L4 Differential calculus review

Greg Ridgeway

2026-02-10

Table of contents

1 Properties of derivatives

The derivative measures the slope of the line tangent to the curve $f(x)$ at x. There are a lot of online derivative calculators that will grind out the derivatives should you need to compute one. Nevertheless, it will be valuable to have some of their basic properties on hand. We will use derivatives a lot when optimizing machine learning methods.

$$\frac{d}{dx}f(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

$$\frac{d}{dx}c = 0$$
$$\frac{d}{dx}x^n = nx^{n-1}$$
$$\frac{d}{dx}e^x = e^x$$
$$\frac{d}{dx}\log(x) = \frac{1}{x}$$
$$\frac{d}{dx}cf(x) = c\frac{d}{dx}f(x)$$
$$\frac{d}{dx}(f(x)+g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$
$$\frac{d}{dx}f(x)g(x) = f(x)\frac{d}{dx}g(x) + g(x)\frac{d}{dx}f(x) \text{ (Product rule)}$$
$$\frac{d}{dx}\frac{f(x)}{g(x)} = \frac{g(x)\frac{d}{dx}f(x) - f(x)\frac{d}{dx}g(x)}{g(x)^2} \text{ (Quotient rule)}$$
$$\frac{d}{dx}f(g(x)) = \frac{d}{dg}f(g) \times \frac{d}{dx}g(x) \text{ (Chain rule)}$$

1.1 Exercises

1. Example: Compute $\frac{d}{d\beta}(y-\beta)^2$

Answer: Use the chain rule where $f(g) = g^2$ and $g(\beta) = y - \beta$.

$$\frac{d}{d\beta}(y-\beta)^2 = 2(y-\beta)(-1)$$

2. Compute $\frac{d}{d\beta}((y_1-\beta)^2 + (y_2-\beta)^2 + (y_3-\beta)^2)$
3. Compute $\frac{d}{d\beta}\sum_{i=1}^n (y_i-\beta)^2$
4. Compute $\frac{d}{dp}\log(p) + \log(1-p)$
5. Compute $\frac{d}{dp}y\log(p) + (1-y)\log(1-p)$
6. Compute $\frac{d}{dx}|x|$ (Hint: draw a picture of $f(x) = |x|$)
7. Compute $\frac{d}{dx}|a+bx|$
8. Compute $\frac{d}{dx}\frac{1}{1+e^{-x}}$

2 Optimization

For smooth functions, $f(x)$, local minima/maxima solve

$$\frac{d}{dx}f(x) = 0$$

The second derivatives help us determine whether a local extreme value is a maximum or a minimum.

$$\frac{d^2}{dx^2} f(x_{\max}) < 0$$
$$\frac{d^2}{dx^2} f(x_{\min}) > 0$$

2.1 Quadratic

Find the minimum/maximum for the $f(x) = ax^2 + bx + c$.

$$\frac{d}{dx} f(x) = 2ax + b$$
$$2a\hat{x} + b = 0$$
$$\hat{x} = -\frac{b}{2a}$$

Is it a maximum or a minimum?

$$\frac{d^2}{dx^2} f(x) = 2a$$

It is a maximum if $a < 0$ and a minimum if $a > 0$.

2.2 Minimize squared differences with three values

Assume we have three numbers, x_1, x_2, x_3. What value of μ minimizes

$$J(\mu) = (x_1 - \mu)^2 + (x_2 - \mu)^2 + (x_3 - \mu)^2$$

$$J'(\mu) = 2(x_1 - \mu)(-1) + 2(x_2 - \mu)(-1) + 2(x_3 - \mu)(-1)$$
$$J'(\hat{\mu}) = 0$$
$$2(x_1 - \hat{\mu})(-1) + 2(x_2 - \hat{\mu})(-1) + 2(x_3 - \hat{\mu})(-1) = 0$$
$$(x_1 - \hat{\mu}) + (x_2 - \hat{\mu}) + (x_3 - \hat{\mu}) = 0$$
$$x_1 + x_2 + x_3 - 3\hat{\mu} = 0$$
$$\hat{\mu} = \frac{x_1 + x_2 + x_3}{3}$$

2.3 Minimize squared differences with n values

Assume we have numbers, $x_1, \ldots, x_n$. What value or μ minimizes

$$J(\mu) = \sum_{i=1}^{n} (x_i - \mu)^2$$

$$\begin{aligned}
\frac{d}{d\mu} J(\mu) &= \frac{d}{d\mu} \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= \sum_{i=1}^{n} \frac{d}{d\mu} \left((x_i - \mu)^2 \right) \\
&= \sum_{i=1}^{n} 2(x_i - \mu)(-1) \\
\frac{d}{d\mu} J(\mu) \Big|_{\hat{\mu}} &= 0 \\
\sum_{i=1}^{n} 2(x_i - \hat{\mu})(-1) &= 0 \\
-2 \sum_{i=1}^{n} (x_i - \hat{\mu}) &= 0 \\
\sum_{i=1}^{n} x_i - n\hat{\mu} &= 0 \\
\hat{\mu} &= \frac{\sum_{i=1}^{n} x_i}{n}
\end{aligned}$$

2.4 Estimating a probability

We have random variables $Y_1, \ldots, Y_n$ that are all either 0 or 1. They are all independent and have the same probability of equaling 1, $P(Y_i = 1) = p$. These are commonly called Bernoulli random variables. The probability function for Bernoulli random variable is $P(Y = y) = p^y (1-p)^{1-y}$.

If p is fixed but unknown, what is the probability of observing the sequence $y_1, \ldots, y_n$? This is known as the likelihood function for p.

$$\begin{aligned}
L(p) &= P(Y_1 = y_1, \ldots, Y_n = y_n) \\
&= P(Y_1 = y_1) \cdots P(Y_n = y_n) \\
&= p^{y_1} (1-p)^{1-y_1} \cdots p^{y_n} (1-p)^{1-y_n} \\
&= p^{y_1 + \ldots + y_n} (1-p)^{1-y_1+\ldots+1-y_n} \\
&= p^{\sum y_i} (1-p)^{n - \sum y_i}
\end{aligned}$$

This function is a little difficult to optimize. Conveniently, if $\hat{p}$ optimizes $L(p)$ then $\hat{p}$ will also optimize $\log L(p) = \ell(p)$.

$$\begin{aligned}
\ell(p) &= \log L(p) \\
&= \log\left(p^{\sum y_i}(1-p)^{n-\sum y_i}\right) \\
&= \left(\sum y_i\right)\log(p) + \left(n - \sum y_i\right)\log(1-p) \\
\ell'(p) &= \left(\sum y_i\right)\frac{1}{p} - \left(n - \sum y_i\right)\frac{1}{1-p}
\end{aligned}$$

Find $\hat{p}$ that solves $\ell'(p) = 0$.

$$\begin{aligned}
0 &= \ell'(\hat{p}) \\
&= \left(\sum y_i\right)\frac{1}{\hat{p}} - \left(n - \sum y_i\right)\frac{1}{1-\hat{p}} \\
&= \left(\sum y_i\right)(1-\hat{p}) - \left(n - \sum y_i\right)\hat{p} \\
&= \left(\sum y_i\right) - \hat{p}\left(\sum y_i\right) - n\hat{p} + \left(\sum y_i\right)\hat{p} \\
\hat{p} &= \frac{\sum y_i}{n}
\end{aligned}$$

This shows that the maximum likelihood estimator for p is the sample mean of the y_is, the same as the commonsense estimate that you would have used.

2.5 Exercises

1. Find $\hat{x}$ that optimizes $f(x) = 2x + \frac{200}{x}$. Is it a maximum or a minimum?
2. Find $\hat{x}$ that optimizes $f(x) = \log(x) + \log(1-x) + 3$. Is it a maximum or a minimum?
3. Find $\hat{t}$ that optimizes $f(x) = 100 + \frac{800t}{t^2+90000}$. Is it a maximum or a minimum?
4. Find $\hat{p}$ that optimizes $\ell(p) = 6\log(p) + 10\log(1-p)$. Is it a maximum or a minimum?

3 Gradient descent

Often we will not be able to directly solve $\frac{d}{dx}f(x) = 0$. The best we can do is figure out how to move a guess for the optimal value of x toward the optimal value. Let's simulate some data where x are a bunch of random standard normal values and $y = \frac{1}{1+\exp(-3x)}$ and we will let J be squared error.

```
x <- rnorm(100)
y <- 1/(1+exp(-3*x))

# squared error
```

```r
J <- function(b, x, y)
{
  yhat <- 1/(1+exp(-b*x))
  return( sum((y-yhat)^2) )
}

# derivative of J with respect to b
dJ <- function(b, x, y)
{
  yhat <- 1/(1+exp(-b*x))
  return( sum(-2*(y-yhat)*yhat*(1-yhat)*x) )
}


b <- seq(0,20, length.out=100)
J0 <- sapply(b, function(beta0) J(beta0,x,y))

plot(b,J0, type="l", xlab=expression(hat(beta)), ylab=expression(J(hat(beta))))
points(c(1,6),c(J(1,x,y),J(6,x,y)))
curve(dJ(1,x,y)*(b-1) + J(1,x,y), add = TRUE, col = 'blue', xname="b",
      from=0,to=1.6)
curve(dJ(6,x,y)*(b-6) + J(6,x,y), add = TRUE, col = 'blue', xname="b",
      from=2,to=10)
```

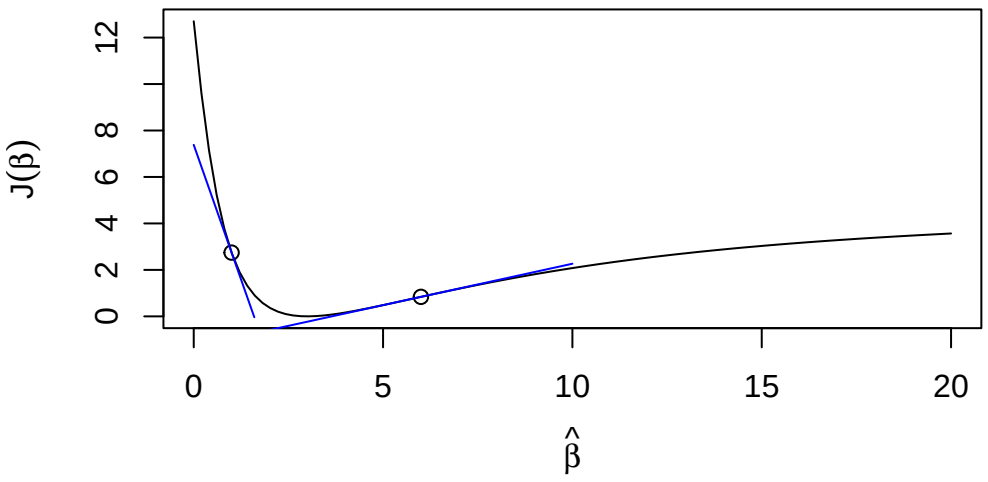

Figure 1: An example $J(\beta)$ showing tangent lines at two points

Consider Figure 1. Here we have a loss function that has its minimum around $\hat{\beta} = 3$ (as it should since we simulated it to be so). If we evaluate the derivative at $\hat{\beta} = 6$, then we get a positive slope, telling us that the optimal β is smaller than 6. If we evaluate the derivative at $\hat{\beta} = 1$, then we get a negative slope, telling us that the optimal $\hat{\beta}$ is larger.

This suggests an optimization process that iteratively adjusts our guess for the optimal $\hat{\beta}$ as

$$\hat{\beta} \leftarrow \hat{\beta} - \lambda J'(\hat{\beta})$$

where λ is a small value representing a "step size" or a "learning rate".

3.1 Taylor series and Newton's method

For any well-behaved function (smooth, with derivatives), Taylor series approximations can come in handy.

Theorem 3.1 (Taylor's Theorem)**.**

$$J(x) = J(x_0) + J'(x_0)(x - x_0) + \frac{1}{2}J''(x_0)(x - x_0)^2 + ... + \frac{1}{n!}J^{(n)}(x_0)(x - x_0)^n + ...$$

Typically, we will just use the series up to the quadratic term to approximate Js that are hard to analyze otherwise. Let's say that we have a $J(x)$ that we want to optimize, but our algebra tricks do not help us figure out its optimal value. A quadratic Taylor series approximation is

$$J(x) \approx J(x_0) + J'(x_0)(x - x_0) + \frac{1}{2}J''(x_0)(x - x_0)^2$$

If we have a starting value, x_0, then we can optimize the right-hand side to find a new candidate for the optimal value. If we take a derivative with respect to x (not x_0!) we arrive at

$$J'(x) \approx J'(x_0) + J''(x_0)(x - x_0)$$

Now let's find an x that optimizes this expression.

$$\begin{aligned} J'(x_0) + J''(x_0)(\hat{x} - x_0) &= 0 \\ \hat{x} &= x_0 - \frac{J'(x_0)}{J''(x_0)} \end{aligned} \tag{1}$$

This implies that if we have a guess, x_0, for the optimal value, then we should be able to get a better guess by calculating (1). We will need to repeat this several times until it converges on a final answer. This is known as Newton's method.

Example In a previous exercise we optimized $\ell(p) = 6\log(p) + 10\log(1-p)$ directly. Let's pretend it was too hard and try Newton's method on this one instead.

$$\begin{aligned} \ell(p) &= 6\log(p) + 10\log(1-p) \\ \ell'(p) &= \frac{6}{p} - \frac{10}{1-p} \\ \ell''(p) &= -\frac{6}{p^2} - \frac{10}{(1-p)^2} \\ \hat{p} &= p_0 - \frac{\ell'(p_0)}{\ell''(p_0)} \\ &= p_0 - \frac{\frac{6}{p_0} - \frac{10}{1-p_0}}{-\frac{6}{p_0^2} - \frac{10}{(1-p_0)^2}} \\ &= p_0 + \frac{p_0(1-p_0)(3-8p_0)}{3-6p_0+8p_0^2} \end{aligned}$$

There are plenty of examples where Newton's method does not work well (slow to converge) or at all (diverges or converges to a local maximum/minimum). For example, if we started with $p_0 = 0$ or $p_0 = 1$, then for this problem $\hat{p}$ would get stuck at 0 or 1 and never move to a better value. Also note that if we started at $p_0 = \frac{3}{8}$ then it also does not change, but that is because $\frac{3}{8}$ is the right answer!

Running this in R shows that it converges to the exact answer in just a few steps

```r
p <- 0.1
# newton iterations
for(i in 1:6)
{
  p <- p + p*(1-p)*(3-8*p)/(3-6*p+8*p^2)
  print(p)
}
```

```
[1] 0.1798387
[1] 0.2854885
[1] 0.3608183
[1] 0.3747613
[1] 0.3749999
[1] 0.375
```

R has built-in optimization functions that you can use for any similar optimization problem. You just need to specify the function, the first derivative (also called the gradient), and the second derivative (also called the Hessian), and then pass those functions to `nlm()`. Note that `nlm()` only solves *minimization* problems. So, if you have a maximization problem, just multiply all your functions and derivatives by -1 to make it a minimization problem.

```r
f <- function(p)
{
  res <- -(6*log(p)+10*log(1-p))
  attr(res, "gradient") <- -(6/p - 10/(1-p))
  attr(res, "hessian")  <- 6/p^2 + 10/(1-p)^2
  return(res)
}
nlm(f, 0.1)
```

```
$minimum
[1] 10.58501

$estimate
[1] 0.3749999

$gradient
[1] -4.155613e-06

$code
[1] 1
```

```
$iterations
[1] 5
```

3.2 Derivatives of multivariate functions

Machine learning models never have a single parameter. The latest ones in the news have trillions of parameters. We will end up optimizing functions of many parameters, so we need some tools to compute derivatives of functions with many parameters.

If $J(\alpha, \beta)$ then there are two *partial* derivatives

$$\frac{\partial}{\partial \alpha} J(\alpha, \beta) = \left.\frac{dJ}{d\alpha}\right|_{\beta}$$
$$\frac{\partial}{\partial \beta} J(\alpha, \beta) = \left.\frac{dJ}{d\beta}\right|_{\alpha}$$

In each of these, the parameter that is not the focus of the derivative is treated as a fixed constant.

We will encounter models with many parameters with derivatives requiring a rather deep level of using the chain rule, especially when we get to backpropagation in neural networks. For example,

$$J(\beta_0, \beta_1, \ldots, \beta_d) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
$$\hat{y}_i = \frac{1}{1 + e^{-z_i}}$$
$$z_i = \beta_0 + \beta_1 x_{1i} + \ldots + \beta_d x_{di}$$

To figure out the "direction" to move β in order to optimize J we need to work through several

levels of derivatives.

$$\begin{aligned}
\frac{\partial J}{\partial \hat{y}_i} &= -2(y_i - \hat{y}_i) \\
\frac{\partial \hat{y}_i}{\partial z_i} &= \frac{e^{-z_i}}{(1+e^{-z_i})^2} \\
&= \frac{1}{1+e^{-z_i}} \frac{e^{-z_i}}{1+e^{-z_i}} \\
&= \frac{1}{1+e^{-z_i}} \frac{1+e^{-z_i}-1}{1+e^{-z_i}} \\
&= \frac{1}{1+e^{-z_i}} \left(\frac{1+e^{-z_i}}{1+e^{-z_i}} - \frac{1}{1+e^{-z_i}} \right) \\
&= \frac{1}{1+e^{-z_i}} \left(1 - \frac{1}{1+e^{-z_i}} \right) \\
&= \hat{y}_i(1-\hat{y}_i) \\
\frac{\partial z_i}{\partial \beta_j} &= x_{ji}
\end{aligned}$$

If we want to know how changes in a particular β_j change the value of J then we chain all the derivatives together.

$$\begin{aligned}
\frac{\partial J}{\partial \beta_j} &= \frac{\partial J}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial \beta_j} + ... + \frac{\partial J}{\partial \hat{y}_n} \frac{\partial \hat{y}_n}{\partial z_n} \frac{\partial z_n}{\partial \beta_j} \\
&= \sum_{i=1}^{n} -2(y_i - \hat{y}_i)\hat{y}_i(1-\hat{y}_i)x_{ji}
\end{aligned}$$

The gradient is the collection of derivatives assembled into a single vector and is usually denoted as ∇J (spoken as "gradient of J" or "del J").

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \vdots \\ \frac{\partial J}{\partial \beta_d} \end{bmatrix}$$

3.3 Multivariate gradient descent

The gradient "points" in the direction that we need to move our current choice of β to optimize $J(\beta)$. That is, if we have a current guess for $\hat{\beta}$, then we can find an improved guess (one that makes $J(\beta)$ even smaller), by adjusting our guess as

$$\begin{aligned}
\hat{\beta} &\leftarrow \hat{\beta} - \lambda \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \vdots \\ \frac{\partial J}{\partial \beta_d} \end{bmatrix} \\
&= \hat{\beta} - \lambda \nabla J
\end{aligned}$$

for some λ that is sufficiently small.

We will get more into this when we pick up some linear algebra skills and matrix derivatives.

4 Summary

We have complete a quick review of differential calculus concepts essential for understanding optimization and machine learning algorithms.

1. **Properties of Derivatives**:
    - Important derivative rules (power rule, product rule, quotient rule, chain rule)
    - Analyzed and optimized functions
2. **Optimization**:
    - Found local maxima and minima using first and second derivatives
    - Solved common optimization problems (minimizing squared differences, estimating probabilities for Bernoulli random variables)
3. **Gradient Descent**:
    - Reviewed Taylor series approximations for analyzing complex functions
    - Used gradient descent as an iterative optimization method for finding minima when direct solutions are infeasible
4. **Multivariate Gradient Descent**:
    - Extended calculus principles to multivariate functions, introducing partial derivatives and gradients
    - Introduced gradient descent for multivariate functions

One important innovation in recent decades is efficient "auto-differentiation" algorithms, "autodiff" for short. The methods can take computer code that evaluates a model's performance and automatically write additional code to compute the gradients. These autodiff methods have become essential in scientific computing (such as Hamiltonian Markov Chain integration) and in learning neural networks. Common neural network software, like TensorFlow and PyTorch, rely on autodiff methods to efficiently compute gradients necessary for optimizing neural networks. Advances have been particularly large on using GPUs (graphical processing units) to run autodiff methods.