Derivatives of Random Functions

(MIT 18.S096, Lecture 7)

Gaurav Arya

February 1, 2023

1 Introduction

In this class, we've learned how to take derivatives of all sorts of crazy functions. Recall one of our first examples:

$$f(A) = A^2, \tag{1}$$

where A is a matrix. To differentiate this function, we had to go back to the drawing board, and ask:

Q1: If we perturb the input slightly, how does the output change?

To this end, we wrote down something like:

$$\mathrm{d}f = (A + \mathrm{d}A)^2 - A^2 = A(\mathrm{d}A) + (\mathrm{d}A)A + \underbrace{(\mathrm{d}A)^2}_{\text{neglected}}. \tag{2}$$

We called $\mathrm{d}f$ and $\mathrm{d}A$ *differentials*. We then had to ask:

Q2: What terms in the differential can we neglect?

We decided that $(\mathrm{d}A)^2$ should be neglected[1], justifying this by the fact that $(\mathrm{d}A)^2$ is "higher-order". We were left with the derivative operator $\mathrm{d}A \mapsto A(\mathrm{d}A) + (\mathrm{d}A)A$: the best possible *linear* approximation to f in a neighbourhood of A.

At a high level, the main challenge here was dealing with a complicated input and output space: f was matrix-valued, and also matrix-accepting. We had to ask ourselves: in this case, what should the notion of a derivative even mean?

In this class, we will face a similar challenge, but with a different output space. This time, the output of our function will be *random*. Now, we need to revisit the same questions. If the output is random, how can we describe its response to a change in the input? And how can we form a useful notion of derivative?

[1]I am admittedly being a bit unclear about what a differential means here, and what it means to neglect a term. In one perspective, we can imagine that $\mathrm{d}A$ is infinitesimally small so that $(\mathrm{d}A)^2 = 0$. Alternatively, we can imagine $\mathrm{d}A$ representing a finite quantity (what might be called ΔA in class), and define the derivative, which is what we really care about, in the limit of $|\mathrm{d}A| \to 0$: this perspective might work better with how I define the differential for a stochastic program later.

2 Stochastic programs

More precisely, we will consider random, or *stochastic*, functions X with real input $p \in \mathbb{R}$ and real-valued random variable output. As a map, we can write X as

$$p \mapsto X(p), \tag{3}$$

where $X(p)$ is a random variable.

The idea is that we can only *sample* from $X(p)$. For example, suppose $X(p)$ follows the exponential distribution with scale p, i.e. $X(p) \sim \mathrm{Exp}(p)$.

```
julia> using Distributions

julia> sample_X(p) = rand(Exponential(p))
X (generic function with 1 method)

julia> sample_X(10.0)
1.7849785709142214

julia> sample_X(10.0)
4.435847397169775

julia> sample_X(10.0)
0.6823343897949835
```

If our program gives a different output each time, what could a useful notion of derivative be? Before we try to answer this, let's ask *why* we might want to take a derivative. The answer is that we may be very interested in *statistical quantities* of random functions, i.e. values that can be expressed using an average. Furthermore:

It is often much easier to produce an "unbiased estimate" of a stochastic quantity than to compute it exactly.

Here, an unbiased estimate means that $X(p)$ averages out to our statitistical quantity of interest.

For example, in deep learning, the variational autoencoder (VAE) [1] is a very common architecture that is inherently stochastic. It is easy to get a stochastic *unbiased estimate* of the loss function by random simulation: if $X(p)$ represents this estimate, then the loss function $L(p)$ is $\mathbb{E}[X(p)]$. However, computing the loss $L(p)$ exactly would require integrating over all possible outcomes. Now, to train the VAE, we also need to differentiate $L(p)$, i.e. differentiate $\mathbb{E}[X(p)]$!

Perhaps an even more compelling example is in the physical sciences, where randomness may be baked into your model of a physical process. In this case, it's hard to get around the fact that you need to deal with stochasticity! For example, you may have two particles that interact with an *average* rate of r. But in reality, the times when these interactions actually occur follow a stochastic process. (In fact, the time until the first reaction would be exponentially distributed, with scale $1/r$.) And if you want to e.g. fit the parameters of your stochastic model to real-world data, it's once again very useful to have derivatives.

If we can't compute our statistical quantity of interest exactly, it seems unreasonable to assume we can compute its derivative exactly. However, we could hope to stochastically estimate its derivative. So, if $X(p)$ represents the full program that produces an unbiased estimate of our statistical quantity, here's one property we'd definitely like our derivative to have: we should be able to construct from it an unbiased gradient estimator [2] $\tilde{X}(p)$ satisfying

$$\mathbb{E}[\tilde{X}(p)] = \frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}. \tag{4}$$

3 Stochastic differentials and the reparameterization trick

Let's begin by answering our first question: how does $X(p)$ respond to a change in p? Let us consider a specific p and write down the *stochastic differential*[2]:

$$\mathrm{d}X(\varepsilon) = X(p+\varepsilon) - X(p), \tag{5}$$

where ε represents a small change in p, which we use instead of $\mathrm{d}p$. What sort of object is $\mathrm{d}X(\varepsilon)$?

Since we're subtracting two random variables, it ought to itself be a random variable. However, $\mathrm{d}X(\varepsilon)$ is still not fully specified! We have only specified the marginal distributions of $X(p)$ and $X(p+\varepsilon)$: to be able to subtract the two, we need to know their *joint distribution*.

One possibility is to treat $X(p)$ and $X(p+\varepsilon)$ as independent. This means that $\mathrm{d}X(\varepsilon)$ would be constructed as the difference of independent samples. Let's see how samples from $\mathrm{d}X(\varepsilon)$ would look like in this case!

```
julia> sample_X(p) = rand(Exponential(p))
sample_X (generic function with 1 method)

julia> sample_dX(ε) = sample_X(p + ε) - sample_X(p)
sample_dX (generic function with 1 method)

julia> p = 10; ε = 1e-5;

julia> sample_dX(ε)
-26.000938718875904

julia> sample_dX(ε)
-2.6157162001718092

julia> sample_dX(ε)
6.352622554495474

julia> sample_dX(ε)
-9.53215951927184

julia> sample_dX(ε)
1.2232268930932104
```

We can observe something a bit worrying: even for a very tiny ε (we chose $\varepsilon = 10^{-5}$), $\mathrm{d}X(\varepsilon)$ is still fairly large: essentially as large as the original random variables. This is

[2]unlike before, I've introduced an explicit dependence on ε in $\mathrm{d}X$, to avoid thinking about infinitesimals. Thus, the stochastic differential is a "differential" in the sense of being a conventional difference, which may be a non-standard use.

not good news if we want to construct a derivative from $\mathrm{d}X(\varepsilon)$: we'd want its magnitude to be getting smaller and smaller with ε, like in the usual case.

Let's try a different approach. It is natural to think of $X(p)$ for all p as forming a *family* of random variables, all defined on the same *probability space*. A probability space, with some simplification, is a sample space Ω, with a probability distribution $\mathbb{P}$ defined on the sample space. On this probability space, each $X(p)$ can be expressed as a function $\Omega \to \mathbb{R}$. Intuitively, all of the "randomness" resides in the probability space, and crucially $\mathbb{P}$ does not depend on p: as p varies, $X(p)$ just becomes a different *deterministic* map on this space. To sample from a particular $X(p)$, we can imagine drawing a random ω from Ω according to $\mathbb{P}$, and then plugging this in to $X(p)$, i.e. computing $X(p)(\omega)$.

The crux here is that all the $X(p)$ now depend on a shared source of randomness: the random draw of ω. This means that $X(p)$ and $X(p+\varepsilon)$ have a non-trivial joint distribution: what does it look like?

For concreteness, let's study an exponential random variable $X(p) \sim \mathrm{Exp}(p)$, Using the "inversion method" parameterization[3], it is possible to choose Ω to be $[0,1]$ and $\mathbb{P}$ to be the uniform distribution over $\Omega = [0,1]$, and make $X(p)$ an increasing function over Ω. Applied to our example $X(p) \sim \mathrm{Exp}(p)$ from before, the inversion method gives $X(p)(\omega) = -p \cdot \log(1-\omega)$, i.e. the below is an equivalent way of sampling $X(p)$:

```
julia> sample_X2(p, ω) = -p * log(1 - ω)
sample_X2 (generic function with 1 method)

julia> # rand() samples a uniform random number in [0,1]
julia> sample_X2(p) = sample_X2(p, rand())
sample_X2 (generic function with 2 methods)

julia> sample_X2(10.0)
8.380816941818618

julia> sample_X2(10.0)
2.073939134369733

julia> sample_X2(10.0)
29.94586208847568

julia> sample_X2(10.0)
23.91658360124792
```

Okay, so what does our joint distribution look like?

[3]This certainly isn't the only way to choose Ω and $\mathbb{P}$, and in fact it's probably not the simplest choice for exponential random variables, but we'll stick to it for consistency with the discrete case later.

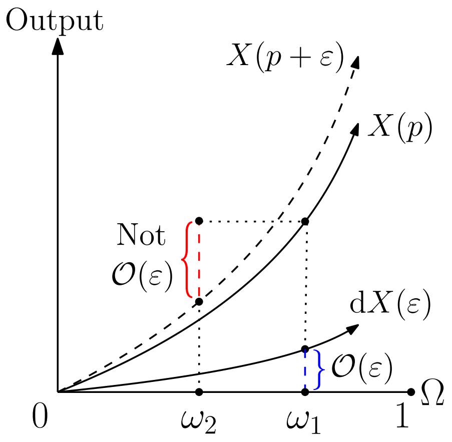
Output
$X(p+\varepsilon)$
$X(p)$
Not $\mathcal{O}(\varepsilon)$
$\mathrm{d}X(\varepsilon)$
$\mathcal{O}(\varepsilon)$
Ω
0
ω_2
ω_1
1

Figure 1: For $X(p) \sim \mathrm{Exp}(p)$ parameterized via the inversion method, we can write $X(p)$, $X(p+\varepsilon)$, and $\mathrm{d}X(\varepsilon)$ as functions from $\Omega = [0,1] \to \mathbb{R}$, defined on a probability space with $\mathbb{P} = \mathrm{Unif}(0,1)$.

As shown in Figure 1, we can plot $X(p)$ and $X(p+\varepsilon)$ as functions over Ω. To sample the two of them jointly, we use the *same* choice of ω: thus, $\mathrm{d}X(\varepsilon)$ can be formed by subtracting the two functions *pointwise* at each Ω. Ultimately, $\mathrm{d}X(\varepsilon)$ is itself a random variable over the same probability space, sampled in the same way: we pick a random ω according to $\mathbb{P}$, and evaluate $\mathrm{d}X(\varepsilon)(\omega)$, using the function $\mathrm{d}X(\varepsilon)$ depicted above. Our first approach with independent samples is depicted in red in Figure 1, while our second approach is in blue. We can now see the flaw of the independent samples approach: the $\mathcal{O}(1)$-sized "noise" from the independent samples washes out the $\mathcal{O}(\varepsilon)$-sized "signal".

What about question 2: how can actually take the limit of $\varepsilon \to 0$ and compute the derivative? The idea is to differentiate $\mathrm{d}X(\varepsilon)$ at each fixed sample $\omega \in \Omega$. In probability theory terms, we take the limit of random variables $\mathrm{d}X(\varepsilon)/\varepsilon$ as $\varepsilon \to 0$:

$$\delta = \lim_{\varepsilon \to 0} \frac{\mathrm{d}X(\varepsilon)}{\varepsilon}. \tag{6}$$

For $X(p) \sim \mathrm{Exp}(p)$ parameterized via the inversion method, we get:

$$\delta(\omega) = \lim_{\varepsilon \to 0} \frac{-\varepsilon \log(1-\omega)}{\varepsilon} = -\log(1-\omega). \tag{7}$$

Once again, δ is a random variable over the same probability space. The claim is that δ is the notion of derivative we were looking for! Indeed, δ is itself in fact a valid gradient estimator:

$$\mathbb{E}[\delta] = \mathbb{E}\left[\lim_{\varepsilon \to 0} \frac{\mathrm{d}X(\varepsilon)}{\varepsilon}\right] \overset{?}{=} \lim_{\varepsilon \to 0} \frac{\mathbb{E}[\mathrm{d}X(\varepsilon)]}{\varepsilon} = \frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}. \tag{8}$$

Rigorously, one needs to justify the interchange of limit and expectation in the above. Here, we will be content with empirical justification!

```julia
julia> delta(ω) = -log(1 - ω)
delta (generic function with 1 method)

julia> delta() = delta(rand())
delta (generic function with 2 methods)

julia> mean(delta() for i in 1:10000)
1.011689946421105
```

So δ does indeed average to 1, which makes sense since the expectation of $\text{Exp}(p)$ is p, which has derivative 1. However, the number 1 also averages to 1, so that's not very impressive by itself. The crux is that this notion of derivative does not lose any information about how the full random variable $X(p)$ reacts to changes in p. This means it can be composed to form derivative estimators of more complicated functions. In fact, it turns out to obey the same chain rule as usual! In contrast, if our notion of derivative was simply $\frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p} = 1$, we would not be able to have a chain rule that enables differentiation of more complicated functions.

Using dual numbers introduced in lecture 5, we can differentiate the expectation of the square of a sample from an exponential distribution, without having an analytic expression for this quantity. (The expression for δ we derived is already implemented as a dual number rule in Julia.) The primal and dual values of the outputted dual number is a sample from the joint distribution of $(X(p), \delta)$.

```
julia> using Distributions, ForwardDiff: Dual

julia> sample_X(p) = rand(Exponential(p))^2
sample_X (generic function with 1 method)

julia> sample_X(Dual(10.0, 1.0)) # sample a single dual number!
Dual}(153.74964559529033,30.749929119058066)

julia> # obtain the derivative!
julia> mean(sample_X(Dual(10.0, 1.0)).partials[1] for i in 1:10000)
40.016569793650525
```

Using the "reparameterization trick" to form a gradient estimator, as we have done, is a fairly old idea. It is also called the pathwise gradient estimator. Recently, it has become very popular in machine learning due to its use in VAEs [1], and lots of resources can be found online on it. Since composition simply works by the usual chain rule, it also works in reverse-mode, and can differentiate functions far more complicated than the one above! Our treatment here is arguably much less concise than it could be, but emphasizes some key conceptual points and sets the stage for understanding what goes wrong with discreteness, which is what we turn to next.

4 Handling discrete randomness

So far we have only considered a continuous random variable. Let's see how the picture changes for a discrete random variable! Let's take a simple Bernoulli variable $X(p) \sim \text{Ber}(p)$, which is 1 with probability p and 0 with probability $1 - p$.

```
julia> sample_X(p) = rand(Bernoulli(p))
sample_X (generic function with 1 method)

julia> p = 0.5
0.6

julia> sample_X(ε) # produces false/true, equivalent to 0/1
true
```

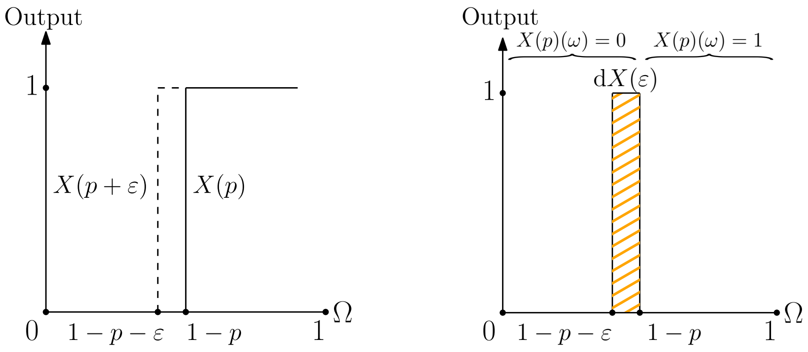

Figure 2: For $X(p) \sim \mathrm{Ber}(p)$ parameterized via the inversion method, plots of $X(p)$, $X(p+\varepsilon)$, and $\mathrm{d}X(\varepsilon)$ as functions $\Omega : [0,1] \to \mathbb{R}$.

```
julia> sample_X(ε)
false

julia> sample_X(ε)
true
```

The parameterization of a Bernoulli variable is shown in Figure 2. Using the inversion method once again, the parameterization of a Bernoulli variable looks like a step function: for $\omega < 1-p$, $X(p)(\omega) = 0$, while for $\omega \geq 1-p$, $X(p)(\omega)$.

Now, what happens when we perturb p? Let's imagine perturbing p by a positive amount ε. As shown in Figure 2, something qualitatively very different has happened here. At nearly every ω except a small region of probability ε, the output does not change. Thus, the δ we defined in the previous section (which, strictly speaking, was defined by an "almost-sure" limit that neglects regions of probability 0) is 0 at every ω: after all, for every ω, there exists small enough ε such that $\mathrm{d}X(\varepsilon)(\omega) = 0$.

However, there is certainly an important derivative contribution to consider here. The expectation of a Bernoulli is p, so we would expect the derivative to be 1: but $\mathbb{E}[\delta] = \mathbb{E}[0] = 0$.

What has gone wrong is that, although $\mathrm{d}X(\varepsilon)$ is 0 with tiny probability, the value of $\mathrm{d}X(\varepsilon)$ on this region of tiny probability is 1, which is *large*. In particular, it does not approach 0 as ε approaches 0. Thus, to develop a notion of derivative of $X(p)$, we need to somehow capture these large jumps with "infinitesimal" probability.

In recent work together with Frank Schäfer, Moritz Schauer, and Chris Rackauckas, [3], we've worked to extend the ideas I've talked about in this class to develop a notion of "stochastic derivative", and made a package called `StochasticAD.jl` which performs automatic differentiation by generalizing the idea of dual numbers to stochastic *triples*, which include a third component exactly for storing these large jumps. For example, the stochastic triple of a Bernoulli variable might say

```
julia> using StochasticAD, Distributions
julia> f(p) = rand(Bernoulli(p)) # 1 with probability p, 0 otherwise
julia> stochastic_triple(f, 0.5) # Feeds 0.5 + ε into f
StochasticTriple of Int64:
0 + 0ε + (1 with probability 2.0ε)
```

Here, ε is imagined to be an "infinitesimal unit", so that the above triple indicates a flip from 0 to 1 with probability that has derivative 2.

However, this problem remains a very difficult one, and there are a lot of improvements still to be made! If you're interested in reading more, you may be interested in our paper [3] and our package, as well as [2] which is a great survey of the field of gradient estimation in general.

At the end of class, we considered a random walk example with `StochasticAD.jl`. Here it is!

```
julia> using Distributions, StochasticAD

julia> function f(p)
           n = 0
           for i in 1:100
               n += rand(Bernoulli(p * (1 - (n+i)/200)))
           end
           return n
       end
f (generic function with 1 method)

julia> mean(f(0.5) for _ in 1:10000) # calculate mean at p = 0.5
32.6956

julia> st = stochastic_triple(f, 0.5) # sample a single stochastic triple at p = 0.5
StochasticTriple of Int64:
32 + 0ε + (1 with probability 74.17635818221052ε)

julia> derivative_contribution(st) # derivative estimate produced by the triple
74.17635818221052

julia> # obtain the derivative!
julia> mean(derivative_contribution(stochastic_triple(f, 0.5)) for i in 1:10000)
56.65142976168479
```

References

[1] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[2] Shakir Mohamed et al. "Monte carlo gradient estimation in machine learning". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 5183–5244.

[3] Gaurav Arya et al. "Automatic Differentiation of Programs with Discrete Randomness". In: *Thirty-Sixth Conference on Neural Information Processing Systems*. 2022. DOI: 10.48550/arXiv.2210.08572.