CS475 Fall'24 HW1: Understanding Backpropagation

Sheikh Shafayat

September 11, 2024

Introduction

As you probably learned in the class, the core problem that a neural network is trying to solve is an optimization problem: that is, we randomly initialize a neural network and then for each batch of data samples, calculate the gradient with respect to the parameters of the neural network, and then update the parameters according to the gradient. We try to find a good way to adjust the parameters of a neural network to minimize the loss function. The big question here is how exactly we calculate the gradient and the answer is an elegant algorithm called backpropagation.

Backpropagation

Backpropagation is the central algorithm that powers all modern deep-learning networks. It was invented several times independently by different groups of scientists working in completely unrelated fields from the 1960s to the 1980s. However, it was Geoffrey Hinton who popularized the idea of backpropagation to train deep neural networks during the 80s.

The core problem that backpropagation solves is how to take partial derivatives of very large and complicated functions with respect to their parameters.

We learned to do differentiation before. Let's start with some simple differentiation exercises:

- Take $y = wx + b$. What is $\frac{\partial y}{\partial w}$ and $\frac{\partial y}{\partial b}$?
- It's very easy: $\frac{\partial y}{\partial w} = x$ and $\frac{\partial y}{\partial b} = 1$.

Now, take a look at this one: $L(x, w, b) = [\sigma(wx + b) - y]^2$. Here σ is the nonlinearity function sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$.

This function looks like a linear logistic classifier that is being trained with L2 loss, where x is the data and y is the label provided.

What is $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$? For now, you can write the derivative of sigmoid as σ'.

$$\begin{aligned}\frac{\partial L}{\partial w} &= 2[\sigma(wx+b)-y]\frac{\partial}{\partial w}(\sigma(wx+b)-y)\\ &= 2[\sigma(wx+b)-y][\sigma'(wx+b)x]\end{aligned}$$

Task 0 (2 pt)

Derive the expression of $\frac{\partial L}{\partial b}$ similar to above. Show the intermediate steps.

Notice a few things:

- This way of explicitly writing down the expression can be error-prone and tiring, and this was a tiny example with only two parameters. You can imagine how big of a mess it will get for a billion parameter neural network. In fact, even for a few layers deep small neural network, the calculation is tedious.
- Note that, there is a big overlap between the two derivative expression derivations. The first lines are almost the same for both expressions. Is there any way to effectively reuse these parts?

In reality, when we do automatic backpropagation through PyTorch (or other framework's) autograd engine, we never explicitly write down the expression. Automatically dealing with expressions (the way we humans do) is called symbolic differentiation and it is really difficult to automate that at scale in computers. Rather, what we actually do is a very neat trick with the chain rule of calculus. In fact, backpropagation is just a chain rule on steroids, which makes everything very neat.

Chain Rule Revisited

In case you need a refresher on the chain rule, here is the segment on chain rule from the Wikipedia[1].

> The chain rule may also be expressed in Leibniz's notation. If a variable y depends on the variable u, which itself depends on the variable x, then y depends on x as well, via the intermediate variable u. In this case, the chain rule is expressed as:
>
> $$\frac{dy}{dx} = \frac{dy}{du}\cdot\frac{du}{dx}$$

However, in deep learning, we are more interested in partial derivatives in a multi-variable setting. The chain rule for multi-variable partial derivatives is like this:

$$\frac{\partial y}{\partial x_i} = \sum_{\ell=1}^{m}\frac{\partial y}{\partial u_\ell}\frac{\partial u_\ell}{\partial x_i} \tag{1}$$

[1] `https://en.wikipedia.org/wiki/Chain_rule`

This just says that if there are dependencies through multiple variables, we need to add those dependencies. That is, if y is a function of many different intermediate variables, we need to take all of them into consideration and add them.

Chain rule provides us with a neat way to rewrite very sophisticated expressions in terms of simpler arithmetic operations (addition, negation, multiplication, division etc) and basic functions (sin, cosine, log exp etc) and then apply chain rule in succession. When done right, we can compute the derivative of very complex functions, using the "local" derivatives that are easy to compute numerically and do not depend on any other part.

It is easier to demonstrate through an example than to say in words.

An Easy Example

Consider the following function (this is the same function as before without the sigmoid, just to reduce the complexity. We will get back to sigmoid one shortly):

$$L = [(wx + b) - y]^2$$

Now, let's rewrite it this way:

$$\begin{aligned} p &= wx \\ q &= b \\ r &= p + q \\ s &= r - y \\ L &= s^2 \end{aligned}$$

The whole expression is written in a way that in each line there is at most one operation. Now, let's take the "local" derivatives for each line:

$$\begin{aligned} \frac{\partial p}{\partial w} &= x \\ \frac{\partial q}{\partial b} &= 1 \\ \frac{\partial r}{\partial p} &= 1, \quad \frac{\partial r}{\partial q} = 1 \\ \frac{\partial s}{\partial r} &= 1 \\ \frac{\partial L}{\partial s} &= 2s \end{aligned}$$

Note several things here: in lines 1 and 4, we don't take derivative wrt x and y because they are constants in this problem and thus zero. Secondly, in all the expressions, the local derivatives only depend on the subexpressions that can be calculated numerically if we are given the values of x, y, w and b (x and y are input values, which are given. The other variables w and b will be initialized randomly). For example, in 5, the “local” gradient is 2s, which can be numerically calculated. Again, remember that the computer doesn’t actually deal with expressions, rather it only computes the numerical values of the partial derivatives and passes them around. The coding assignment makes this more explicit. The reason we are writing out the whole expression of the gradient, instead of plugging in values, is for educational purposes: it is easier to verify and play around with expressions.

Now, let’s do the actual job: calculating $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$.

$$\begin{aligned}\frac{\partial L}{\partial w} &= \frac{\partial L}{\partial s}\frac{\partial s}{\partial w} \\ &= \frac{\partial L}{\partial s}\frac{\partial s}{\partial r}\frac{\partial r}{\partial w} \\ &= \frac{\partial L}{\partial s}\frac{\partial s}{\partial r}\left(\frac{\partial r}{\partial p}\frac{\partial p}{\partial w} + \frac{\partial r}{\partial q}\frac{\partial q}{\partial w}\right)\end{aligned}$$

The first line is just applying the chain rule on s, and so is the second line. The third line has an addition because r is dependent on p and q, which is why we need to sum their dependencies.

Notice that everything in the final line is actually already computed as “local” derivative. We can simply plug them in and we are done! [2]

If you plug everything in, you will get 2sx, which expands to 2x(wx + b -y). You can differentiate manually to see that this is the right answer.

Task 1 (3 pt)

Compute the expression of $\frac{\partial L}{\partial b}$ using the ”local” gradients computed above.

Backpropagation is a general algorithm and can be applied anywhere to take the derivative of any complicated function, not just in neural networks or loss functions.

To illustrate this, take this random function $f(x, b, w) = (we^{bx} + \ln(wx))b$ which is very highly unlikely to show up in neural networks. We will derive $\frac{\partial f}{\partial b}$ and $\frac{\partial f}{\partial w}$ now. [3]

And let’s derive $\frac{\partial f}{\partial b}$. First, let’s make a table of the expression breakdown and their local gradients (Table 1).

[2] We didn’t compute dq/dw before because q doesn’t depend on w and is thus zero; we usually initialize every gradient as zero to deal with this case.

[3] We are using the same variable names x, w, and b to keep the conventions consistent throughout this homework.

Breakdown	**Local gradient**	**Expansion**
$p = wx$	$\frac{\partial p}{\partial w} = x$	$p = wx$
$q = \ln p$	$\frac{\partial q}{\partial p} = \frac{1}{p}$	$q = \ln(wx)$
$r = bx$	$\frac{\partial r}{\partial b} = x$	$r = bx$
$s = e^r$	$\frac{\partial s}{\partial r} = e^r$	$s = e^{bx}$
$t = sw$	$\frac{\partial t}{\partial s} = w$, $\frac{\partial t}{\partial w} = s$	$t = e^{bx} \cdot w$
$u = t + q$	$\frac{\partial u}{\partial t} = 1$, $\frac{\partial u}{\partial q} = 1$	$u = e^{bx} \cdot w + \ln(wx)$
$f = bu$	$\frac{\partial f}{\partial b} = u$, $\frac{\partial f}{\partial u} = b$	$f = b(e^{bx} \cdot w + \ln(wx))$

Table 1: In this table, we put everything in one place: the expression breakdown, their local gradients and expansions of the breakdowns, which we call forward pass here. The expansions come in handy later in substituting back.

$$\begin{aligned}
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial u}\frac{\partial u}{\partial b} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial b} \\
&= \frac{\partial f}{\partial u}\frac{\partial u}{\partial b} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial b} \\
&= \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial t}\frac{\partial t}{\partial b} + \frac{\partial u}{\partial q}\frac{\partial q}{\partial b}\right) + \frac{\partial f}{\partial b} \\
&= \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial t}\frac{\partial t}{\partial s}\frac{\partial s}{\partial b} + \frac{\partial u}{\partial q}\frac{\partial q}{\partial b}\right) + \frac{\partial f}{\partial b} \\
&= \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial t}\frac{\partial t}{\partial s}\frac{\partial s}{\partial r}\frac{\partial r}{\partial b} + \frac{\partial u}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial b}\right) + \frac{\partial f}{\partial b}
\end{aligned}$$

In the first line, we applied the multivariable partial derivative chain rule since f is a function of both u and b. In the second line, $\frac{\partial b}{\partial b}$ is 1 and we substituted in the next line. In the third line, we applied the multivariable partial derivative chain rule on u.

If we substitute the individual local derivatives in the final expression, we get: $b \cdot w \cdot x \cdot e^{bx} + w \cdot e^{bx} + \ln(w \cdot x)$ You can manually do the differentiation and verify it.

Task 2 (5 pt)

Calculate $\frac{\partial f}{\partial w}$ for the function above.

Now, let's come back to the deep learning world. Let's try to compute the gradient for this function wrt w and b, which we originally showed.

$$L = [\sigma(wx + b) - y]^2$$

Task 3 (5 pt)

But first, as an exercise: show that $\frac{\partial d}{\partial x}\sigma(x) = \sigma(x)(1-\sigma(x))$ using backpropagation that we learned. Please make a table like the above and only use elementary operations and chain rule of partial derivative. You may use some algebraic manipulation at the end to rewrite the expression in the expected form.

Task 4 (10 pt)

Now, derive $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ for $L = [\sigma(wx+b) - y]^2$ using the method shown above. Write your result in terms of w, x, y, and b. However, you can leave out σ and σ' as they are so that the expression is not too cluttered. First, make a table like the one above and then do the calculation. If you feel drawing a computation graph will help you, feel free to do so, but you don't have to.

Task 5 (20 pt)

Take this expression: $f(w_1, w_2, b_1, b_2, x, \lambda) = [y - \sigma(w_2\sigma(w_1x + b_1) + b_2)]^2 + \lambda[w_1^2 + w_2^2]$ where x is the input and λ is a constant. The expression looks like a two-layer neural network with L2 regularization. Derive partial derivatives of f with respect to w_1, w_2, b_1, b_2. Make a table like before and show your calculations.

Task 6 (5 pt)

From your derivation in Task 5, what is the best ordering of variables to compute so that we can maximize reusing previous computations? Answer in one sentence.