Matrix differentials

So far we've learned a lot of tools for working with linear functions and linear equations. Of course, in reality, we need to deal with nonlinear functions. One good way to do this is to use linear approximations: work with a linear function that is close to our true nonlinear function in a local region. We can get these linear approximations by differentiating to form a Taylor series; that's the main subject of this set of lecture notes.

A common complication in machine learning is that our nonlinear functions have high-dimensional arguments: multiple vectors, matrices, or even tensors. It's possible but unwieldy to differentiate such functions component by component; instead it's often better and faster to work directly in matrix notation. Tools for the latter are called *matrix differential calculus*.

In these notes we'll use concrete coordinates for all of the vectors and matrices. Many of the same techniques work for abstract vector spaces, but with concrete representations, we get to see how the overall derivatives depend on the derivatives for individual coordinates.

First-order Taylor approximation

We can approximate any nonlinear function $f \in \mathbb{R}^n \to \mathbb{R}^m$ around a point $(\hat{x}, f(\hat{x}))$ with a linear function called the *first-order Taylor approximation* or *Taylor series* for f at $\hat{x}$:

$$y - \hat{y} \approx A(x - \hat{x}) \qquad y, \hat{y} \in \mathbb{R}^m,\ x, \hat{x} \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times n}$$

Here $y = f(x)$ and $\hat{y} = f(\hat{x})$. And, the matrix A is the *Jacobian* of f at $\hat{x}$; that is, it is the first derivative of y with respect to x, evaluated at $\hat{x}$:

$$A_{ij} = \left.\frac{\partial y_i}{\partial x_j}\right|_{\hat{x}}$$

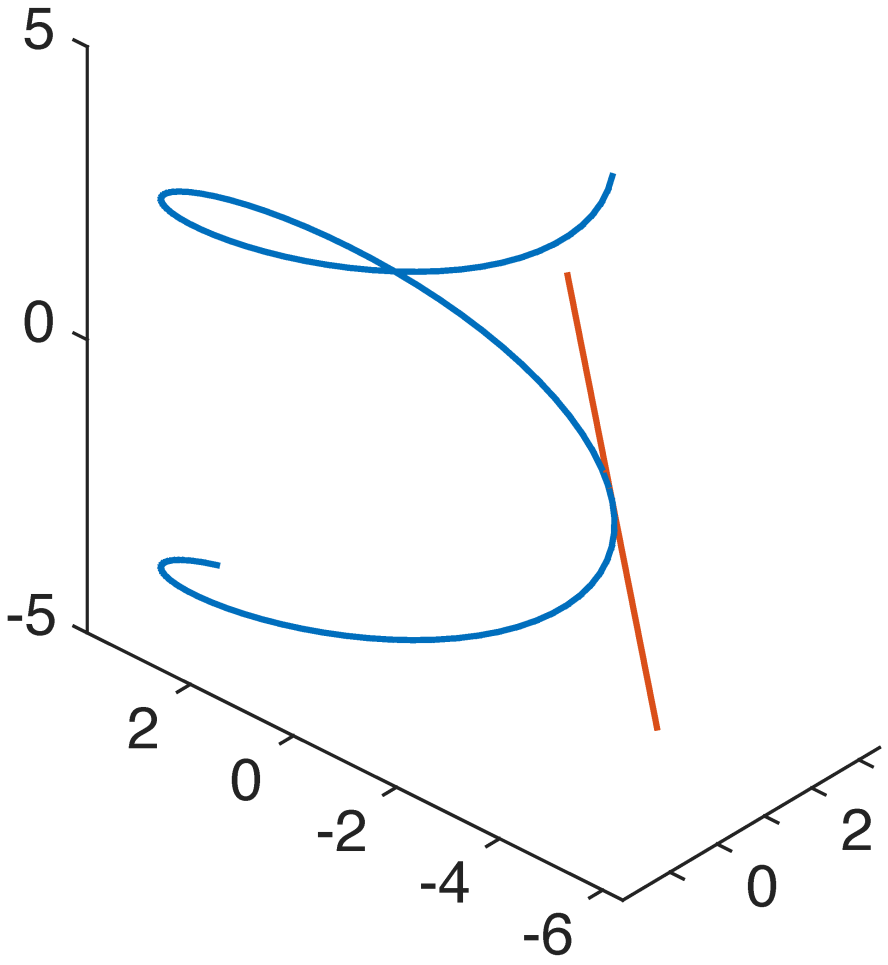
The Jacobian A implicitly depends on the point $\hat{x}$ where we evaluate the derivative. We can be more precise by writing the Taylor approximation as

$$(y - \hat{y}) \approx A(\hat{x})\,(x - \hat{x})$$

> *Note that there are competing conventions for how to represent the Jacobian: some books define it to be the transpose of what we've written here. Our convention is called the numerator layout, since the numerator (∂y_i) corresponds to the first dimension of A (the row index).*

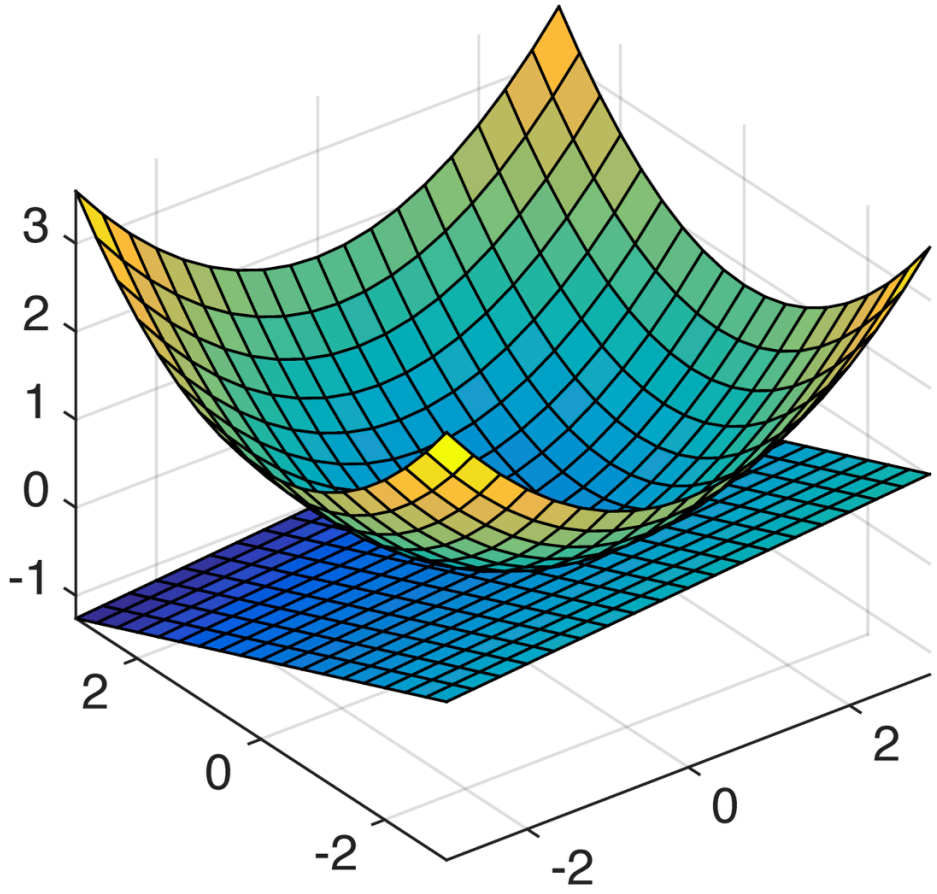
Taylor series examples

If we pick $n = 1$ so that f represents a curve, then the Taylor approximation is a line that is tangent to the curve:



The matrix A will be a column vector, and represents the instantaneous velocity of a point moving along the curve.

If instead we pick $m = 1$, so that f represents a curved surface in $\mathbb{R}^{n+1}$, then the Taylor approximation is a hyperplane (an n-dimensional subspace of $\mathbb{R}^{n+1}$) that is tangent to the surface:



The Jacobian matrix A will be a row vector, equal to the gradient of f. So, the components of A will be the slopes of the hyperplane in each direction.

Differentials

We can also write the first-order Taylor approximation using a somewhat different but equivalent notation:

$$dy = A\,dx$$

Informally, we can think of dy as being short for $y - \hat{y}$, and dx as being short for $x - \hat{x}$. More formally, dy is a vector in the *tangent space* to f at $\hat{x}$, as in the plots in the previous section. (That's why we can write $=$ instead of $\approx$ above: the tangent space is linear even if f is not.)

The vectors dx and dy are called *differentials*. We often think of them as small changes in x and y, and that's a reasonable intuition since the Taylor approximation is typically best in a small region about $\hat{x}, \hat{y}$. But in fact it's perfectly valid to consider large values of dx, dy; it's just that the tangent space may no longer be close to f in this case.

No matter what inner product space we're working in, a differential always has the same type as its original variable. For example, we could have a matrix $A \in \mathbb{R}^{3\times 2}$ that depends on a scalar $x \in \mathbb{R}$. In this case $dx \in \mathbb{R}$ would be a scalar, and $dA \in \mathbb{R}^{3\times 2}$ would be a matrix.

A good mnemonic for the new notation is that if we "divide through by dx", we get $\frac{dy}{dx} = A$, which states that A is the first derivative of y with respect to x. Just like the menmonic for the chain rule, the notation is only suggestive, since we can't actually divide by a vector like dx.

Just as before, we can make it explicit that A depends on x:

$$dy = A(x)\,dx$$

> *Unlike the notation for the Taylor approximation, note that we've written $A(x)$ instead of $A(\hat{x})$. This convention is traditional, and saves a bit of ink, but it can be confusing: in $A(x)\,dx$, the x in $A(x)$ means the place where we evaluate the derivative, while the x in dx means the variable we are taking the derivative of. There's no ambiguity since one is inside a differential and one is not.*

We can use the same sort of notation to work with more-complicated expressions as well. In general, $d(\text{expression})$ means the linear part of a change in the value of expression. So, if we work out a formula for $d(\text{expression})$, it lets us directly read off a first-order Taylor series. For example, the equation

$$d(x^2 + 2x + 3) = 2x\,dx + 2dx$$

says that, if we make a change dx to the value of x, the value of $(x^2 + 2x + 3)$ changes by $(2x + 2)dx$ to first order. (We haven't yet shown how to derive this equation using differential notation, but we can check it using our knowledge of scalar derivatives.)

So why did we bother? The old notation was fine for many purposes, but it turns out that the new notation will be a lot more convenient when we have to manipulate complex expressions. In fact, we will see cases later where it's easy to use the new notation to take derivatives, but it's not even possible to write the answer compactly in the old notation. One reason for the extra flexibility is that we have separated individual differentials like dx and dy, giving each one its own meaning, instead of giving a meaning only to a combined notation like $\frac{dy}{dx}$.

Working with differentials

One of the advantages of differential notation is that it lets us write derivatives of

complicated expressions cleanly and compactly. For example, we can mix scalar, vector, and matrix variables; we can avoid explicit manipulation of indices for vectors, matrices, and tensors; and we can work with high-dimensional derivatives without having to reshape them into standard shapes.

Just as with scalar derivatives, some of the most useful tools for working with differentials are linearity, the product rule, and the chain rule. Using these tools, we can often reduce a complex differential to a bunch of lookups of derivatives of standard functions. We'll look at each of these techniques in turn.

Standard functions

For scalar functions like $\cos$ and $\exp$, the differential is equivalent to the scalar derivative:

$$df(x) = f'(x)\,dx$$

So for example, $d\cos x = \sin x\,dx$, $d\exp x = \exp x\,dx$, and $dx^k = kx^{k-1}dx$.

In addition, there are identities for standard matrix and vector functions. For example, if A is a symmetric positive definite matrix, the differential of the log-determinant is

$$d\ln\det A = \langle A^{-1}, dA\rangle$$

where $\langle X, Y\rangle$ is the standard matrix inner product $\operatorname{tr}(X^TY)$. Note the similarity to the scalar logarithm: for scalar $x > 0$, $d\ln\det x = d\ln x = x^{-1}dx$.

For another example, if x is a vector, the differential of the vector norm is

$$d\|x\| = \frac{x}{\|x\|}dx \qquad x \neq 0$$

A final example is that the differential of a constant is zero. More specifically, it is the zero vector of matching type: if $x \in V$ is constant then $dx = 0 \in V$.

There are too many identities for standard functions to list here; a good reference is the Matrix Cookbook.

Linearity

The differential symbol d acts like a linear operator. So for example we can rewrite

$$d(af + bg) = a\,df + b\,dg$$

and we can rewrite

$$d(A^T X + 3Y) = A^T dX + 3\,dY$$

Combining linearity with identities for standard functions lets us go a long way: e.g.,

$$d(3x^7 + 5\sin x + \ln x) = 21x^6 dx + 5\cos x\,dx + x^{-1}dx \qquad x > 0$$

The product rule

The differential of a product is

$$d(fg) = df\,g + f\,dg$$

This is true for any *product-like operation*: e.g., scalar multiplication, matrix multiplication, Kronecker product, composition of linear functions, vector convolution, dot product, or componentwise multiplication of vectors. It's true even when the product is non-commutative, as for example with matrix multiplication. (In this case the order of the terms above matters.) And it extends to more than two terms: e.g.,

$$d(fgh) = df\,gh + f\,dg\,h + fg\,dh$$

In the scalar case the product rule only applies to ordinary multiplication; but for vector spaces there's a nearly infinite variety of product-like operations. For this reason, the generalized version of the product rule can be highly expressive and useful.

Prime notation

The scalar equality $df(x) = f'(x)dx$ is so useful that it's worth upgrading it to work more generally. It turns out that, whenever we have an equation like

$$df(x) = \text{expression involving } x \text{ and } dx$$

we can always factor out dx so that the equation takes the form

$$df(x) = [\text{linear operator that depends on } x]\,dx$$

The overall expression is nonlinear, since it can have a nonlinear dependence on x. But the differential dx always appears linearly.

For example, if $x = (u, v)^T \in \mathbb{R}^2$, and if $f(x) = u^2 + v^2$, then we can use the rules above (namely linearity of d and the identity for monomials) to show that

$$df(x) = 2u\,du + 2v\,dv = (2u\ 2v)\,dx$$

As claimed, we have a linear operator (multiplication by the vector $(2u\ 2v)$) applied to $dx = (du, dv)^T$.

This factorization holds no matter how complicated the type of f is: f could take a tuple of three matrices and a vector as input, and produce a tensor as output, and we would still be able to define a linear operator that relates dx to df. But depending on the types involved, the linear function $f'(x)$ could take different forms: a dot product, a matrix multiplication, or something else. We'll say more about the "something else" case below.

Given the above factorization, it makes sense to give a name to the linear operator: we will call it $f'(x)$, so that the equation becomes

$$df(x) = f'(x)\,dx$$

just as for the scalar case. So for example, if $f(u, v) = u^2 + v^2$ as above, then $f'(u, v) = (2u\ 2v)$.

> *You will sometimes find different conventions for f', such as transposing it or using special notation in some cases for specific kinds of linear operators. The convention we define here has the advantage of uniformity: the equation $df(x) = f'(x)dx$ always holds, and is always interpreted as application of a linear operator to dx.*

The chain rule

Given the new prime notation, the chain rule for differentials looks a lot like the chain rule we're used to for scalars:

$$d(f(g(x))) = f'(g(x))\,d(g(x))$$

For example, if we define a nonlinear function with a matrix argument like

$$f(U) = \ln\det U \qquad g(X) = A^T X A$$

then we can use the chain rule to get an expression for $d(f(g(X))$. (Here X and U are symmetric positive definite matrices.)

The first step is to separately differentiate f and g. For f, the identity for $\ln\det$ tells us that

$$df(U) = \langle U^{-1}, dU\rangle$$

making $f'(U)$ the linear operator that maps dU to $\langle U^{-1}, dU \rangle$.

For g, we can use the linearity of d (or the product rule) to show

$$dg(X) = A^T dX\, A$$

Putting these together, we can substitute $U = g(X)$ and apply the linear operator $f'(U)$ to $dg(X)$ to get

$$df(g(X)) = \langle (A^T X A)^{-1}, A^T dX\, A \rangle$$

Example: linear regression

We'll do this example in class.

With two or more variables

If we have multiple variables $x, y, \ldots$ then we can collect them together into a tuple u, so that an expression like $f(x, y, \ldots) = \ldots$ becomes $f(u) = \ldots$. Taking the differential, we have

$$df(u) = f'(u)\, du$$

We can then expand back to use the original variables $x, y, \ldots$. Focusing on the two-variable case for simplicity, we get

$$df(x, y) = f_x(x, y)\, dx + f_y(x, y)\, dy$$

That is, we can split the linear operator $f'(u)$ into two separate linear operators $f_x(u)$ and $f_y(u)$, one acting on dx only and one on dy only. Each of these functions represents the partial derivative of $f(x, y)$ with respect to one of its arguments. Just as in the single-variable case, we can *define* the functions f_x and f_y by the above equation.

There are two sets of variables here: the first set $x, y, \ldots$ is the place where we are evaluating the derivative, and the second set $dx, dy, \ldots$ is the change we're making in the first set. The overall expression can be nonlinear in the first set of arguments, but is always linear in the second set.

For example, if $f = \sin(x + y)$ then

$$df(x, y) = \cos(x + y)(dx + dy)$$

by the chain rule. This expression represents the Taylor approximation

$$f(x, y) - f(\hat{x}, \hat{y}) \approx \cos(x + y)((x - \hat{x}) + (y - \hat{y}))$$

A note on notation

When we have multiple variables of different types, expressions for differentials can get complicated, and it may not be straightforward to define a clear notation. One place where notation sometimes fails us is in representing an equation of the form $dy = f'(x)dx$. The value of $f'(x)$ is a linear function, so we have to have notation that lets us apply a linear function of the correct type.

If x and y have simple types, there is often a standard notation for linear function application. E.g., if x and y are real vectors, then $f'(x)$ returns a matrix, and we write linear function application as matrix multiplication.

On the other hand, suppose that our variables are matrices. In this case the output of f' is a linear function between matrices — often represented as a fourth-order tensor. There may be no simple expression for this tensor, even if the calculations involving differentials are not too bad.

In the worst case, we can always re-introduce explicit indices and work component by component. To ease index manipulation, we can sometimes use the Einstein summation convention; see `torch.einsum` for more details.

Example: batch norm

For example, consider the *batch normalization* operation that is often used in deep networks. There are two steps in batch normalization: first subtract the batch mean from every example in the batch, and second divide each example in the batch by the batch standard deviation. If we let the matrix X represent our batch, with one column per example in the batch, then we can write these steps as

$$Y = X - \mu(X)e^T$$

$$Z = (V(Y))^{-\frac{1}{2}} \circ Y$$

where e is a vector with all elements equal to 1, the functions μ and V compute row-wise mean and variance, and $\circ$ denotes the broadcasting product. The inverse square root of V is taken coordinatewise. If there are n vectors in our batch, then X will have n

columns, and e will be in $\mathbb{R}^n$.

We can write the mean and variance as

$$\mu(X) = \frac{1}{n} X e$$

$$V(Y) = \frac{1}{n} \mathrm{diag}(Y Y^T)$$

where diag extracts the diagonal of a matrix as a vector.

If we want to find dZ in terms of dY, we can first separately differentiate the individual steps in computing Z:

$$dV(Y) = \frac{1}{n} \mathrm{diag}(Y\, dY^T + dY\, Y^T)$$

$$d(v^{-\frac{1}{2}}) = -\frac{1}{2} v^{-\frac{3}{2}} \circ dv$$

$$dZ = \left(-\frac{1}{2} V(Y)^{-\frac{3}{2}} \circ dV(Y) \right) \circ Y + V(Y)^{-\frac{1}{2}} \circ dY$$

The first equation above uses linearity and the product rule. The second equation handles each coordinate separately with the identity for monomials, and puts the results together coordinatewise using $\circ$. The third uses the chain rule (with $dv^{-\frac{1}{2}}$) and the product rule.

Finally we can combine dZ and dV with the chain rule to get the final answer:

$$dZ = -\frac{1}{2n} V(Y)^{-\frac{3}{2}} \circ \mathrm{diag}(Y\, dY^T + dY\, Y^T) \circ Y + V(Y)^{-\frac{1}{2}} \circ dY$$

This does in fact represent a linear function of dY for each value of Y; but there's not an easy way to write this function using standard notation. We'd have to construct a fourth-order tensor that accomplishes the same thing as the above equation, which is somewhat tricky and error-prone.

On the other hand it's easy to continue working with the above equation. For example, we can calculate

$$dY = dX - d\mu(X) e^T = dX - \frac{1}{n} dX\, e e^T$$

and substitute in to get dZ in terms of dX.

Type checking

We advised earlier that it's generally a good idea to type-check your expressions to help catch bugs: that is, make sure you know the type of each sub-expression, and that every operation makes sense given the types of its arguments. Type-checking is particularly important with differentials, since there can be a lot of different vector spaces floating around.

For this purpose, it helps to be clear about the exact types of expressions involving differentials. Suppose we start from an expression that looks like

$$f = \dots x \dots$$

This equation describes how a variable $f \in V$ depends on another variable $x \in U$; in other words it defines a function in $U \to V$. We often call this function f or $f(x)$.

> *A source of confusion here is the distinction between the function itself (an element of $U \to V$) and its value (an element of V). You will unfortunately see both f and $f(x)$ used to refer to both of these quantities. When working with differential notation, it's probably easiest to use the convention that we omit all placeholder arguments in equations, and assume that all symbols refer to values: e.g., $f = x^2 + 3y$ and not $f(x, y) = x^2 + 3y$.*

The differential will then look like

$$df = f'(x)\, dx$$

where $x \in U$, $dx \in U$, and $df \in V$. (Here, $f'(x)$ typically won't appear verbatim; instead we'll get an expression that shows how to apply the linear function $f'(x)$ to dx, like $\langle (A^T X A)^{-1}, A^T dX\, A \rangle$ in the example above.)

The type of f' is $U \to (U \to V)$: given x it produces a function that maps $dx \in U$ to $df \in V$. The function f' can be (and often is) nonlinear in its argument x. But the output of f' has to be a linear function that acts on dx; for example, if $U = \mathbb{R}^n$ and $V = \mathbb{R}^m$, then f' returns a matrix $f'(x) \in \mathbb{R}^{m \times n}$.

Total vs. partial derivatives

One advantage of our new notation is that there's no need for separate treatment of total and partial derivatives ($\frac{d}{dx}$ vs. $\frac{\partial}{\partial x}$). Instead we can use a more flexible and expressive convention: all variables that appear inside a differential can change simultaneously and independently, and any other variables are held constant. (With the

$\frac{d}{dx}$ vs. $\frac{\partial}{\partial x}$ convention, it's hard to keep track of mixtures of variables that are either changing together or held constant.) So for example in the expression

$$dL = f(x, y, z)\, dx + g(x, y, z)\, dy$$

we are relating changes in L to changes in x and y while holding z fixed.

If x and y can't change independently, but instead depend on another variable t, we can include more equations:

$$x = \ell(t) \qquad y = m(t)$$

We can separately calculate differentials for each of these new equations:

$$dx = \ell'(t)dt \qquad dy = m'(t)dt$$

and then substitute them in if desired:

$$dL = f(\ell(t), m(t), z)\, \ell'(t)dt + g(\ell(t), m(t), z)\, m'(t)dt$$

(This substitution effectively implements the chain rule.) This way, we can implement any combination of variables that change independently or together, or are held constant.

If we want to go back to the old notation, we can manipulate to get an equation that only contains one differential on each side, and then "divide through" by one of them. E.g., if we "divide" the equation above by dt, we get the total derivative

$$\frac{dL}{dt} = f(\ell(t), m(t), z)\, \ell'(t) + g(\ell(t), m(t), z)\, m'(t)$$

As always, we aren't really dividing here, so the heuristic doesn't always work; but it's a good mnemonic.

> *What we're really doing is pulling out an expression for the linear function that acts on dt. The heuristic takes advantage of the fact that we often write application of a linear function like multiplication, but it fails when we can't do this.*