (Aside: Some use less information, i.e., they don't use gradients. Some use more information, i.e., higher order derivative. We won't go into these algorithms in the course.)

Optimization Algorithm Cartoon

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

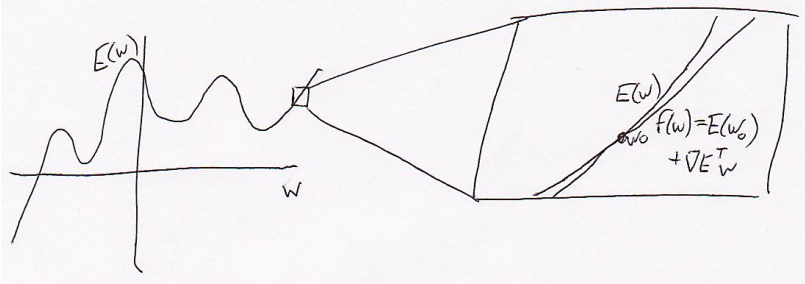
$$\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \ldots$$
$$E(\mathbf{w}_0), E(\mathbf{w}_1), E(\mathbf{w}_2), \ldots$$
$$\nabla E(\mathbf{w}_0), \nabla E(\mathbf{w}_1), \nabla E(\mathbf{w}_2), \ldots$$

- 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 $\mathbf{w}$

A Choice of Direction

- The simplest choice **d** is the current gradient ∇E.
- It is locally the steepest descent direction.
- (Technically, the reason for this choice is Taylor's theorem from calculus.)



Gradient Descent

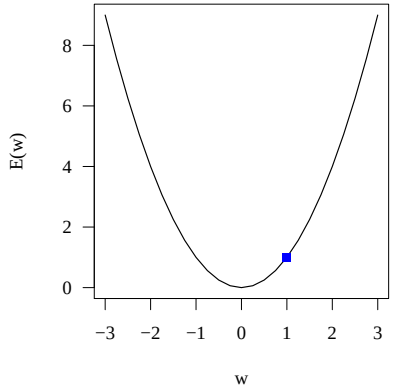
- Simple gradient descent algorithm:

initialize **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 **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
$E(w) = w^2$

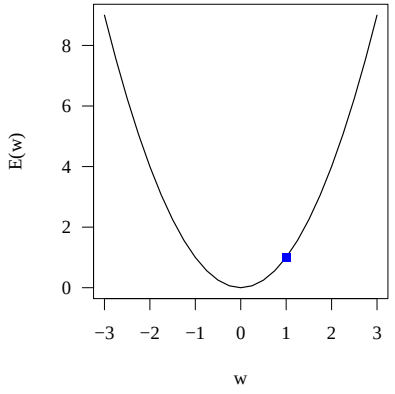


- Take $\eta = 0.1$. Works well.

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

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

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

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

"Bold Driver" Gradient Descent

- Simple heuristic for choosing η which you can use if you're desperate.

  initialize $\mathbf{w}$, η
  initialize $e \leftarrow E(\mathbf{w})$; $\mathbf{g} \leftarrow \nabla E(\mathbf{w})$ while $\eta > 0$
  $\quad \mathbf{w}_1 \leftarrow \mathbf{w} - \eta \mathbf{g}$
  $\quad e_1 = E(\mathbf{w}_1)$; $\mathbf{g}_1 = \nabla E$
  $\quad$ **if** $e_1 \geq e$
  $\quad\quad \eta = \eta/2$
  $\quad$ **else**
  $\quad\quad \eta = 1.01\eta$; $\mathbf{w} \leftarrow \mathbf{w}_1$; $\mathbf{g} \leftarrow \mathbf{g}_1$; $e = e_1$
  end while
  return $\mathbf{w}$

- Finds a *local* minimum of E.

Batch vs online

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

$$E(\mathbf{w}; D) = \sum_{i=1}^{n} E_i(\mathbf{w}; y_i, \mathbf{x}_i).$$

  $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_n, y_n)\}$ is the training set.
- Each term sum depends on only one training instance
- Example: Logistic regression: $E_i(\mathbf{w}; y_i, \mathbf{x}_i) = \log p(y_i | \mathbf{x}_i, \mathbf{w})$.
- The gradient in this case is always

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

- The algorithm on slide 10 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 of thousand? Or maybe even from just one?

Batch vs online

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

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{i} \frac{\partial E_i}{\partial \mathbf{w}}$$

- **On-line** learning: adapt weights after each pattern presentation, using $\frac{\partial E_i}{\partial \mathbf{w}}$
- **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
  $\quad$ calculate $\mathbf{g} \leftarrow \sum_{i=1}^{N} \frac{\partial E_i}{\partial \mathbf{w}}$
  $\quad \mathbf{w} \leftarrow \mathbf{w} - \eta\, \mathbf{g}$
  end while
  return $\mathbf{w}$

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

Algorithms for Online Gradient Descent

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

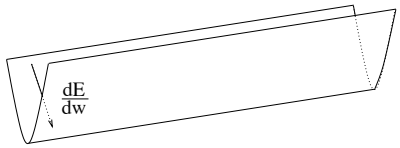
  initialize **w**
  while $E(\mathbf{w})$ is unacceptably high
  Pick j as uniform random integer in $1 \ldots N$
  calculate $\mathbf{g} \leftarrow \frac{\partial E_j}{\partial \mathbf{w}}$
  $\mathbf{w} \leftarrow \mathbf{w} - \eta\, \mathbf{g}$
  end while
  return **w**
- This version is also called "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:



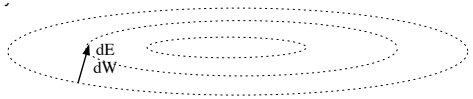
- Gradient descent goes very slowly once it hits the shallow valley.
- One hack to deal with this is *momentum*

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

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

Curved Error Surfaces

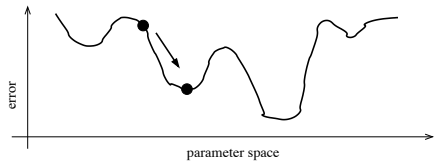
- A second problem with gradient descent is that the gradient might not point towards the optimum. This is because of curvature



- Note: gradient is the *locally* steepest direction. Need not directly point toward local optimum.
- Local curvature is measured by the Hessian matrix: $H_{ij} = \partial^2 E / \partial w_i w_j$.

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 squared error, logistic regression likelihood are *convex*, meaning that the second derivative is always positive. This implies that any local minimum is global.
- There is no great solution to this problem. It is a fundamental one. Usually, the best you can do is rerun the optimizer multiple times from different random starting points.

Advanced Topics That We Will Not Cover (Part I)

- Some of these issues (shallow valley, curved error surfaces) can be fixed
  - Some of these are *second-order* methods like Newton's method that use the second derivatives
  - Also there are fancy first-order methods like quasi-Newton methods (e.g., *limited memory BFGS*) and conjugate gradient
  - They are the state of the art methods for logistic regression (as long as there are not too many data points)
  - We will not discuss these methods in the course.
- Other issues (like local minima) cannot be easily fixed

Advanced Topics That We Will Not Cover (Part II)

- Sometimes the optimization problem has *constraints*
  - 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

$$\max_{\mu,\sigma} \sum_{i=1}^{2} \frac{1}{2\sigma^2}(x_i - \mu)^2$$

subject to $\sigma > 0$

  - There are ways to solve this (in this case: can be done analytically). We will not discuss them in this course.

Summary

- 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.