Below is the full "manual backprop" for a network with 2 inputs, one hidden layer, 2 outputs and `nn.CrossEntropyLoss()`.

---

0) Setup and notation

Network architecture:

- Input: $x \in \mathbb{R}^2$
- First affine:

$$a_1 = W_1 x + b_1, \quad W_1 \in \mathbb{R}^{16\times 2},\ b_1 \in \mathbb{R}^{16}$$

- ReLU:

$$h = \mathrm{ReLU}(a_1) \in \mathbb{R}^{16}$$

- Second affine (logits):

$$z = W_2 h + b_2, \quad W_2 \in \mathbb{R}^{2\times 16},\ b_2 \in \mathbb{R}^2$$

- Targets: $y \in \{0, 1\}$ (class index)
- Loss: `CrossEntropyLoss`

For a **batch** of size N:

- $X \in \mathbb{R}^{N\times 2}$
- $A_1 \in \mathbb{R}^{N\times 16}$
- $H \in \mathbb{R}^{N\times 16}$
- $Z \in \mathbb{R}^{N\times 2}$
- $y \in \{0, 1\}^N$

---

1) CrossEntropyLoss derivative (loss → logits)

`nn.CrossEntropyLoss()` performs:

1. $P = \mathrm{softmax}(Z)$
2. $\mathcal{L} = -\dfrac{1}{N}\displaystyle\sum_{i=1}^{N} \log P_{i,y_i}$

Define one-hot targets $Y \in \mathbb{R}^{N\times 2}$.

Key result (softmax + NLL combined):

$$\frac{\partial \mathcal{L}}{\partial Z} = \frac{1}{N}(P - Y)$$

Define:

$$G_Z := \frac{\partial \mathcal{L}}{\partial Z}$$

Shape: $G_Z \in \mathbb{R}^{N \times 2}$

---

2) Backprop through the last Linear layer

Forward:

$$Z = HW_2^\top + \mathbf{1}b_2^\top$$

Gradients w.r.t. parameters

Weights

$$\frac{\partial \mathcal{L}}{\partial W_2} = G_Z^\top H$$

Bias

$$\frac{\partial \mathcal{L}}{\partial b_2} = \sum_{i=1}^{N} G_Z[i,:]$$

Gradient w.r.t. layer input

$$G_H := \frac{\partial \mathcal{L}}{\partial H} = G_Z W_2$$

---

3) Backprop through ReLU

ReLU:

$$H = \mathrm{ReLU}(A_1)$$

Derivative:

$$\frac{\partial H_{ij}}{\partial A_{1,ij}} = \begin{cases} 1 & A_{1,ij} > 0 \\ 0 & A_{1,ij} \leq 0 \end{cases}$$

Define mask:

$$M = \mathbf{1}[A_1 > 0]$$

Then:

$$G_{A_1} = \frac{\partial \mathcal{L}}{\partial A_1} = G_H \odot M$$

---

4) Backprop through the first Linear layer

Forward:

$$A_1 = XW_1^\top + \mathbf{1}b_1^\top$$

Gradients w.r.t. parameters

Weights

$$\frac{\partial \mathcal{L}}{\partial W_1} = G_{A_1}^\top X$$

Bias

$$\frac{\partial \mathcal{L}}{\partial b_1} = \sum_{i=1}^{N} G_{A_1}[i,:]$$

Gradient w.r.t. input

$$G_X := \frac{\partial \mathcal{L}}{\partial X} = G_{A_1} W_1$$

---

5) End-to-end backward summary

1. $P = \text{softmax}(Z)$
2. $G_Z = \frac{1}{N}(P - Y)$
3. $\nabla W_2 = G_Z^\top H$
4. $\nabla b_2 = \sum_i G_Z[i,:]$
5. $G_H = G_Z W_2$
6. $G_{A_1} = G_H \odot \mathbf{1}[A_1 > 0]$
7. $\nabla W_1 = G_{A_1}^\top X$
8. $\nabla b_1 = \sum_i G_{A_1}[i,:]$
9. $G_X = G_{A_1} W_1$ (optional)

This is exactly what PyTorch autograd computes internally.