Module 5 - Optimization and Geometry

- 5.1 Unconstrained Optimization I
- 5.2 Unconstrained Optimization II
- 5.3 Unconstrained Optimization III
- 5.4 Convexity I
- 5.5 Linear algebra IV: Positive Semidefinite Matrices
- 5.6 Convexity II

总结:

5.1 Unconstrained Optimization I

Topics we'll cover

1. Optimization by local search
2. The problem of multiple local optima
3. Gradient descent
4. Taking the derivative of a function of many variables

Over the past few lectures, we have been using optimization to solve prediction problems. What we do, basically, is to define a suitable loss function, and then learn our parameters by minimizing that function. That's what we did for ***least squares regression***, and for ***logistic regression***, and we'll be doing a whole lot more of that in the upcoming weeks. Indeed, optimization lies at the heart of modern machine learning. Now, sometimes, the problem that you need to solve falls neatly into some predefined category, and then you're in luck. You can just use existing software for logistic regression, or lasso, or support vector machines. But, other times, you find that in order to get a really good solution you have to come up with a customized loss function, something that's really tailored made to the domain. What do you do in those cases? In those cases, you can no longer fall back upon existing code.

Well, it turns out that there are some relatively straightforward ways of coming up with minimization algorithms for essentially any loss function, and today we'll look at one of those methods called **gradient descent**. Becoming familiar with this is really a source of power, because it liberates you from having to use canned solutions. So today, we'll start by talking about what it means to solve an optimization problem using local search, and about the problem of having multiple local optima. We'll then look at the algorithm, gradient descent. All that's needed in order to use it is to be able to compute the derivative of the loss function, and we'll talk a little bit about what that means.

Minimizing a loss function

Usual setup in machine learning: choose a model w by minimizing a loss function $L(w)$ that depends on the data.

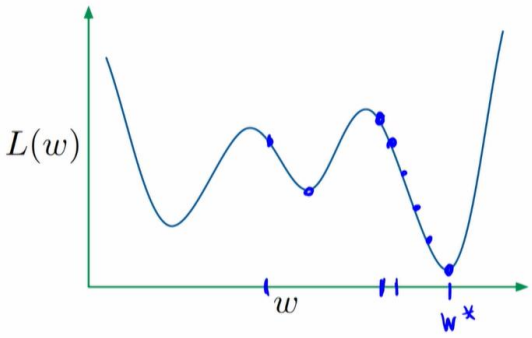
- Linear regression: $L(w) = \sum_i (y^{(i)} - (w \cdot x^{(i)}))^2$
- Logistic regression: $L(w) = \sum_i \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)})})$

Default way to solve this minimization: **local search**.

So **for much of machine learning, the way we learn is by defining a suitable loss function, and then minimizing it**. So for example, these are the loss functions for least squares regression and for logistic regression. How do we minimize loss functions like this? How do we find the minimizing parameters, W? And for the most part, we do this by local search. So we start with some guess for W, and then we tweak it a little bit to make it better, to reduce the loss, and then we tweak is some more, and then we keep going until this whole process converges.

总结：

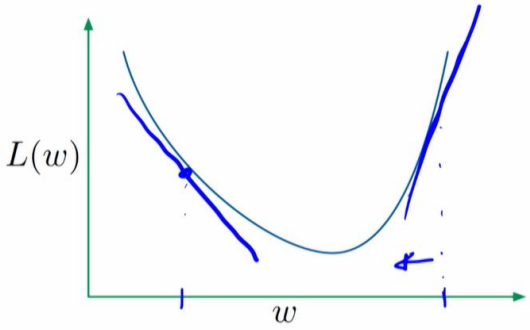
Local search



- Initialize w arbitrarily
- Repeat until w converges:
  - Find some w' close to w with $L(w') < L(w)$.
  - Move w to w'.

A good situation for local search

When the loss function is **convex**:



Idea for picking search direction:
Look at the **derivative** of $L(w)$ at the current point w.

Pictorially, this is what it looks like. So along the horizontal access I have the parameters, W, and along the vertical axis is the loss function, L of W. And the thing we're looking for is the minimizer of this loss function. So this is the point we want, let's call that W star. What does local search do? Well, local search starts with any old solution, so let's say it starts over here, at this loss value. And then it tweaks it to make it a little bit better. So maybe jumps a little bit like that, to this point. And then it tweaks it some more, and so on, and it keeps going, until it finally gets to a place where there's no longer any little tweak that's going to help. And then it stops. And in this case, it ends up finding the correct answer.Now, a lot hinges upon where you start. So if your initial guess at the parameter, for example, is over here, so some point like this, then, local search might simply wind you up over here, which is a local optimum, but it's not the right solution. It could be something that's much worse than W star. So the problem is that this particular loss function has a lot of local optima. There are all these different places where a local search algorithm can get trapped. Unfortunately, this kind of loss function arises all the time in machine learning, and by and large we are resigned to this. But we really love the situations where the loss function is a little bit better behaved, and we actively seek them out.

So what kind of loss functions do we want? Well, we want loss functions like this. Loss functions that are convex. So, very soon we'll be defining exactly what this means, but for the time being, the way you can think of it is that the loss function looks like a bowl, and we just want to get to the bottom of the bowl. So there's only one local optimum, and that's also the global optimum. And as a result, functions like this are pretty easy to minimize. So, how exactly do we do this, though? Suppose we're at a particular value of W, how do we know which way to move? So, this picture, for example, has a one-dimensional W. How do we know whether to go left or right? One easy way to do this is by looking at the slope, or derivative of the function. So let's see what exactly that means. Let's say, for example, that my current guess of W is over there. So let's look at the slope over there. This is the slope, and the slope is positive. Now, what that means is if you increase W, the function goes up, and if you decrease W, the function goes down. What do we want to do? Do we want the function to go up or down? Well, we're trying to minimize the function, so we want it to go down, so that means we have to move to the left. So the slope tells us that we have to move to the left. The slope is positive, so we want to go the other way, to the left. Now, suppose our parameter estimate was somewhere there. So the slope over there looks like this, and in that case, the slope is negative. So, if we want to decrease the function, we have to move to the right. Again, the slope is negative, so we increase W, we move in the opposite direction to the slope. And this is a rule that can be used in general. The way you can adjust the current parameters, W, is to simply move opposite to the gradient.

总结:

Gradient descent

For minimizing a function $L(w)$:

- $w_o = 0, t = 0$
- while $\nabla L(w_t) \not\approx 0$:
  - $w_{t+1} = w_t - \eta_t \nabla L(w_t)$
  - $t = t + 1$

Here η_t is the *step size* at time t.

And that gives us the gradient descent algorithm. So, let's say there's this function we want to minimize, L of W, and W might say consist of D parameters. So W might be a D-dimensional vector. We start by initializing W to anything, so our initial setting for W might, for example, just be zero. And then we keep tweaking W. At time T, our current setting for W, our current guess at W is W sub T, and then we tweak it a little bit, and get W sub T plus one. So what is exactly is this tweak? Well, what we do is we compute the derivative of the loss function at W sub T, the derivative, and then we want to move in the opposite direction, so we take the negative of that. That tells us the direction in which we want to move, but we also have to decide how far we go in that direction. And that is the step size, eta sub T. Now that thing that looks like an N is actually the Greek letter eta, and that's the step size. For now you can think of it as a small constant if you like, we'll talk about it in more detail later on. And that tells us how far we go in that direction. So we have our current estimate, W sub T, we tweak it a little bit, we get W sub T plus one, and we keep going in this way until we finally converge, and that's it, that's the answer. An extremely simple algorithm, it's just four lines of code, and it can be used to solve a huge variety of different problems. So what do we need to do in order to use this algorithm? Well, there's really just one thing we need to do, and that is to compute this derivative over here. The derivative of the loss function.

Multivariate differentiation

$y = 3x^2 \quad \frac{dy}{dx} = 6x$

Example: $w \in \mathbb{R}^3$ and $F(w) = 3w_1w_2 + w_3$.

$F(w_1, w_2, w_3)$

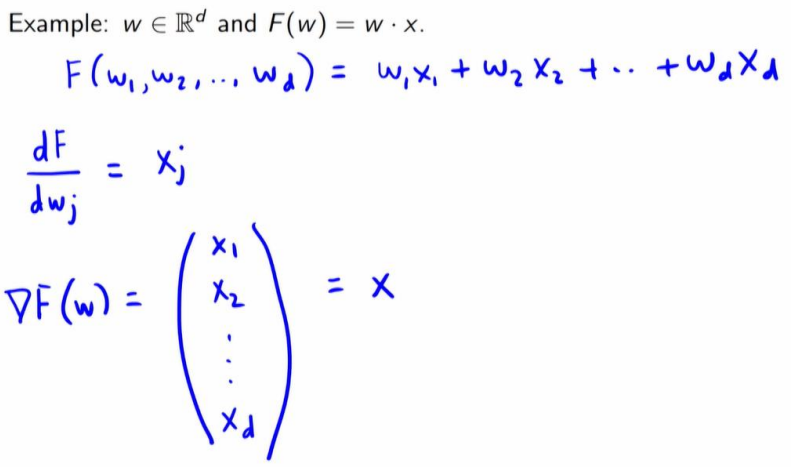
$\frac{dF}{dw_1} = 3w_2, \quad \frac{dF}{dw_2} = 3w_1, \quad \frac{dF}{dw_3} = 1$

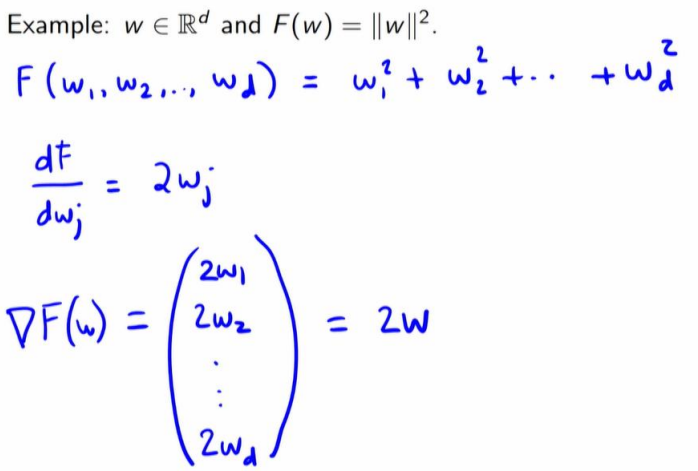
$$\nabla F(w) = \begin{pmatrix} \frac{dF}{dw_1} \\ \frac{dF}{dw_2} \\ \frac{dF}{dw_3} \end{pmatrix} = \begin{pmatrix} 3w_2 \\ 3w_1 \\ 1 \end{pmatrix}$$

Let's see exactly what that entails. So one thing that I think we're all pretty familiar with is finding the derivative of a function of one variable. So if you have, for example, Y equals three X squared, what is the derivative of Y, what is D Y D X? It's just six X. So we know how to do that. But what happens when you have a function of multiple variables? So there's a function over here, which is a function of three variables. It's a function of three variables, W one, W two, and W three. What does it mean to do the derivative in that case? Well, now you can take the derivative with respect to three different things, and the first step is to take these three derivatives separately. So let's start with the derivative with respect to W one. When we're computing this derivative, what we do is we treat the other variables, W two and W three, essentially as constants. So what do we get? What is the derivative of this expression over here with respect to W one? It's just three W two. Now, let's take the derivative with respect to W two. Okay, what do we get, three W one. And finally, let's take the derivative with respect to W three. Now there's only one term that involves W three, the last term, the derivative is one. Okay, so those are the three separate derivatives.

The overall derivative of F, which we denote like this,is obtained by just stacking together these three derivatives with respect to a single variable. So it's a vector, where we start by taking the derivative with respect to the first variable, then the derivative with respect to the second variable, and finally the derivative with respect to the third variable. So it's the vector three W two, three W one, and one. It's a three-dimensional vector, that's the derivative of F at any value W.

总结:





Let's do another example. So this time, we have a function of D variables. So we have a function of D variables, W one, W two, all the way to W D. And what is the function?It's just the dot product of W and X. So it's W one X one, plus W two X two, all the way to W D X D. So now we compute derivatives with respect to each of these D variables separately. So what is the derivative with respect to W J? Well, in that big summation there's only one term that involves W J, W J X J. So the derivative is just X J. Now let's stack these together in a vector. So the derivative of F at W is the derivative with respect to W one, which is X one, the derivative with respect to W two, which is X two, all the way to the derivative with respect to W D, which is X D. And what is this vector? It's just X. So the derivative of this function is just the D-dimensional vector, X.

Okay, one more. Again, we have a function of D variables. Let's go ahead and write in out in full. So we have these D variables, W one, W two, all the way to W D, and what is the function? It's the squared norm of W, so it's W one squared plus W two squared, all the way to W D squared. And now again, we have to compute the derivative with respect to each of the D things individually. So what is the derivative with respect to W J? Well, W J is in only one of those terms and that's W J squared, so the derivative is two W J. And putting these together in a vector, we find that the derivative at W is two W one, two W two, all the way to two W D. In other words, it's two W. Again, a D-dimensional vector. So in general, if you have a function of D variables, then its derivative is a D-dimensional vector.

Gradient descent

For minimizing a function $L(w)$:

- $w_o = 0, t = 0$
- while $\nabla L(w_t) \not\approx 0$:
  - $w_{t+1} = w_t - \eta_t \nabla L(w_t)$
  - $t = t + 1$

Here η_t is the *step size* at time t.

So now let's go back to the gradient descent algorithm. And let's say that the function we're minimizing is a function of D variables. Let's go ahead and look at this update rule. So what is this derivative now? Well, we've seen that that derivative is a D-dimensional vector, so this update will make sense. W is D-dimensional, and we're subtracting off a D-dimensional vector, so it's all sensible, at least in terms of the dimensions being aligned. How exactly does this rule work? Why does it reduce the loss to move in this direction? Well that's something we'll see next time. So, that's it for now, and we've got a little bit of a glimpse of gradient descent. And next time, we'll look at this algorithm in a little bit more detail. See you then.

POLL

Using gradient descent, if the derivative of a function at the value w_0 equals 0, what adjustment must be made to find the function's minimum?

A random w should be chosen, for example w = 0	9%
A larger value of w must be chosen, i.e. w > w_0	5%
A smaller value of w must be chosen, i.e. w < w_0	4%
✓ w_0 is the value that results in a local minimum, no adjustment necessary	82%

总结：

5.2 Unconstrained Optimization II

Topics we'll cover

1. Why does gradient descent work?
2. Setting the step size
3. Gradient descent for logistic regression

Gradient descent

For minimizing a function $L(w)$, over $w \in \mathbb{R}^d$:

- $w_o = 0, t = 0$
- while $\nabla L(w_t) \not\approx 0$:
  - $w_{t+1} = w_t - \eta_t \nabla L(w_t)$
  - $t = t + 1$

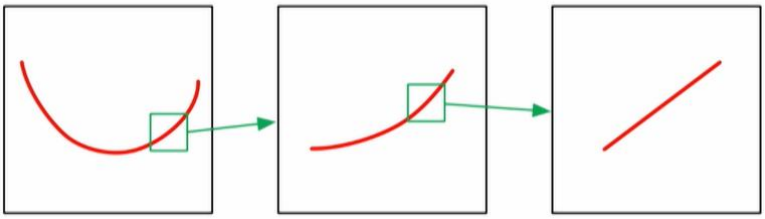
Here η_t is the *step size* at time t.

Last time, we introduced gradient descent,a general purpose algorithm for minimizing a loss function. Today, we will continue that discussion. So we will by talking about why exactly gradient descent works, and also about the issue of how to set the step size. Then, we will see how easy it is to derive a gradient descent algorithm by looking at logistic regression as an illustrative example.

So this is the gradient descent algorithm.There's some loss function, l, that we want to minimize, and we wanna find the parameter vector, w, that optimizes this function. The way we do it is by starting w at zero and then entering into this loop where we keep adjusting w, we keep tweaking it a little bit, until it finally converges. So, what are these updates? Well, if the current parameter vector is w sub t, what we do is we compute the gradient. We compute the derivative of the loss function, at w sub t, and we move in the opposite direction to that derivative with some step size a to sub t. It's a very simple update. And, the question is, why does this work? So, let's think about this a little bit.

Gradient descent: rationale

"Differentiable" $\Longrightarrow$ "locally linear".



For *small* displacements $u \in \mathbb{R}^d$,

$$L(w+u) \approx L(w) + u \cdot \nabla L(w)$$

Therefore, if $u = -\eta \nabla L(w)$ is small,

$$L(w+u) \approx L(w) - \eta \|\nabla L(w)\|^2 < L(w)$$

$L(w+u) = L(w) - \eta \nabla L(w) \cdot \nabla L(w)$

Now, gradient descent uses thederivative of the loss function. If the derivative exists, it means that the loss function is locally linear. What's that? Well here's a picture of loss function. Is it linear? No, it's not. But let's look at a little window, let's look at a little neighborhood. So, let's pick a point on the function and look at a little window around that point. Let's zoom in a little bit. Here's what it looks like. Is it linear now? No, it's not linear, but it's more linear than before. Let's zoom in a little bit more. We pick a point, zoom in, and now it's linear. If one looks at the function close up, if one looks at a small enough neighborhood, then the function is approximately linear in that neighborhood. And the linear approximation is given by the derivative of the function. So let's look at this more concretely. So we have some loss function, l (L), and we're gonna look at it in just a very small neighborhood.

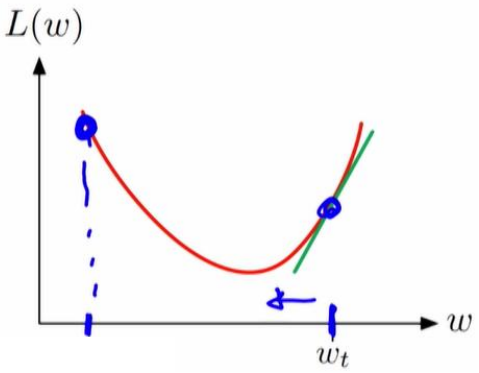
A small neighborhood around the point w. That means that the kind of points that we're looking at are points very close to W, points of the form w plus u, where u is some minuscule displacement. Now, because this neighborhood is so small, the function is approximately linear, and in fact, this is what it looks like. The value of l at w plus u is given by the value of w plus u dot product with the derivative of l and w. This is linear, and the derivative gives us this linear approximation. Let's see what this means for gradient descent. In gradient descent, l is this functionthat we are trying to minimize, and w is our current parameter vector. We're gonna tweak w a little bit, we're gonna move it from w to w plus u, and the specific u we pick in gradient descent is this value. It's the negative of the derivative times some step size ada. Okay, so let's plug in that value, u, into our linear approximation for the loss function. What do we get? So we get l of w plus u, so just looking at this linear approximation over here, l of w. Now u is negative ada times the derivative dot product with the same derivative. Now when you take a dot product of a vector with itself, you get the vector squared. And so, this is the value, approximately, of the loss function at w plus u. And what is it? It's the loss function at w, okay, minus something positive. So it's less than l of w. So moving in the direction opposite to the derivative is guaranteed to reduce the loss function. This is why gradient descent works.

总结：

The step size matters

Update rule: $w_{t+1} = w_t - \eta_t \nabla L(w_t)$

- Step size η_t too small: not much progress
- Too large: overshoot the mark



Some choices:

- Set η_t according to a fixed schedule, like $1/t$
- Choose by line search to minimize $L(w_{t+1})$

Okay, so we also have to talk about the step size, however. How should we set this ada (eta: η)? Well, if we think back to our discussion, if we think back to our mathematical justification, that linear approximation only holds in very small neighborhoods.

So, that suggests setting the step size adato something tiny. But if we did that, what it would mean, is that w would only change a tiny bit on each iteration and progress would be very slow. So that's not really something we'd like to do. We'd like to set ada larger. But if we set it too large, then we might overshoot the mark, and end up at a worse place than where we began. Okay, so let's look at this example over here. So here we have a one dimensional w, and we have a loss function shown on the y-axis, and let's say that our current parameter is this value over here. By looking at the gradient, we know that we have to move to the left. But we have to set the step size ada, we have to decide how far to the left we wanna move. If we move too far to the left, we could end somewhere like this. We could end up with a loss value that's higher than where we began. So, this sounds pretty tricky. How are we gonna set this value ada? Now, unfortunately, there's no straight answer to this. Different people have different ways that they like to set this constant. And all I'm gonna do here is to put out two alternatives. One thing that one could do is to set this step size ada to some fixed schedule, like setting ada t to be one over t. In fact, a lot of the mathematical analysis of gradient descent assumes something like this. Another, more sophisticated alternative is to set the step size adaptively. So, we're at a particular vector w, we know which direction the derivative is in, so we know which direction we're gonna move in, and what we can do now is to do a line search along that direction to find the optimal displacement, the value of ada for which the loss will decrease the most. You can imagine some combination of sampling and binary search to achieve this.

Example: logistic regression

$d(\ln u) = \frac{du}{u}$; $d(e^u) = e^u du$

For $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \{-1, +1\}$, loss function

$$L(w) = \sum_{i=1}^{n} \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)})})$$

$w = (w_1, \ldots, w_d)$

$-y^{(i)}(w_1 x_1^{(i)} + \ldots + w_d x_d^{(i)})$

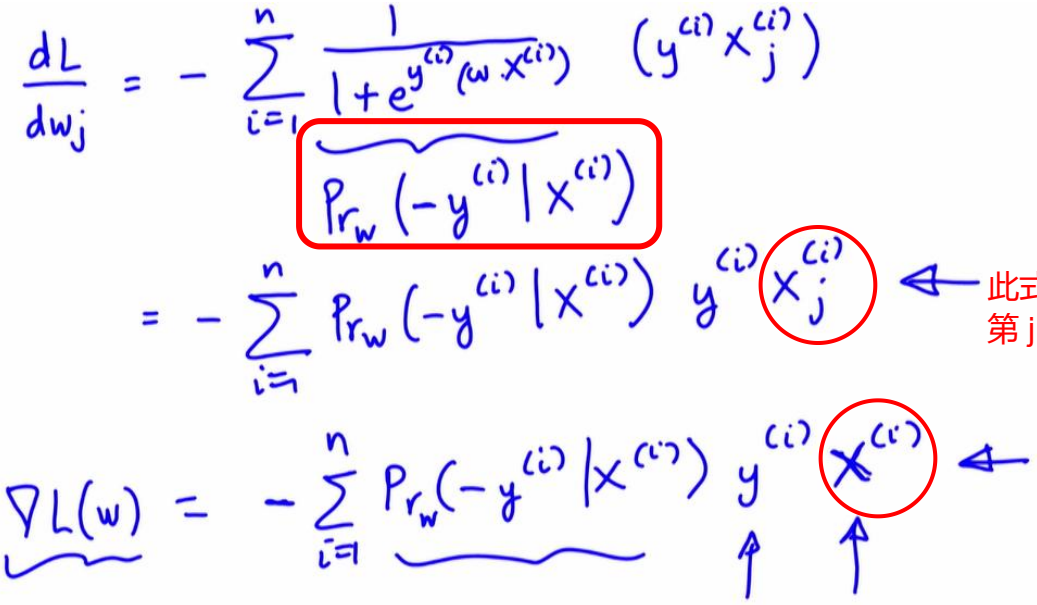
What is the derivative?

$$\frac{dL}{dw_j} = \sum_{i=1}^{n} \frac{e^{-y^{(i)}(w \cdot x^{(i)})}}{1 + e^{-y^{(i)}(w \cdot x^{(i)})}} \left(-y^{(i)} x_j^{(i)}\right)$$

$$\times \frac{e^{y^{(i)}(w \cdot x^{(i)})}}{e^{y^{(i)}(w \cdot x^{(i)})}}$$

$$\rightarrow \frac{1}{1 + e^{y^{(i)}(w \cdot x^{(i)})}}$$

总结:



So we've been talking about gradient descentfairly abstractly, but now let's get more concrete and actually go ahead and derive a gradient descent algorithm, for the specific case of logistic regression. So, let's see, we have a data set, x1, y1 through xn yn. If you recall in logistic regression, you're looking for a linear function given by some parameter vector w that minimizes this loss function over here. Now, actually we usually have an offset v as well. So we're looking for w and v. But if you remember, one of the things we showed is that you can always assimilate v into w by adding an extra feature. So, let's just assume we've done that and that therefore, this is the loss function we're trying to minimize. Now to use gradient descent, all we really need to do is to compute the derivative of this loss function. So, let's go ahead and do that. Okay so, our parameter vector, w, has d numbers is in it. So, w is of the form w1 through wd, and so the derivative is gonna be a d-dimensional vector. So let's go ahead and compute the derivative of the loss with respect to some particular w sub j. What is it? Well, first off, the derivative of the sum is the sum of the derivative. So let's just pull out that summation. And then we have a logarithm. If you remember from high school or a long time ago, the derivative of log u is du over u. So let's go ahead and write down that denominator over there, one plus e to the minus yi w dot xi. And then on top, we have to write down the derivative of that, so it's Okay, so for that there's another rule, the derivative of e to the u is e to the u du. Okay so we get e to the minus yi w dot xi. And now we need to compute the derivative of this thing over here. What is that? So there's a dot product in there, let's just expand it out. So we have minus yi w1 times the first coordinate of xi all the way to wd times the dth coordinate of xi. Okay, we've expanded out the dot product And now we need to take the derivative of that with respect to wj, well there's onlyone term that involves wj. So the derivative is minus yi times xi sub j. Okay, so we've computed the derivative. Now that first term's a little bit messy, so let's simplify it. And one way we can do that is to take it and multiply top and bottom by e to the yi w dot xi. Okay, and if we do that, what do we get? Well the numerator just becomes one, and the denominator becomes 1 plus e to the yi w dot xi. So let's just copy this over now.

So the derivative of the loss function of wj is minus the sum over all points from one to n of one over one plus e to the yi w dot xi times yi $x^{(i)}_j$. Now that first term actually looks familiar. Comparing it to the logistic regression function what we see is it's actually the probability under w of the label being negative yi given xi. And so this derivative is just the minus the sum over all data points of the probability of the opposite label negative yi given xi, the labels are all plus one or minus one, so minus yi is the wrong label, times yi xi j. Okay so that's the derivative in respect to wj, and the overall derivative is the vector in which we just stacked these up. So the overall derivative of the loss function at w is a vector where most of this is exactly the same, and we just have xi at the end. So why is that, why is that what you get when you stack up these individual derivatives? Well, just look at this equation over here. Most of the terms here are scalers, are just numbers, so this is some number, this ($y^{(i)}$) is either plus one or minus one, **the only thing that's a vector is this x at the end**. So the **jth coordinate** of this thing is exactly this formula over here. You copy down the same thing but you justtake the jth coordinate of x.

总结:

Gradient descent for logistic regression

- Set $w_0 = 0$
- For $t = 0, 1, 2, \ldots$, until convergence:

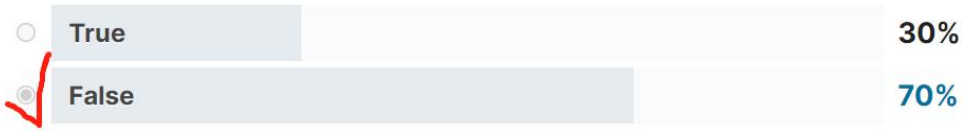
$$w_{t+1} = w_t + \eta_t \sum_{i=1}^{n} y^{(i)} x^{(i)} \Pr_{w_t}(-y^{(i)} | x^{(i)})$$

So we've computed the derivative of the loss functionand now we're ready to write down a gradient descent algorithm. This is what it looks like. We initialize w to zero, then we begin our iterative updates. And when the current parameter vector is w sub t, the update looks like this. We compute a weighted sum of the data points and we simply add it to w sub t with sum step size ada sub t. So what exactly is this weighted sum? This is the ith data point, what is this weight over here? It is the probability of the wrong label under the current parameter vector. In other words, this weight is high if we are doing badly on that data point. And this makes perfect sense, we're placing more emphasis on the points on which we are currently doing badly. Okay, so that's it for now and for gradient descent, you now have, in your repertoire, the gradient descent procedure, and at this point, with a little bit of practice, you should be able to derive a gradient descent algorithm for any loss function you like, any loss function you wish to minimize.

POLL

True or false: a larger step size will result in a faster algorithm and fewer iterations required to find an optimal solution.

结果



总结:

5.3 Unconstrained Optimization III

Topics we'll cover

1. Stochastic gradient descent for logistic regression
2. Stochastic gradient descent more generally

Last time we wrapped up our discussion of gradient descent. What we'll do today is look at a variant of gradient descent that is suitable for very large data sets, it's called stochastic gradient descent and it's an essential part of the machine learning toolbox. So we'll begin by looking at the stochastic gradient descent algorithm for the specific case of logistic regression. And then we'll step back and see how stochastic gradient descent can be derived for more general, arbitrary, loss functions.

Recall: gradient descent for logistic regression

- Set $w_0 = 0$
- For $t = 0, 1, 2, \ldots$, until convergence:

$$w_{t+1} = w_t + \eta_t \sum_{i=1}^{n} y^{(i)} x^{(i)} \Pr_{w_t}(-y^{(i)}|x^{(i)})$$

Each update involves the entire data set, which is inconvenient.

Stochastic gradient descent: update based on just one point:

- Get next data point (x, y) by cycling through data set
- $w_{t+1} = w_t + \eta_t \, y \, x \Pr_{w_t}(-y|x)$

So last time we derived the gradient descent algorithmfor logistic regression. And this is what it looks like. So we want to find the parameter w, a parameter vector that minimizes the logistic regression loss function. And the way we do it, is as follows. We start by initializing w to zero. And then we enter into a loop where we keep updating w. We keep adjusting it until it finally converges. And each update is quite simple. If our current parameter vector is w sub t, what we do is to compute a weighted sum of the data points. Where the weight of a particular point is increased if we are currently doing badly on that data point, if this probability is high. So we compute this weighted sum and then we simply add it to w with a particular step size eta sub t. So this is fairly straightforward. But one down side of this algorithmis that each update involves looking at the entire data set. So if there are n data points, the time taken for an update is order n. this update could be quite slow.And at the end of it all, the magnitude of the update could be something quite minuscule. So a whole lot of work for really a very small benefit. Because of this, in situations where there is a lot of data, there's a popular alternative to this algorithm called **stochastic gradient descent**.

And this is shown over here. So it does much the same thing.Yet again initializes the w to zero and keeps doing updates. But now each update is based on a single data point, not on the entire data set. As a result, the updates are very quick. So it starts with w at zero and then it updates w based just on the first data point. And then it updates w again based on the second data point. And then on the third data point and it goes down the list and when it reaches the end of the data set it just cycles back around to the top and it keeps going in this way. What are these updates? Well here are the updates right here. It's exactly the same as the gradient descent update, except restricted to just one point so there's no summation any longer. So nice, simple, and practical. Now this is just for logistic regression, but something similar can be done for other loss functions, as well. It turns out that the gradient descent procedure typically ends up looking something like this. We end up with a summation over all the data points and then the stochastic gradient descent version is exactly the same thing, but instead of the summation, just doing one data point at a time. So let's look a little bit more carefully at how this all works out.

总结:

Decomposable loss functions

Loss function for logistic regression:

$$L(w) = \sum_{i=1}^{n} \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)})}) = \sum_{i=1}^{n} (\text{loss of } w \text{ on } (x^{(i)}, y^{(i)}))$$

Most ML loss functions are like this: for training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$,

$$L(w) = \sum_{i=1}^{n} \ell(w; x^{(i)}, y^{(i)})$$

where $\ell(w; x, y)$ captures the loss on a single point.

$$\ell(w; x, y) = \ln(1 + e^{-y(w \cdot x)})$$

Here if you will recall is the loss functionfor logistic regression. We are looking for the parameter vector, the model w that minimizes this loss function. And this is a loss function over the entire training set. It can also be expressed as the loss on the first data point, plus the loss on the second data point, plus the loss on the third data point, and so on. The loss function decomposes in this way. And in fact almost all the loss functions that we encounter in machine learning are of this form. So if we have a training set, x one, y one, through x n, y n. Then the loss associated with a particular model w, with a particular parameter vector w, can typically be expressed in this form. Here little l is a function that captures the loss on a single data point. So l of w x, y captures the loss of the model w on the single data point x, y. And the overall loss of w on the entire training set is then just the sum of these point y's losses. So what is this loss function? What does this little l in the case of logistic regression? Well it's just this function over here. So in the case of logistic regression, l of w;x,y is just the natural logarithm of one plus e to the minus y, w dot x.

Gradient descent and stochastic gradient descent

For minimizing

$$L(w) = \sum_{i=1}^{n} \ell(w; x^{(i)}, y^{(i)})$$

$$\nabla L(w) = \sum_{i=1}^{n} \nabla \ell(w; x^{(i)}, y^{(i)})$$

Gradient descent:

- $w_o = 0$
- while not converged:
  - $w_{t+1} = w_t - \eta_t \sum_{i=1}^{n} \nabla \ell(w_t; x^{(i)}, y^{(i)})$

Stochastic gradient descent:

- $w_o = 0$
- Keep cycling through data points (x, y):
  - $w_{t+1} = w_t - \eta_t \nabla \ell(w_t; x, y)$

So let's say we have a loss function of this kind. As we've seen logistic regression is of this form and so is least squares regression, ridge regression, lasso, this is standard. So a loss function looks like this. And we want to find the parameter vector w that minimizes this loss. Now if we gonna use gradient descent to do this, we have to compute the derivative of l. Now l is a summation, so the derivative of l is the sum of the derivatives. So the derivative of l, is the sum from ah equals one to n of the derivative of little l, for the specific point x, i, y, i. So computing a derivative, involves a sum over the data set. Now as a result, this is what the gradient descent procedure looks like. To minimize this loss function, we start w at zero and then we start performing all these updates until we converge. When the current parameter vector is w sub t, the update involves moving in the opposite direction to the derivative. And as we've seen, the derivative works out to this summation over here. And what this means, is that computing one of these updates involves summing over the entire data set, which could be painful. The stochastic gradient descent version does something quite similar, except that each update only involves one data point. So again, it starts w at zero, but now it cycles through all the data points. And when it gets to data point x, y, the update just involves the derivative of little l at that specific point x, y. So it's something much simpler and quicker. And in the case of a very large data set, it's something much more practical.

总结:

Variant: mini-batch stochastic gradient descent

Stochastic gradient descent:

- $w_o = 0$
- Keep cycling through data points (x, y):
  - $w_{t+1} = w_t - \eta_t \nabla \ell(w_t; x, y)$

Mini-batch stochastic gradient descent:

- $w_o = 0$
- Repeat:
  - Get the next batch of points B
  - $w_{t+1} = w_t - \eta_t \sum_{(x,y) \in B} \nabla \ell(w_t; x, y)$

Now one thing worth pointing outis that gradient descent and stochastic gradient descent are really extreme ends of a spectrum. For gradient descent, each update involves the entire data set. And for stochastic gradient descent each update only involves one data point and is therefore potentially quite noisy. A nice compromise is to have each update depend on a reasonable amount of data, say 100 or a thousand data points. This is called mini-batch stochastic gradient descent. So what happens in this case, is that you start again with w at zero. But now the first update involves a batch of say the first 100 or thousand data points. So you update based on those points. Do the usual gradient descent update based on those points and then you move onto the next batch of a thousand points. Do one update based on those and then the next batch of a thousand points, and so on. And when you reach the bottom, you cycle back around to the top. So it's a very reasonable and sensible compromise. Okay, well that's it for today. We now have in our repertoire, two extremely powerful and general purpose optimization techniques, gradient descent and stochastic gradient descent. These have got to be among the most influential algorithms of all time.

POLL

Rank the following in order of execution time, from fastest to slowest:

Gradient descent, stochastic gradient descent, mini-batch stochastic gradient descent.

gradient descent, mini-batch stochastic gradient descent, stochastic gradient descent	10%
mini-batch stochastic gradient descent, gradient descent, stochastic gradient descent	4%
✓ stochastic gradient descent, mini-batch stochastic gradient descent, gradient descent	68%
mini-batch stochastic gradient descent, stochastic gradient descent, gradient descent	19%

总结：

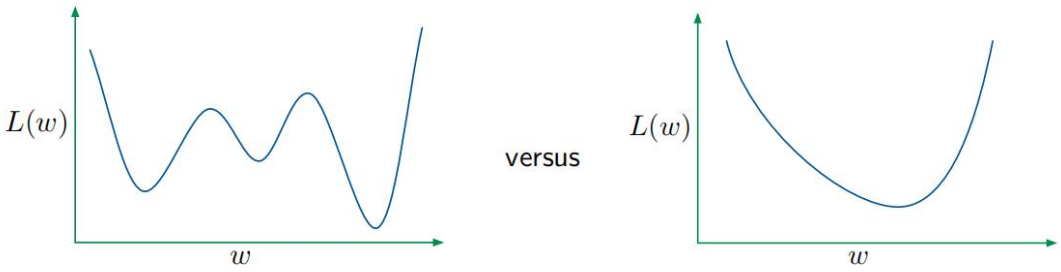
5.4 Convexity I

Topics we'll cover

1. Definition of convexity
2. The second-derivative test for convexity

As we have seen, learning often boils down to optimization. We define a suitable loss function, and then we learn parameters by minimizing that function. Now, some loss functions are much better behaved that others and permit a wide variety of different solution methods. So it's important to be able to identify these special cases. The loss functions that are the best behaved of all are those that are convex. This is a concept that we have been skirting around for a little while, and today, we'll finally get to focus on it. So we'll begin by defining the notion of convexity. And then we'll see that establishing convexity often boils down to looking at second derivatives.

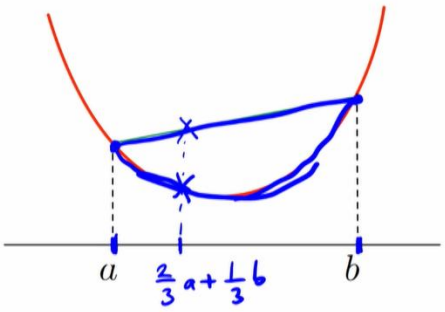
Is our loss function convex?

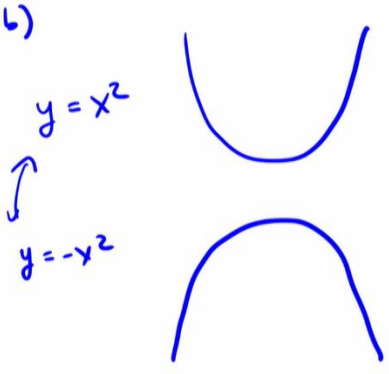


This is the general situation we're in. We have some loss function for our learning problem, which is basically an equation, a formula of some sort. And we want to know, does it look like the picture on the left, or does it look like the picture on the right? Does it have a whole bunch of local optima that we could get trapped in and that are ultimately gonna become the focus of the optimization? Or, is it a much nicer situation like the one on the right, which is gonna be a lot easier to optimize and will permit us to use a wide range of methods? To start with, what is the formal property that makes the picture on the right so nice? It's called convexity.

Convexity

$f\left(\frac{2}{3}a + \frac{1}{3}b\right) \leq \frac{2}{3}f(a) + \frac{1}{3}f(b)$





A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if for all $a, b \in \mathbb{R}^d$ and $0 < \theta < 1$,

$$f(\theta a + (1-\theta)b) \leq \theta f(a) + (1-\theta)f(b).$$

It is **strictly convex** if strict inequality holds for all $a \neq b$.

f is **concave** $\Leftrightarrow -f$ is convex

总结:

Let's take a look. So, what does it mean for a function to be convex? Here's what it means. Take any two points, a and b, here's a and here's b, and look at the value of the function at these points. So f of a and f of b and now, draw a line between them, so we have this green line over here. What we need is that between a and b, the function should lie underneath that line, which is exactly what is happening here. The function lies entirely underneath that line. Okay? So, for instance, if we were to look at this point over here, that point may be 2/3 a plus 1/3 b, the value of the function at that point is f of 2/3 a plus 1/3 b, so that's this value over here. And the value of the line at that point, this value over here is 2/3 f of a plus 1/3 f of b. Now, from the picture, we can see that that left-hand side is less than the right-hand side. The curve lies under the line, and we need this to be true not just for 2/3 and 1/3, but for every point between a and b.

So, in this definition of convexity, that's the role played by theta. Theta is a fraction that goes from zero to one, and as theta goes from zero to one, we get all the points that lie between a and b, and for every single one of these points, we need the curve for f to lie underneath that line. Now, the canonical example of a convex function is perhaps y equal to x squared, it looks like this. Sometimes, we are also interested in functions that look like upside-down bowls, like y equals negative x squared, that looks like this. That's called a concave function. And in general, we say a function is concave if negative of that function is convex, as we see in this particular example.

Checking convexity for functions of one variable

A function $f : \mathbb{R} \to \mathbb{R}$ is convex if its second derivative is ≥ 0 everywhere.

Example: $f(z) = z^2$

$\frac{df}{dz} = 2z$

$\frac{d^2f}{dz^2} = 2$

So we have the definition of convexity. How do we check if a given function is convex? This is particularly easyif the function has just one variable. If we have a function from R to R, so a function of just one variable, a quick check for convexity is to compute the second derivative. And if it turns out that the second derivative is always greater than or equal to zero, it means that the function is convex. Let's look at this example here. Let's look at f of z equals z squared. We know this function is convex. Let's see how this text works out. What's the first derivative? Df, dz is two z. So that's the first derivative. The second derivative, d squared f, dz squared is two. Okay, and that's greater than zero, so the test really does work in this example. So it's a nice simple test.

But the loss functions that we'll typically be dealing with are not functions of just one variable. What does one do when one has a function of multiple variables? How does one check convexity in those cases?

Checking convexity

Function of one variable	Function of d variables
$F : \mathbb{R} \to \mathbb{R}$	$F : \mathbb{R}^d \to \mathbb{R}$
• Value: number	• Value: number
• Derivative: number	• Derivative: d-dimensional vector
• Second derivative: number	• Second derivative: $d \times d$ matrix
Convex if second derivative is always ≥ 0	Convex if second derivative matrix is always positive semidefinite

Here is the big picture that's useful to bear in mind. When you have a function of just one variable, which is the case over here, then the value of function at any point is a number, and the derivative of the function at any given point is a number, and the second derivative at any given point is a number, and the function is convex if that second derivative is always greater than or equal to zero. Now let's say you have a function of d variables. In that case, the value of the function at any given point is again a number, but the first derivative at any point is now a d-dimensional vector, and the second derivative at any point is a d by d matrix. This function is convex if this second derivative matrix is always positive semidefinite. So what does all of this mean? To start with, why is the second derivative a d by d square matrix? Let's take a look.

总结:

First and second derivatives of multivariate functions

For a function $f : \mathbb{R}^d \to \mathbb{R}$,

- the first derivative is a vector with d entries:

$$\nabla f(z) = \begin{pmatrix} \frac{\partial f}{\partial z_1} \\ \vdots \\ \frac{\partial f}{\partial z_d} \end{pmatrix}$$

- the second derivative is a $d \times d$ matrix, the **Hessian** $H(z)$:

$$H_{jk} = \frac{\partial^2 f}{\partial z_j \partial z_k}$$

$$H(z) = \begin{bmatrix} \frac{d^2 f}{dz_1^2} & \frac{d^2 f}{dz_1 dz_2} & \frac{d^2 f}{dz_1 dz_3} & \cdots \\ \frac{d^2 f}{dz_2 dz_1} & \frac{d^2 f}{dz_2^2} & & \\ & & \ddots & \end{bmatrix}$$

So we have some function on d variables,so it's a function that maps d-dimensional vectors to just some number. Now, we've already talked about the first derivative. We do have a function like this with d variables, then there are d things, with respect to which you can take the derivative. So a derivative with respect to the first variable, the second variable, all the way to the d variable. So you can compute each of these individual derivatives and then package them into a vector. And that's why the first derivative is a d-dimensional vector. What about the second derivative? So now, we're taking the derivative twice, and for the first time around, we have a choice of d variables. And for the second time, we again have a choice of d variables. So there's a total of d squared options, and we write them together in a d by d matrix, and this matrix is called the Hessian. So it's a d by d matrix, where the jk entry of the matrix is the second derivative with respect to the jth variable and kth variable. So the matrix looks something like this. It's a d by d matrix, and the first entry is the second derivative of f with respect to z one squared. The second entry is the derivative with respect to z one and then z two. And then, the second derivative with respect to z one and then z three, and so on. And, on the next row, the first entry is with respect to z two and then z one, and then the diagonal entry is z two squared, and so on. So for a function of d variables, the second derivative is actually a matrix, a d by d matrix.

Example

Find the second derivative matrix of $f(z) = \|z\|^2$.

$$f(z_1, \ldots, z_d) = z_1^2 + z_2^2 + \cdots + z_d^2$$

$$\frac{df}{dz_j} = 2z_j$$

$$\frac{d^2 f}{dz_k dz_j} = \begin{cases} 2 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}$$

$$H(z) = \begin{bmatrix} 2 & & 0 \\ & 2 & \\ & & \ddots \\ 0 & & 2 \end{bmatrix} = 2I$$

Let's work this out in a specific example. Here, we have a function of d variables, so let's just right it out in full a function of z one to z d, and the function is just the length of z squared. So it's z one squared plus z two squared plus all the way to z d squared. Let's compute the second derivatives of this function. So first, the derivative with respect to zj. Well, zj is only in one of those terms, zj squared, so the first derivative is two zj. Now, let's compute the second derivative with respect to zk and then zj. So we take this thing over here, and now, we're gonna compute its derivative with respect to zk. So what if k equals j, then the derivative is two. And if j is not equal to k, then the derivative is zero. So the second derivative matrix, in this case, is a diagonal matrix. So the second derivative matrix at a point z looks like this. Along the diagonals, it has twos, and everywhere else, it's zero. So a concise way of writing this is to say that it's twice the identity matrix. Okay, we now we know what the second derivative matrices are.

Let's go back to our overall picture. So, we're dealing with functions of d variables. The value of the function is a number, the derivative is a vector, the second derivative is a matrix, and function is convex if the second derivative matrix is positive semidefinite. What does that mean? That sounds mysterious. Well, it turns out, it's a very important concept in machine learning, and it's what we'll turn to next. But we've done quite a bit for today, so let's take a break now, and when we meet next, we'll talk all about positive semidefinite matrices. See you then.

POLL

For a function of d variables, the first derivative results in a d-dimensional vector. The second derivative results in a dxd matrix called the Hessian. Is the Hessian symmetric?

Yes, the Hessian is always symmetric	60% ✗
No, the Hessian is never symmetric	5%
✓ The Hessian is typically symmetric, but not always.	35%

总结:

5.5 Linear algebra IV: Positive Semidefinite Matrices

Topics we'll cover

1. Positive semidefinite matrices
2. Properties of PSD matrices
3. Checking if a matrix is PSD
4. A hierarchy of square matrices

Positive semidefinite matrices are square matricesthat have many special properties. They arise all over the place in machine learning. So today, we'll see what it means for a matrix to be positive semidefinite, and we'll look at some of the properties of these matrices. We'll also cover the important question of how you figure out if a matrix is of this type.

When is a square matrix "positive"?

- A superficial notion: when all its entries are positive
- A deeper notion: **when the quadratic function defined by it is always positive**

Example: $M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ $\quad x \longmapsto x^T M x$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto (x_1 \; x_2) \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$= (x_1 \; x_2) \begin{pmatrix} x_1 + x_2 \\ x_1 + x_2 \end{pmatrix}$$

$$= x_1^2 + 2x_1x_2 + x_2^2$$

$$= (x_1 + x_2)^2 \geq 0$$

Now, when we were talking about convexity, we said that a function of one variable is convex if its second derivative is always positive. When you have a function of D variables however, the second derivative is a D by D matrix. What might it mean for a D by D square matrix to be positive? How could we define that? Well, one natural notion is to say that a matrix is positive if all the entries in it are positive. This is reasonable enough, but it turns out to be a superficial notion that doesn't really help us very much. There's a deeper and more insightful notion. We can say that a matrix, a square matrix, is positive if the quadratic function defined by it is always positive. Hmmm, let's see what this might mean. Okay, so here's a matrix. What is the quadratic function defined by this matrix? Well if you recall, the quadratic function defined by this matrix is the function that maps a vector x to x transpose Mx. So in this particular example, x has to be a two dimensional vector, okay. And the quadratic function sends this vector, say x one, x two, to x transpose, so x one, x two times M, one, one, one, one times x, x one, x two. What is that? Well, let's work it out. Let's start by doing this. So we get x one, x two, and here we get x one plus x two, x one plus x two. And now we do the last product and we get x one squared plus two x one x two plus x two squared. This is the quadratic function defined by the matrix, okay? Now if you look at the function, you can see that it can also be written like this: x one plus x two squared. And when you write it like that, it's a square which means that it always has to be greater than or equal to zero, no matter what x you plug in, okay? We have a matrix, in this case two by two. It defines a quadratic function on two variables, and this function is always greater than or equal to zero no matter what x you plug in. So we call this matrix M positive semidefinite.

总结:

Positive semidefinite matrices

Recall: every **square** matrix M encodes a **quadratic function**:

$$x \mapsto x^T M x = \sum_{i,j=1}^{d} M_{ij} x_i x_j$$

(M is a $d \times d$ matrix and x is a vector in $\mathbb{R}^d$)

A symmetric matrix M is **positive semidefinite (psd)** if:

$$x^T M x \geq 0 \quad \text{for all vectors } x$$

More generally, let's say we have a D by D square matrix. In that case the quadratic function defined by the matrix is this one, that sends a D dimensional vector x to x transpose Mx. So what exactly is x transpose Mx? Well, it works out to a number. And if you multiply it out, you'll see that it's actually just the summation. It's the sum over all ij of the ij entry of the matrix, times the i-th entry of x times the j-th entry of x. And if you add all those things up, that's x transpose Mx. So we'll say that a matrix N is positive semidefinite if first of all, it is symmetric, and second, if this quadratic function, x transpose Mx, is always greater than or equal to zero, no matter what x you plug in. This turns out to be a very powerful notion of what it means for a matrix to be positive.

A symmetric matrix M is **positive semidefinite (psd)** if:

$$x^T M x \geq 0 \quad \text{for all vectors } x$$

We saw that $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ is PSD. What about $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$?

$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto x_1^2 + 4x_1x_2 + x_2^2$

$x_1 = 1$, $x_2 = -1$ $\rightarrow 1 - 4 + 1 = -2$

Let's look at an example. So, we've seen that this matrix is positive semidefinite, we just worked through that example. What about this other matrix over here? Is that positive semidefinite as well? Well, it's a two by two matrix, so the quadratic function is the one that sends x one, x two to x transpose Mx, and if we just plug into our usual expansion, this would be, it would send this to x one squared plus four x one x two plus x two squared. That's the quadratic function given by the matrix. So is this always greater than or equal to zero? You know what, let's try some values. What about trying x one equals one and x two equals minus one? What happens then? In that case we get one minus four plus one, which is negative two. Okay, so this matrix is not positive semidefinite. And yet, it looks more positive than the other matrix, which we know is positive semidefinite.

Here's another question.Suppose we have a diagonal matrix. Under what conditions is it positive semidefinite? So we have some diagonal matrix, the diagonal entries are M_{11}, M_{22}, all the way to M_{dd}. And everything else is a zero. When is a matrix like this positive semidefinite? Well, we know x^TMx has to be greater than or equal to 0. Why don't we try this x? Let's try the vector x, which is 0 everywhere, except for a one at the i-th position. Okay so it's all zeroes except for a one at the i-th position. What is x transpose Mx? Well if we think of our summation, the sum of Mijxixj, you can see that x transpose Mx is just the i i entry of the matrix. And so we know that this has to be greater than or equal to zero. So that gives us a simple condition. A diagonal matrix is positive semidefinite if and only if all the diagonal entries are greater than or equal to zero. So for diagonal matrices, it's very easy to check.

A symmetric matrix M is **positive semidefinite (psd)** if:

$$x^T M x \geq 0 \quad \text{for all vectors } z$$

When is a diagonal matrix PSD?

$$M = \begin{pmatrix} M_{11} & & 0 \\ & M_{22} & \\ 0 & & \ddots \\ & & & M_{dd} \end{pmatrix}$$

M PSD $\Leftrightarrow$ all $M_{ii} \geq 0$

$x = (0, 0, 0, \ldots, 1, 0, \ldots, 0)$ ← position i

$x^T M x = M_{ii}$

总结:

A symmetric matrix M is **positive semidefinite (psd)** if:

$$x^T M x \geq 0 \text{ for all vectors } z$$

If M is PSD, must cM be PSD for a constant c? yes, if $c \geq 0$

$$x^T (cM) x = c \underbrace{(x^T M x)}_{\geq 0}$$

Now suppose we have a positive semidefinite matrix M,what about two times M? Is that positive semidefinite as well? Yes it is. And more generally what about c times M? Well, let's take a look. So, we have x transpose c times Mx, we can see it's just a constant, so we can pull it out. So it's c times x transpose Mx. And M is positive semidefinite, so this is greater than or equal to zero. C times that, is that greater than or equal to zero? Yes, provided c is greater than or equal to zero. So the answer is yes if c is greater than or equal to zero.

A symmetric matrix M is **positive semidefinite (psd)** if:

$$x^T M x \geq 0 \text{ for all vectors } z$$

If M, N are of the same size and PSD, must $M + N$ be PSD? yes

$$x^T (M+N) x = \underbrace{x^T M x}_{\geq 0} + \underbrace{x^T N x}_{\geq 0}$$

$$\geq 0$$

Here's a related question. Suppose you have two matrices that are each positive semidefinite. What about their sum? Is that positive semidefinite? Well again, we can just try directly by plugging in to the definition. Let's take their sum, M plus N, and look at x transpose M plus N x. Now we can factor this, this is just equal to x transpose Mx plus x transpose Nx. And each of these are greater than or equal to zero. So the whole thing is greater than or equal to zero. And so the answer to this question is just a simple "yes."

Checking if a matrix is PSD

A matrix M is PSD if and only if it can be written as $M = UU^T$ for some matrix U.

Quick check: say $U \in \mathbb{R}^{r \times d}$ and $\underbrace{M = UU^T}_{r \times r}$. $(AB)^T = B^T A^T$

1. ✓ M is square.
2. ✓ M is symmetric. $(uu^T)^T = (u^T)^T u^T = uu^T$
3. ✓ Pick any $x \in \mathbb{R}^r$. Then

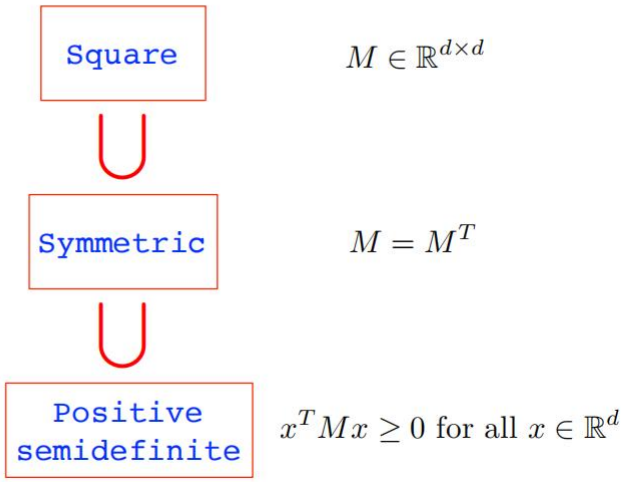
$$\underbrace{x^T M x}_{\checkmark} = \underbrace{x^T U U^T x}_{\checkmark} = (x^T U)(U^T x)$$
$$= \underbrace{(U^T x)^T (U^T x)}$$
$$= \underbrace{\|U^T x\|^2}_{} \geq 0.$$

Another useful fact: any covariance matrix is PSD.

总结:

So we've seen many properties of positive semidefinitematrices, but the thing that we're gonna have to do pretty often is to figure out whether a given matrix is positive semidefinite. Now technically that means that we have to check if x transpose Mx is greater than or equal to zero for all x. That's difficult to do because there are infinitely many choices for x. Is there a simple way? Is there some more direct way of checking whether a matrix is positive semidefinite? And indeed there is. There's a very simple condition. It's positive semidefinite if it's of the form UU transpose. So let's just briefly go over the justification for this. So, let's say that M is of this form, UU transpose, and U is just any old matrix, let's say it's an R by D matrix, okay. And let's just go and check all the properties for positive semidefinite. So first of all, is M even a square matrix? Well let's see. M is UU transpose. U is R by D, so, and U transpose is D by R, so M is an R by R matrix. Yes, it's square. Okay, we're off to a good start. Now is M symmetric? Okay. Let's see. Well, UU transpose, what's the transpose of that? What is that equal to? Well if you remember we had this formula for transpose, AB transpose is B transpose, A transpose, so if we apply that we get U transpose transpose times U transpose. What is transpose transpose, well if you just transpose a matrix twice, you get back the same matrix, so that's just UU transpose. And yes, M transpose equals M, so it is symmetric. Okay, so just one last property to check, okay. So x transpose Mx, we plug in, UU transpose for M. And now we have this matrix vector product involving four terms, we can just parenthesize this however we like. So let's do it like this, let's put the parentheses there. Okay. And let's look at these terms. Now, x transpose U. By again using this transpose formula, we can just write that as U transpose x transposed. And now, U transpose x is just a vector, and so we get something of the form, vector transposed with itself. That's familiar, if you remember Z transpose Zis just the length of Z squared, okay? So U transpose x transposed times U transpose X is just the length of U transpose x squared. And since it's the squared length, it has to be greater than or equal to zero. And so a matrix of this form has to be positive semidefinite. This will turn out to be a very useful rule. Now, essentially the same argument can be used to show something else that will be useful which is that any covariance matrix is guaranteed to be positive semidefinite.

A hierarchy of square matrices



Here's a picture that I would like to leave you with.We're talking about square matrices, say D by D matrices. Some of these matrices are symmetric, that is M equals M transpose. This is a special property. An even more special case is when the quadratic function defined by the matrix, x transpose Mx, is always greater than or equal to zero, no matter what x you plug in. These are the positive semidefinite matrices. They arise all the time in machine learning and we rely upon them very heavily in many different contexts. Okay, that's enough for today. We've finally seen what it means for a matrix to be positive semidefinite, and we've already seen one use of these matrices in deciding whether a function is convex. We'll soon see many more uses. See you soon.

POLL

True or false: Diagonal matrices are always positive semidefinite.

结果

○	True	22%
✓	False	78%

总结:

5.6 Convexity II

Topics we'll cover

1. Second derivative test for convexity
2. Convexity examples

A lot of machine learning is aboutminimizing loss functions and the way you do that depends in part upon whether the function is convex. Today we'll see how we can check that. So we'll review the second-derivative test for convexity, and then we'll go through a whole slew of examples.

Second-derivative test for convexity

A function of several variables, $F(z)$, is convex if its second-derivative matrix $H(z)$ is positive semidefinite for all z.

More formally:
Suppose that for $f : \mathbb{R}^d \to \mathbb{R}$, the second partial derivatives exist everywhere and are continuous functions of z. Then:

1. $H(z)$ is a symmetric matrix
2. f is convex $\Leftrightarrow$ $H(z)$ is positive semidefinite for all $z \in \mathbb{R}^d$

H P.S.D. $\Leftrightarrow$ $x^T H x \geq 0$ for all x

$\Leftarrow$ $H = UU^T$

We've already seen what it means for a function to be convex: when we have a function of d variables, it's convex if its second-derivative matrix which is a d by d matrix, is positive semi-definite at all points. And so what exactly does this mean? So at any given point, the second-derivative matrix is a d by d matrix. It's positive semi-definite, a matrix H is positive semi-definite if the quadratic function defined by the matrix x transpose H x is always greater than or equal to zero, no matter what x you plug in. We also saw an alternative characterization of positive semi-definite matrices that is somewhat easier to use in many situations. Okay, so a matrix is positive semi-definite if it can be written in the form u u transpose, where u is any old matrix. Okay so we'll be making use of both of those definitions.

Example

Is $f(x) = \|x\|^2$ convex?

$y = x^2$

$H(x) = 2I$

$z^T(2I)z = 2z^T I z$

$= 2z^T z = 2\|z\|^2 \geq 0$

So let's look at an example.This function If of x equals the length of x squared. Is it convex? Well the first step is to compute its second-derivative matrix, the Hessian. And we actually did that last time and this turned out to be twice the identity matrix. Is that positive semi-definite? Well let's see. What is z transposed times twice the identity, times z? Well we can pull the two out, and we get z transposed times the identity times z. The identity times z is just z. So this is twice z transpose z which is twice the length of z squared which is greater than or equal to zero. So this is positive semi-definite. Another way to see it is that matrix twice the identity is a diagonal matrix and all the entries along the diagonal are positive, so the matrix is positive semi-definite. Good. So f is a convex function, it looks like a bowl. And in fact it's the high-dimensional analog of our classical convex function, y equals x squared. This is the d dimensional version of exactly the same thing.

总结:

Example

Fix any vector $u \in \mathbb{R}^d$. Is this function $f : \mathbb{R}^d \to \mathbb{R}$ convex? yes.

$$f(z) = (u \cdot z)^2$$

$$f(z_1, .., z_d) = (u_1 z_1 + u_2 z_2 + .. + u_d z_d)^2$$

$$\frac{df}{dz_j} = 2(u_1 z_1 + .. + u_d z_d)\, u_j$$

$$\frac{d^2 f}{dz_k\, dz_j} = 2 u_j u_k$$

$$H(z) = 2uu^T \quad \text{P.S.D.}$$

Let's do another example. So here u is any fixed vector in d dimensional space and we define a function f of z, so z has d variables, z one to z d. And f is simply the dot product we've used, squared. So it's u one, z one, plus u two, z two all the way to u d, z d squared. Is this a convex function? Well, let's start by computing the second derivative, okay? So first, what is d f, d c ? What is the first derivative with respect to z sub j? When we have something of the form, something squared, we first pull down the two, then we copy down the whole expression. And then we take the derivative of the inside part with respect to z j and that's just u j. Okay, so that's the first derivative with respect to z j. And now let's take the second derivative, with respect to z sub k of this thing. Okay? So we're going to start with this and now we're going to take the derivative with respect to z sub k. Now, in this whole expression there's only one term that involves z sub k, okay? And that's the term with u k, z k. So the second derivative is just two u j, u k. So, the second derivative matrix is a d by d matrix, whose j k entry is twice u j, u k. One concise way of writing this is to say that the Hessian at point c is twice u u transpose. So do you remember this? If you have a vector, u then u u transpose is a d by d matrix whose i j entry is u i, u j. So that's exactly what we need in this case. So is this positive semi-definite? Well, u u transpose is positive semi-definite, it has that special form that immediately identifies as being of this type, and when we multiply it by two, it remains positive semi-definite. So this psd, and so this function, is indeed convex.

Least-squares regression

Recall loss function: for data points $(x^{(i)}, y^{(i)}) \in \mathbb{R}^d \times \mathbb{R}$,

$$L(w) = \sum_{i=1}^{n} (y^{(i)} - (w \cdot x^{(i)}))^2 = \sum_i \left(y^{(i)} - (w_1 x_1^{(i)} + .. + w_d x_d^{(i)})\right)^2$$

$$\frac{dL}{dw_j} = \sum_i 2\left(y^{(i)} - (w_1 x_1^{(i)} + .. + w_d x_d^{(i)})\right)(-x_j^{(i)})$$

$$\rightarrow = -2 \sum_i \left(y^{(i)} - (w_1 x_1^{(i)} + \cdots + w_d x_d^{(i)})\right) x_j^{(i)}$$

$$\frac{d^2 L}{dw_k\, dw_j} = -2 \sum_i \left(-x_k^{(i)} x_j^{(i)}\right) = 2 \sum_i x_j^{(i)} x_k^{(i)}$$

$$H(w) = 2 \sum_{i=1}^{n} \underbrace{x^{(i)} x^{(i)T}}_{\text{P.S.D.}}$$

P.S.D.

P.S.D.

Convex.

总结:

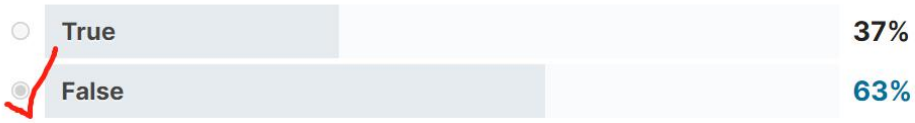
Okay, those were warm-ups, let's move onto a somewhat more challenging example.
So this if you will recall was our loss function for least-squares regression. So we have a set of data points, n data points x i, x j, where the xs are vectors in d dimensional space and the ys are the response values, real values. And we want to find a linear function given by w that minimizes the squared error. Now, normally we would also have an intercept term in here, a plus b. But, if you recall, we had a way of assimilating that into w, so you can imagine that we've already done that and that this is the function we now want to minimize. Is this a convex function? Well let's start by computing the second derivative and on route to that, the first derivative. So our function over here is the sum over i of y I minus w dot x i, that is w one, x i one, all the way to w d, x i d, squared. Okay, so the derivative with respect to w sub j. Well the derivative of the sum is the sum of the derivative. So, we can just pull the summation out. Then we have something of the form u squared, the derivative of that is two u d u so we bring down the two and we have y i minus this whole thing over here, w one x i one, all the way to w d, x i d, okay? And now we take a derivative of just that inside part with respect to w j. So we get negative, x i j. Okay? That's the first derivative. Let's pull the minus two in front, just for convenience. Okay, so we get something like this. w d, x i d, x i j. Okay, now let's take the derivative with respect to w sub k. So we take this expression and we take the derivative of that with respect to w sub k, so that's the second derivative. And what do we get? Well for the term in the middle, there's only one term that involves w sub k. So we get negative x i k, x i j.

And so the negatives cancel out and if we just copy this over, what we have is the d squared I with respect to w k, w j, is twice the summation over i. Let's see what we had, x i j and x i k. So the jth coordinate of the ith data point and the kth coordinate of the ith data point. Okay so this is the second derivative, or at least one term of the second-derivative matrix. What's the entire matrix? Well this is again something of that special form, u u transpose. So if you remember you have that vector u, u u transpose is a d by d matrix whose i j entry is u i, u j, okay? So this entire matrix over here, the Hessian and w is twice times the sum over i, equals one to n, of the ith data point, ith data point transposed. Okay, this is a d by d matrix over here. So that's the Hessian, is this positive semi-definite? Well this particular term here does have that u u transposed form, so this is positive semi-definite. Okay? And we've already seen that if you have the sum of positive semi-definite matrices, that's also positive semi-definite. So this is also positive semi-definite. And we've seen that multiplying a positive semi-definite matrix by a positive constant, again leaves you with something positive semi-definite. So the whole thing is positive semi-definite and that means that this is a convex function. And this is one of the many reasons why we like the least-squares loss function so much.

Okay well that's it for today. I think we have convexity under control now and in fact we have all the tools we need to get started on our next set of classification methods. See you soon.

POLL
True or false: multiplying a positive semidefinite matrix with any constant will result in another positive semidefinite matrix

结果



总结: