EECS 182 Deep Neural Networks — Fall 2025 — Anant Sahai and Gireeja Ranade — 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. You may assume values for b's and w's, for example, the solutions will use these values for the plots: $w_1^{(1)} = 2, w_2^{(1)} = 2, b_1^{(1)} = -1, b_2^{(1)} = -2, w_1^{(2)} = 0.5, w_2^{(2)} = 2.0$.

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.

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

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

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

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

(e) When you fine-tune your network, you observe a coupled motion in your parameters as gradient descent adjusts them. In particular, whenever the bias $b_1^{(1)}$ on the first hidden unit moves up by 0.01 it seems that the weight $w_2^{(1)}$ on the second hidden unit also moves up by 0.02. Draw a 1-dimensional synthetic feature (function of x) that represents this observed regularity. Sketch a small representative plot of this synthetic feature.

Hint: Fix all the parameters in the network except for $b_1^{(1)}$ and $w_2^{(1)}$ and consider how the output will change due to small changes in these two parameters. We can approximate this with the following:

$$\theta = \begin{bmatrix} w_2^{(1)} & b_1^{(1)} \end{bmatrix} \tag{1}$$

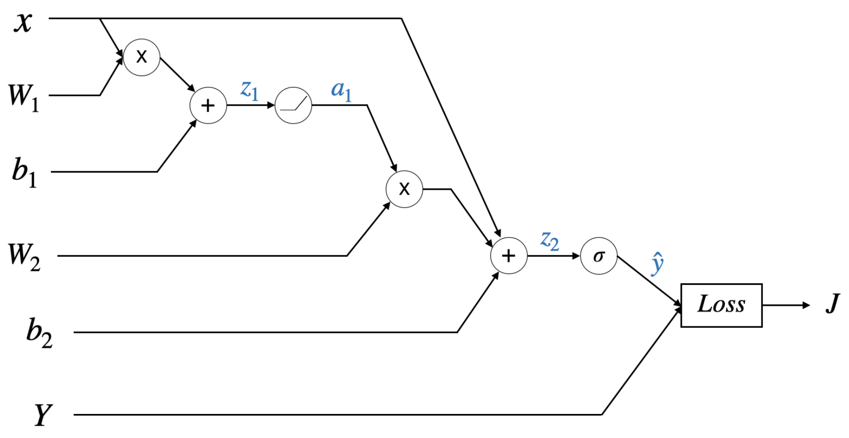
$$y = f_{\theta_0 + \Delta\theta}(x) \approx f_{\theta_0}(x) + \frac{\partial f}{\partial \theta}(x)\Delta\theta. \tag{2}$$

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. $loss = -y \ln \hat{y} - (1-y)\ln(1-\hat{y})$.
You may find this quantity useful:

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



(a) **Express $\hat{y}$ as a function of $\boldsymbol{x}$, $\boldsymbol{W}_1$, $\boldsymbol{b}_1$, $\boldsymbol{W}_2$ and $\boldsymbol{b}_2$.**

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

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

(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 the computation of derivatives?

3. ReLU with different Optimizers

Work through the notebook to explore how the simple neural network with ReLU non-linearities from last discussion learns using different optimizers. The notebook compares SGD, SGD with momentum, and Adam. Notebook url - `https://tinyurl.com/cs182-dis02-code`

Contributors:

- Saagar Sanghavi.
- Anant Sahai.
- Qiyang Li.
- Kevin Li.
- Kumar Krishna Agrawal.
- Naman Jain.