EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Discussion 2

1. Visualizing Derivatives

Consider a simple neural network that takes a scalar real input, has 1 hidden layer with k units in it and a ReLU nonlinearity for those units, and an output linear (affine) layer.

We can algebraically write any function that it represents as

$$y = W^{(2)}(\max(\mathbf{0}, W^{(1)}x + \mathbf{b^{(1)}})) + b^{(2)}$$

Where $x, y \in \mathbb{R}$, $W^{(1)} \in \mathbb{R}^{k\times 1}$, $W^{(2)} \in \mathbb{R}^{1\times k}$, and $\mathbf{b}^{(1)} \in \mathbb{R}^{k\times 1}$, and $b^{(2)} \in \mathbb{R}$. The superscripts are indices, not exponents and the max given two vector arguments applies the max on corresponding pairs and returns a vector.

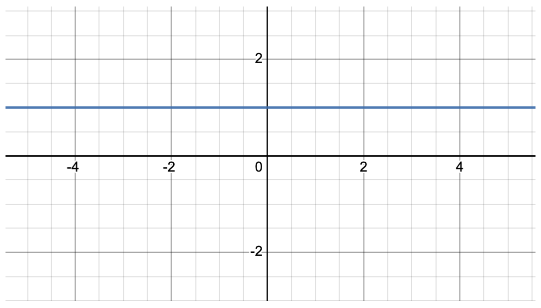
For each part, **calculate the partial derivative and sketch a small representative plot of the derivative as a function of** x. Make sure to clearly label any discontinuities, kinks, and slopes of segments. The subscript i refers to the i-th element of a vector.

Solution: Note that the numeric values on the axes, except for part (a), are **not** indicative of where the non-linearity/discontinuity happens, so please focus on the shape of the function and not the numbers. The numbers are by-products generated by the online graphing calculator. All the critical points in the following subparts happen at $x = \frac{-b_i^{(1)}}{W_i^{(1)}}$.

(a) $\frac{\partial y}{\partial b^{(2)}}$

Solution:

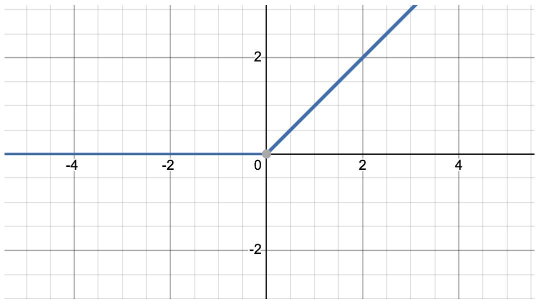
$$\frac{\partial y}{\partial b^{(2)}} = 1$$



(b) $\frac{\partial y}{\partial w_i^{(2)}}$

Solution:

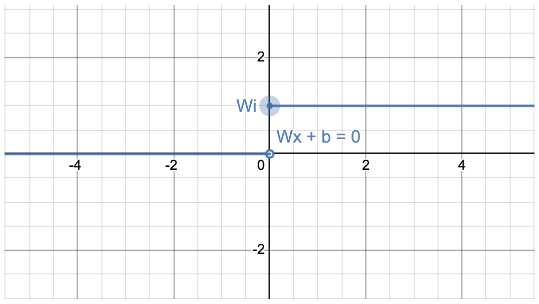
$$\frac{\partial y}{\partial W_i^{(2)}} = \max(0, W_i^{(1)}x + b_i^{(1)})$$



(c) $\frac{\partial y}{\partial b_i^{(1)}}$

Solution:

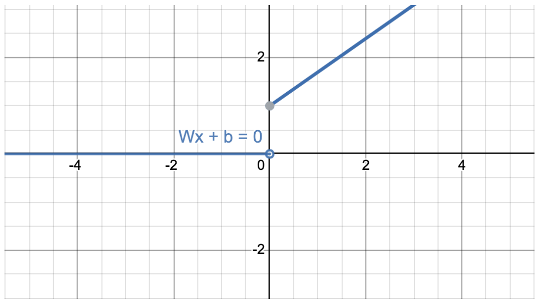
$$\frac{\partial y}{\partial b_i^{(1)}} = \begin{cases} W_i^{(2)}, \text{ if } W^{(1)}x + b^{(1)} > 0 \\ 0, \text{ if } W^{(1)}x + b^{(1)} < 0 \end{cases}$$



(d) $\frac{\partial y}{\partial w_i^{(1)}}$

Solution:

$$\frac{\partial y}{\partial w_i^{(1)}} = \begin{cases} W_i^{(2)}x, \text{ if } W^{(1)}x + b^{(1)} > 0 \\ 0, \text{ if } W^{(1)}x + b^{(1)} < 0 \end{cases}$$

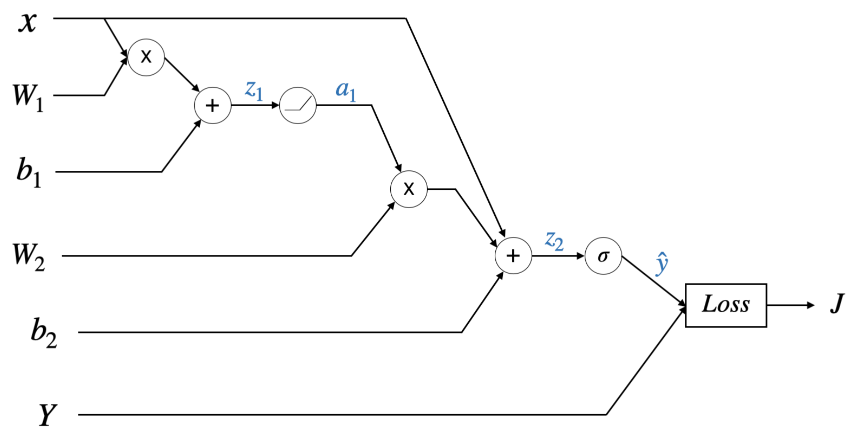


2. Computation Graph Review

One of the reasons why modern neural networks are growing much larger than ten years ago is due to the advent of automatic differentiation softwares, also known as *Autodiff*. Autodiff softwares can compute the gradients of any differentiable function by constructing a *computation graph* of such functions, which allows researchers to simply focus on building models with effective information propagation and not to worry about the complex back prop.

Here is the computation graph of an one hidden layer MLP with *skip connection*, which basically means adding the input to the output of some later layers (this is a commonly used design that we will cover in detail later in the course). σ denotes the sigmoid function and J is the final loss. Assume that we use binary cross-entropy as our loss function with a ground truth label y, ie. $loss = -y \log \hat{y} - (1-y) log(1-\hat{y})$.
You may find this quantity useful:

- $\frac{\partial J}{\partial z_2} = \hat{y} - y$



(a) **Express $\hat{y}$ as a function of x, W_1, b_1, W_2 and b_2.**

Solution:

$$\begin{aligned}\hat{y} &= \sigma(z_2) \\ &= \sigma(W_2 a_1 + b_2 + x) \\ &= \sigma(W_2 \cdot ReLU(W_1 x + b_1) + b_2 + x)\end{aligned} \tag{1}$$

(b) **Compute the gradient $\frac{\partial J}{\partial W_2}$, $\frac{\partial J}{\partial b_2}$.**

Solution:

- $\frac{\partial J}{\partial W_2} = \frac{\partial J}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2} = (\hat{y} - y) \cdot a_1$
- $\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = \hat{y} - y$

(c) **Compute the gradient $\frac{\partial J}{\partial W_1}$, $\frac{\partial J}{\partial b_1}$, $\frac{\partial J}{\partial x}$.**

Solution:

Note how $\frac{\partial J}{\partial x}$ relies on two streams of gradients.

- $\frac{\partial J}{\partial W_1} = \frac{\partial J}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial W_1} = (\hat{y} - y) \cdot W_2 \cdot \frac{\partial a_1}{\partial z_1} \cdot x$

- $\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = (\hat{y} - y) \cdot W_2 \cdot \frac{\partial a_1}{\partial z_1}$, where

$$\frac{\partial a_1}{\partial z_1} = \begin{cases} 1, \text{if } z_1 > 0 \\ 0, \text{if } z_1 < 0 \end{cases}$$

- $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial x} + \frac{\partial J}{\partial z_2} \cdot \frac{\partial z_2}{\partial x} = (\hat{y} - y) \cdot W_2 \cdot \frac{\partial a_1}{\partial z_1} \cdot W_1 + (\hat{y} - y)$

(d) What **intermediate variables do we need to cache** in this case? **Why is it more efficient to have an additional backward pass** on top of the forward pass for gradient computation?

Solution:

We will need to cache $\hat{y}$, x and all the intermediate variables z_2, z_1, a_1, as we will need them when applying the chain rule.

As we see in part (b) and (c), the downstream gradients depends on the upstream. Therefore by doing back propagation, we may compute all the parameters' gradients in a dynamic programming manner according to the computation graph, without having to compute the same partial derivative terms repeatedly.

3. MAP Interpretation of Ridge Regression

Consider the Ridge Regression estimator,

$$\underset{\mathbf{w}}{\operatorname{argmin}} \|X\mathbf{w} - y\|_2^2 + \lambda\|\mathbf{w}\|^2$$

We know this is solved by

$$\hat{\mathbf{w}} = (X^T X + \lambda I)^{-1} X^T \mathbf{y} \tag{2}$$

One interpretation of Ridge Regression is to find the Maximum A Posteriori (MAP) estimate on **w**, the parameters, assuming that the prior of **w** is $\mathcal{N}(0, I)$ and that the random **Y** is generated using

$$\mathbf{Y} = X\mathbf{w} + \sqrt{\lambda}N$$

Note that each entry of vector N is zero-mean, unit-variance normal. **Show that Equation 2 is is indeed the MAP estimate for w given an observation on Y=y.**

Solution: From how we define MAP estimation,

$$MAP(\mathbf{w}|\mathbf{Y} = \mathbf{y}) = \underset{\mathbf{w}}{\operatorname{argmax}} f(\mathbf{w}|\mathbf{Y} = \mathbf{y})$$

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \frac{f(\mathbf{w}, \mathbf{y})}{f(\mathbf{y})}$$

The denominator doesn't affect the argmax since it doesn't depend on **w**, so we can omit it. Then we can use the chain rule to expand out the numerator

$$= \underset{\mathbf{w}}{\operatorname{argmax}} f(\mathbf{w}) f(\mathbf{y}|\mathbf{w})$$

Now, split up the conditional joint density into the product of conditional densities since each element of the **y** is independent given **w**.

$$= \underset{\mathbf{w}}{\operatorname{argmax}} f(\mathbf{w}) \prod_{i=1}^{n} f(y_i|\mathbf{w})$$

We can now recall the formula for standard normal pdf is $f_Z(z) = \frac{e^{-z^2/2}}{\sqrt{2\pi}}$. To find$f(y_i|\mathbf{w})$, we know that $y_i = \mathbf{x_i}^T\mathbf{w} + \sqrt{\lambda}N_i$, where $N_i \sim \mathcal{N}(0, 1)$, so we'd have $\frac{y_i - \mathbf{x_i}^T\mathbf{w}}{\sqrt{\lambda}} \sim \mathcal{N}(0, 1)$. Now, plugging in the pdf in the previous expressions we'd have

$$MAP = \underset{\mathbf{w}}{\operatorname{argmax}} \frac{e^{-\|\mathbf{w}\|^2/2}}{\sqrt{2\pi}} \prod_{i=1}^{n} \frac{e^{-(\frac{y_i - \mathbf{x_i}^T\mathbf{w}}{\sqrt{\lambda}})^2/2}}{\sqrt{2\pi}}$$

We can ignore multiplicative scaling constants (since they don't affect the value of the argmax. Also, since log is a monotonically increasing function, we can take log of both sides without affecting the argmax. Taking logs is useful since it allows us to turn products into sums. So we now have:

$$MAP = \underset{\mathbf{w}}{\operatorname{argmax}} -\frac{\|\mathbf{w}\|^2}{2} - \frac{1}{\lambda}\sum_i \frac{(y_i - \mathbf{x}_i^T\mathbf{w})^2}{2}$$

We can ignore scaling factors again, and arrange all the summation terms in a vector and taking the norm-squared. We can also turn argmax into argmin by negating the objective function:

$$MAP = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\|^2 + \frac{1}{\lambda}\|\mathbf{y} - X\mathbf{w}\|^2$$

Finally, we can multiply through by λ without changing the argmax since it is a positive constant:

$$MAP = \underset{\mathbf{w}}{\operatorname{argmax}}\, \lambda\|\mathbf{w}\|^2 + \|X\mathbf{w} - \mathbf{y}\|^2$$

And now it is the same form as the original Ridge Regression optimization problem.

Contributors:

- Saagar Sanghavi.
- Anant Sahai.
- Kevin Li.