On the Complexity of First-Layer Gradient Calculation in Multi-Layer Neural Networks

When we train a neural network, every weight in every layer needs to know how it should change to reduce the overall error. This knowledge comes from the gradient — the rate at which the loss function changes with respect to each parameter. At the output layer, these gradients are relatively straightforward to compute because the error signal is directly visible. However, as we move closer to the input layer, the picture becomes far more complex.

In networks with several hidden layers, the gradients for the first layer are not just influenced by the error at the output but also by the chain of transformations applied by every intermediate layer. Each neuron's activation, each derivative of its nonlinearity, and each connection weight contribute to how the error signal is reshaped before it finally reaches the first layer. This makes the process of finding first-layer gradients both computationally heavier and conceptually less transparent.

In this article, we'll explore why computing gradients near the input layer requires more steps than for later layers, how the chain rule drives this complexity, and what this means for understanding backpropagation in deep neural networks.

The Network Setup

Let us take an example of a simple Neural Network with:

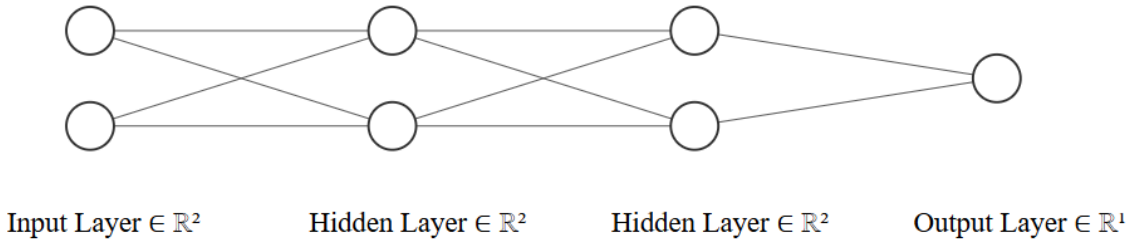
- Input vector column $X = \begin{bmatrix} x_1 \ x_2 \end{bmatrix}$
- Two hidden layers with 2 nodes in each layer
- Output layer with one node

For the simplicity of this article, we will not dwell deep into what the activation functions are, nor how they work. We'll just represent them by symbols

mentioned below.

Activations:

- For hidden layers (applies to each neuron): $\phi(\cdot)$
- For the final output layer: $\psi(\cdot)$



The Neural Network in reference

Forward Propagation: Pre-activations and Activations:

$$
\begin{aligned}
z^{(1)} &= W^{(1)}x + b^{(1)} \quad \in \mathbb{R}^2, \\
a^{(1)} &= \phi(z^{(1)}), \\
z^{(2)} &= W^{(2)}a^{(1)} + b^{(2)} \quad \in \mathbb{R}^2, \\
a^{(2)} &= \phi(z^{(2)}), \\
z^{(3)} &= W^{(3)}a^{(2)} + b^{(3)} \quad \in \mathbb{R}, \\
\hat{y} &= \psi(z^{(3)}).
\end{aligned}
$$

Error Signals

Before that, let me introduce "error signals": $\delta^{(l)} = \dfrac{\partial L}{\partial z^{(l)}}$. It is simply the gradient of the loss L w.r.t pre-activation z of layer l. It tells us how sensitive

the loss is to changes in that pre-activation.

Step-by-Step Gradient Flow

Our goal is to find $\frac{\partial L}{\partial W^{(1)}}$. For this, we will compute $\frac{\partial \mathcal{L}}{\partial z^{(l)}}$ for each layer l first.

1. Output layer

$$\delta^{(3)} = \frac{\partial L}{\partial \hat{y}}\, \psi'(z^{(3)})$$

Here,

- $\partial \mathcal{L}/\partial \hat{y}$ is the derivative of the loss wrt the output (a scalar).
- $\psi'(z^{(3)})$ is the derivative of the output activation at $z^{(3)}$.

2. Second hidden layer

$$\delta^{(2)} \;=\; \left(W^{(3)}\right)^{T} \delta^{(3)} \;\circ\; \phi'\!\left(z^{(2)}\right),$$

- $\left(W^{(3)}\right)^{T} \delta^{(3)}$ is the incoming error to layer 2 from the layer above it (output layer, in this case).
- $\circ\, \phi'\!\left(z^{(2)}\right)$ is the local derivative
- $\circ$: Hadamard (elementwise) product with the activation derivatives.

To calculate each neuron's error:

$$\delta_i^{(2)} = W_{1i}^{(3)}\, \delta^{(3)} \cdot \phi'(z_i^{(2)}), \quad i = 1, 2.$$

3. First hidden layer

$$\delta^{(1)} \;=\; \left(W^{(2)}\right)^{T} \delta^{(2)} \;\circ\; \phi'\!\left(z^{(1)}\right),$$

For individual neuron:

$$\delta_i^{(1)} = \left(W_{1i}^{(2)}\, \delta_1^{(2)} + W_{2i}^{(2)}\, \delta_2^{(2)} \right) \cdot \phi'(z_i^{(1)}), \quad i = 1, 2.$$

This shows:

- δi(1) comes from both neurons in layer 2 (since both depend on $a_i^{(1)}$)
- Then it is scaled by the derivative at the neuron's own pre-activation.

Since, this is a compact form, we might not get a clear view of the complexity. Let me expand this:

- For neuron 1 in first layer:

$$\delta_1^{(1)} = \left[W_{11}^{(2)}\, W_{11}^{(3)}\, \phi'(z_1^{(2)}) + W_{21}^{(2)}\, W_{12}^{(3)}\, \phi'(z_2^{(2)}) \right] \cdot \frac{\partial L}{\partial \hat{y}}\, \psi'(z^{(3)}) \cdot \phi'(z_1^{(1)}).$$

- Similarly, for neuron 2:

$$\delta_2^{(1)} = \left[W_{12}^{(2)}\, W_{11}^{(3)}\, \phi'(z_1^{(2)}) + W_{22}^{(2)}\, W_{12}^{(3)}\, \phi'(z_2^{(2)}) \right] \cdot \frac{\partial L}{\partial \hat{y}}\, \psi'(z^{(3)}) \cdot \phi'(z_2^{(1)}).$$

As we can see, the complexity for calculation of gradient increases drastically near the input layers in a deep network. Despite using a shallow network, consisting of 2 hidden layers, the gradient calculation is complex.

Weight Gradients in the First Hidden Layer

General formula:

$$\frac{\partial L}{\partial W_{ij}^{(1)}} = \delta_i^{(1)} \cdot x_j.$$

So explicitly:

$$
\begin{aligned}
\frac{\partial L}{\partial W_{11}^{(1)}} &= \delta_1^{(1)} \cdot x_1, \\
\frac{\partial L}{\partial W_{12}^{(1)}} &= \delta_1^{(1)} \cdot x_2, \\
\frac{\partial L}{\partial W_{21}^{(1)}} &= \delta_2^{(1)} \cdot x_1, \\
\frac{\partial L}{\partial W_{22}^{(1)}} &= \delta_2^{(1)} \cdot x_2.
\end{aligned}
$$

Observation

To update just **one weight in the first layer**, we need:

- the derivative of the **loss at the output**,
- the derivative of the **output activation**,
- contributions from *both neurons in hidden layer 2*,
- the weights connecting hidden layer 2 to the output,
- the derivative of hidden layer 2's activation,
- the derivative of the first layer neuron itself,
- the input value connected to that weight.

This layering shows why **gradients near the input are complex**: they combine signals from all deeper weights and nonlinearities, not just the local connection.

Why This Matters

This explicit expansion shows two important things:

- **Propagation of influence:** Every weight in the first layer is influenced by the whole stack of weights and derivatives above it — so training those early weights depends sensitively on deeper parameters and on activation function slopes.

- **Numerical issues:** Because many multiplicative factors multiply together, gradients can become very small (vanishing) or very large (exploding) as more layers are stacked. That is precisely why initialization, activation choice, residual connections, normalization and gradient clipping are common fixes.

Conclusion

Even in a small network, computing the gradient for first-layer weights shows how backpropagation involves chained influences from every deeper layer. This expansion not only explains why training early layers is tricky but also motivates the techniques used in modern deep learning to stabilize gradient flow.