

EECS 182 Deep Neural Networks

Fall 2026 Gireeja Ranade

Discussion 2

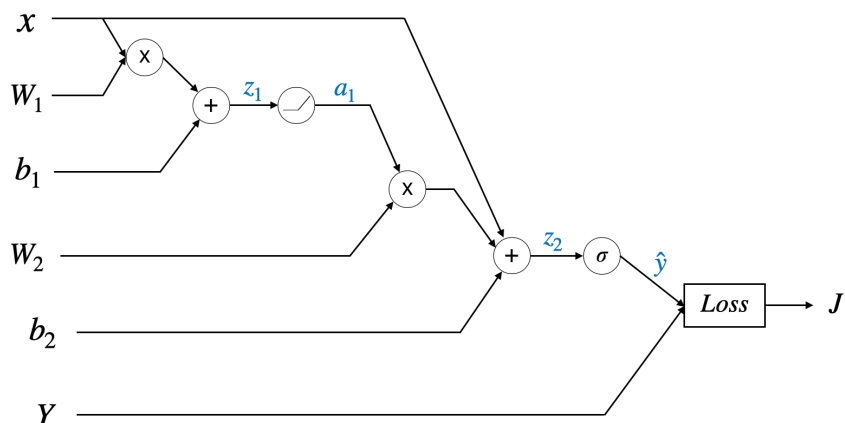
1. Computation Graph Review

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 x , W_1 , b_1 , W_2 and 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) **[Jump to this in xx:30 (skip previous ones if you haven't finished)]** 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?

(e) **Quantify the memory used by this AutoDiff framework.**

Every named quantity here is a scalar. Count the parameter; input/label; forward values; parameter gradients; and backward deltas. Each is stored in `float32` and occupies **4 bytes**.

For each storage group, compute its memory in bytes and sum the total peak memory.

(f) Under the memory accounting in part (e), can a naive reverse-mode AutoDiff implementation that retains every listed value perform direct gradient backpropagation with only **64 bytes** of memory? If not, briefly discuss alternatives that can still produce the exact gradients.

2. Visualizing Derivatives

Consider a simple neural network that takes a scalar real input, has one hidden layer with $k = 2$ units and a ReLU nonlinearity for those units, and has an output linear (affine) layer.

We can algebraically write any function that it represents in matrix form as

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

Here $x, y \in \mathbb{R}$, $W^{(1)} \in \mathbb{R}^{2 \times 1}$, $W^{(2)} \in \mathbb{R}^{1 \times 2}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{2 \times 1}$, 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. Let's assume

$$W^{(1)} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \quad W^{(2)} = \begin{bmatrix} 0.5 & 2.0 \end{bmatrix}.$$

Calculate each requested partial derivative. Then sketch **seven separate figures**, one for each of the following gradients as a function of x :

$$\frac{\partial y}{\partial b^{(2)}}, \quad \frac{\partial y}{\partial w_1^{(2)}}, \quad \frac{\partial y}{\partial w_2^{(2)}}, \quad \frac{\partial y}{\partial b_1^{(1)}}, \quad \frac{\partial y}{\partial b_2^{(1)}}, \quad \frac{\partial y}{\partial w_1^{(1)}}, \quad \frac{\partial y}{\partial w_2^{(1)}}.$$

Clearly label all discontinuities, kinks, and segment slopes.

(a) Calculate $\frac{\partial y}{\partial b^{(2)}}(x)$ and draw its figure.

(b) Calculate $\frac{\partial y}{\partial w_1^{(2)}}(x)$ and $\frac{\partial y}{\partial w_2^{(2)}}(x)$, and draw one separate figure for each gradient.

(c) Calculate $\frac{\partial y}{\partial b_1^{(1)}}(x)$ and $\frac{\partial y}{\partial b_2^{(1)}}(x)$, and draw one separate figure for each gradient.

(d) Calculate $\frac{\partial y}{\partial w_1^{(1)}}(x)$ and $\frac{\partial y}{\partial w_2^{(1)}}(x)$, and draw one separate figure for each gradient.

(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 a small amount $\alpha > 0$, the weight $w_2^{(1)}$ on the second hidden unit moves up by 2α . Design a 1-dimensional synthetic feature (function of x) showing how this coupled parameter movement affects output y by 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. The quantities $\frac{\partial f}{\partial \theta}(x)$ are what we have just plotted in the previous parts and are features (functions of x) of the neural network locally. We can approximate the output using

$$\theta = \begin{bmatrix} w_2^{(1)} \\ b_1^{(1)} \end{bmatrix}, \quad \Delta\theta = \alpha \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad (1)$$

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

Contributors:

- Matteo Guarrera.
- Mert Cemri.
- Sizhe Chen.
- Kevin Li.
- Saagar Sanghavi.
- Anant Sahai.
- Qiyang Li.