Feed-Forward Neural Network: From Logistic Regression to Back-Propagation

Why this looks like logistic regression—only deeper

Logistic regression is a *single neuron*:

$$\hat{y} = \sigma(\boldsymbol{w}^T\mathbf{x} + b).$$

A feed-forward neural network is built by **stacking** that same "affine map + non-linearity" block. Each stack learns more expressive, non-linear features while the loss stays the familiar binary-cross-entropy (BCE).

---

1 The Training Objective

Key Insight: To train a neural network, we need to update each trainable parameter to minimize the loss function. The trainable parameters in our network are:

- First layer weights: $\boldsymbol{W}^{(1)}$ and biases: $\boldsymbol{b}^{(1)}$
- Output layer weights: $\boldsymbol{w}$ and bias: b

For each parameter θ, we need to compute $\frac{\partial L}{\partial \theta}$ to know:

- Which direction to move the parameter (increase or decrease)
- How much the loss changes when we change that parameter

This is why backpropagation computes the gradient of the loss with respect to **every trainable parameter**.

2 Network Architecture (2-2-1 Network)

Consider a concrete example with $d = 2$ inputs and $h = 2$ hidden units:

- **Input layer**: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^{2\times 1}$.
- **Linear transformation**:

$$a^{(1)} = \boldsymbol{W}^{(1)}\mathbf{x} + \boldsymbol{b}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Trainable parameters: $\boldsymbol{W}^{(1)} \in \mathbb{R}^{2\times 2}$ (4 parameters), $\boldsymbol{b}^{(1)} \in \mathbb{R}^{2\times 1}$ (2 parameters)

- **Hidden activation**: Using ReLU

$$\mathbf{h}^{(1)} = \text{ReLU}(a^{(1)}) = \begin{bmatrix} \max(0, a_1^{(1)}) \\ \max(0, a_2^{(1)}) \end{bmatrix}$$

- **Output layer**:

$$z = \boldsymbol{w}^T \mathbf{h}^{(1)} + b = [w_1, w_2] \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \end{bmatrix} + b$$

  Trainable parameters: $\boldsymbol{w} \in \mathbb{R}^{2\times 1}$ (2 parameters), $b \in \mathbb{R}$ (1 parameter)

- **Sigmoid prediction**: $\hat{y} = \sigma(z) = \dfrac{1}{1+e^{-z}} \in (0, 1)$.

Total trainable parameters: $4 + 2 + 2 + 1 = 9$ parameters

3 Binary-Cross-Entropy Loss

For a single sample:

$$L = -\big[y \ln \hat{y} + (1-y)\ln(1-\hat{y})\big]$$

Step-by-step derivation of $\frac{\partial L}{\partial \hat{y}}$:

$$\begin{aligned}
\frac{\partial L}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}}\big[-y \ln \hat{y} - (1-y)\ln(1-\hat{y})\big] && (1) \\
&= -y \cdot \frac{1}{\hat{y}} - (1-y) \cdot \frac{1}{1-\hat{y}} \cdot (-1) && (2) \\
&= -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} && (3) \\
&= \frac{-y(1-\hat{y}) + (1-y)\hat{y}}{\hat{y}(1-\hat{y})} && (4) \\
&= \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1-\hat{y})} && (5) \\
&= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} && (6)
\end{aligned}$$

4 Back-Propagation: Computing Loss Gradients for Each Trainable Parameter

We work backwards from the output, computing how the loss changes with respect to each trainable parameter.

4.1 Output Layer Gradients

4.1.1 1. Pre-activation gradient (not a parameter, but needed for chain rule)

First, derive $\frac{\partial \hat{y}}{\partial z}$ where $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$:

$$
\begin{aligned}
\frac{\partial \hat{y}}{\partial z} &= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) && (7)\\
&= -\frac{1}{(1+e^{-z})^2} \cdot \frac{\partial}{\partial z}(1+e^{-z}) && (8)\\
&= -\frac{1}{(1+e^{-z})^2} \cdot (-e^{-z}) && (9)\\
&= \frac{e^{-z}}{(1+e^{-z})^2} && (10)\\
&= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} && (11)\\
&= \sigma(z) \cdot \left(1 - \frac{1}{1+e^{-z}}\right) && (12)\\
&= \hat{y}(1-\hat{y}) && (13)
\end{aligned}
$$

Now apply chain rule:

$$
\begin{aligned}
\frac{\partial L}{\partial z} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} && (14)\\
&= \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) && (15)\\
&= \hat{y} - y && (16)
\end{aligned}
$$

$$\boxed{\frac{\partial L}{\partial z} = \hat{y} - y}$$

4.1.2 2. Gradient for trainable parameter b (output bias)

Since $z = \boldsymbol{w}^T \mathbf{h}^{(1)} + b$:

$$\frac{\partial z}{\partial b} = \frac{\partial}{\partial b}(\boldsymbol{w}^T \mathbf{h}^{(1)} + b) = 1 \qquad (17)$$

Therefore:

$$
\begin{aligned}
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} && (18)\\
&= (\hat{y} - y) \cdot 1 && (19)\\
&= \hat{y} - y && (20)
\end{aligned}
$$

$$\boxed{\frac{\partial L}{\partial b} = \hat{y} - y} \quad \text{(Gradient for trainable parameter } b\text{)}$$

4.1.3 3. Gradient for trainable parameter $\boldsymbol{w}$ (output weights)

Since $z = \boldsymbol{w}^T \mathbf{h}^{(1)} + b = w_1 h_1^{(1)} + w_2 h_2^{(1)} + b$:

$$\frac{\partial z}{\partial w_i} = h_i^{(1)} \tag{21}$$

$$\Rightarrow \frac{\partial z}{\partial \boldsymbol{w}} = \mathbf{h}^{(1)} \quad \text{(as a column vector)} \tag{22}$$

Step-by-step:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = (\hat{y} - y) \cdot h_1^{(1)} \tag{23}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_2} = (\hat{y} - y) \cdot h_2^{(1)} \tag{24}$$

In vector form:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \left(\frac{\partial L}{\partial z} \right)^T \mathbf{h}^{(1)T} \tag{25}$$

$$= (\hat{y} - y) \cdot \mathbf{h}^{(1)} \tag{26}$$

$$= \begin{bmatrix} (\hat{y} - y) \cdot h_1^{(1)} \\ (\hat{y} - y) \cdot h_2^{(1)} \end{bmatrix} \tag{27}$$

$$\boxed{\frac{\partial L}{\partial \boldsymbol{w}} = (\hat{y} - y) \cdot \mathbf{h}^{(1)}} \quad \text{(Gradient for trainable parameter } \boldsymbol{w}\text{)}$$

4.2 Hidden Layer Gradients

4.2.1 4. Hidden activation gradient (not a parameter, but needed for chain rule)

Since $z = \boldsymbol{w}^T \mathbf{h}^{(1)} + b$:

$$\frac{\partial z}{\partial h_i^{(1)}} = w_i \tag{28}$$

$$\Rightarrow \frac{\partial z}{\partial \mathbf{h}^{(1)}} = \boldsymbol{w} \tag{29}$$

Therefore:

$$\frac{\partial L}{\partial \mathbf{h}^{(1)}} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial \mathbf{h}^{(1)}} \tag{30}$$

$$= (\hat{y} - y) \cdot \boldsymbol{w} \tag{31}$$

$$= \begin{bmatrix} (\hat{y} - y) \cdot w_1 \\ (\hat{y} - y) \cdot w_2 \end{bmatrix} \tag{32}$$

$$\boxed{\frac{\partial L}{\partial \mathbf{h}^{(1)}} = \frac{\partial L}{\partial z} \boldsymbol{w}}$$

4.2.2 5. Pre-activation gradient (needed for first layer parameters)

For ReLU: $h_i^{(1)} = \max(0, a_i^{(1)})$, so:

$$\frac{\partial h_i^{(1)}}{\partial a_i^{(1)}} = g'(a_i^{(1)}) = \begin{cases} 1 & \text{if } a_i^{(1)} > 0 \\ 0 & \text{if } a_i^{(1)} \leq 0 \end{cases}$$

Using element-wise multiplication:

$$\frac{\partial L}{\partial a_1^{(1)}} = \frac{\partial L}{\partial h_1^{(1)}} \cdot g'(a_1^{(1)}) \tag{33}$$

$$\frac{\partial L}{\partial a_2^{(1)}} = \frac{\partial L}{\partial h_2^{(1)}} \cdot g'(a_2^{(1)}) \tag{34}$$

In vector form:

$$\boxed{\frac{\partial L}{\partial a^{(1)}} = \frac{\partial L}{\partial \mathbf{h}^{(1)}} \odot g'(a^{(1)})^T}$$

4.2.3 6. Gradient for trainable parameter $\boldsymbol{b}^{(1)}$ (first layer bias)

Since $a^{(1)} = \boldsymbol{W}^{(1)}\mathbf{x} + \boldsymbol{b}^{(1)}$:

$$\frac{\partial a_i^{(1)}}{\partial b_j^{(1)}} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{35}$$

Therefore:

$$\frac{\partial L}{\partial b_1^{(1)}} = \frac{\partial L}{\partial a_1^{(1)}} \cdot 1 + \frac{\partial L}{\partial a_2^{(1)}} \cdot 0 = \frac{\partial L}{\partial a_1^{(1)}} \tag{36}$$

$$\frac{\partial L}{\partial b_2^{(1)}} = \frac{\partial L}{\partial a_1^{(1)}} \cdot 0 + \frac{\partial L}{\partial a_2^{(1)}} \cdot 1 = \frac{\partial L}{\partial a_2^{(1)}} \tag{37}$$

$$\boxed{\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \frac{\partial L}{\partial a^{(1)}}} \quad \text{(Gradient for trainable parameter } \boldsymbol{b}^{(1)}\text{)}$$

4.2.4 7. Gradient for trainable parameter $\boldsymbol{W}^{(1)}$ (first layer weights)

Since $a^{(1)} = \boldsymbol{W}^{(1)}\mathbf{x} + \boldsymbol{b}^{(1)}$:

$$a_i^{(1)} = \sum_{k=1}^{2} W_{ik}^{(1)} x_k + b_i^{(1)} \tag{38}$$

$$\Rightarrow \frac{\partial a_i^{(1)}}{\partial W_{jk}^{(1)}} = \begin{cases} x_k & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{39}$$

Step-by-step for each weight:

$$\frac{\partial L}{\partial W_{11}^{(1)}} = \frac{\partial L}{\partial a_1^{(1)}} \cdot x_1 \tag{40}$$

$$\frac{\partial L}{\partial W_{12}^{(1)}} = \frac{\partial L}{\partial a_1^{(1)}} \cdot x_2 \tag{41}$$

$$\frac{\partial L}{\partial W_{21}^{(1)}} = \frac{\partial L}{\partial a_2^{(1)}} \cdot x_1 \tag{42}$$

$$\frac{\partial L}{\partial W_{22}^{(1)}} = \frac{\partial L}{\partial a_2^{(1)}} \cdot x_2 \tag{43}$$

In matrix form:

$$\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \left(\frac{\partial L}{\partial a^{(1)}}\right)^T \mathbf{x}^T \tag{44}$$

$$= \begin{bmatrix} \frac{\partial L}{\partial a_1^{(1)}} \\ \frac{\partial L}{\partial a_2^{(1)}} \end{bmatrix} [x_1, x_2] \tag{45}$$

$$= \begin{bmatrix} \frac{\partial L}{\partial a_1^{(1)}} \cdot x_1 & \frac{\partial L}{\partial a_1^{(1)}} \cdot x_2 \\ \frac{\partial L}{\partial a_2^{(1)}} \cdot x_1 & \frac{\partial L}{\partial a_2^{(1)}} \cdot x_2 \end{bmatrix} \tag{46}$$

$$\boxed{\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \left(\frac{\partial L}{\partial a^{(1)}}\right)^T \mathbf{x}^T} \quad \text{(Gradient for trainable parameter } \boldsymbol{W}^{(1)}\text{)}$$

---

5 Summary: Why We Need Each Gradient

5.1 Complete Gradient Flow

We computed gradients for all trainable parameters:

1. **Output bias** b: $\frac{\partial L}{\partial b} = \hat{y} - y$
2. **Output weights** $\boldsymbol{w}$: $\frac{\partial L}{\partial \boldsymbol{w}} = (\hat{y} - y) \cdot \mathbf{h}^{(1)}$
3. **First layer bias** $\boldsymbol{b}^{(1)}$: $\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \frac{\partial L}{\partial a^{(1)}}$
4. **First layer weights** $\boldsymbol{W}^{(1)}$: $\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \left(\frac{\partial L}{\partial a^{(1)}}\right)^T \mathbf{x}^T$

5.2 Gradient Descent Updates

For each trainable parameter θ, we update it using:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \frac{\partial L}{\partial \theta}$$

where η is the learning rate.

Key insight: The gradient $\frac{\partial L}{\partial \theta}$ tells us:

- If positive: increasing θ increases the loss $\Rightarrow$ we should decrease θ
- If negative: increasing θ decreases the loss $\Rightarrow$ we should increase θ
- The magnitude tells us how sensitive the loss is to changes in θ

A neural network learns to map inputs to outputs through a sequence of transformations—each layer takes the previous layer's output, applies weights and biases (linear transformation), then a non-linear activation function, progressively extracting more complex features until the final layer produces a prediction.

Each layer performs: $f(\mathbf{W}\mathbf{x} + \mathbf{b})$

The composition creates: $f_3(f_2(f_1(\mathbf{x})))$

Training adjusts $\mathbf{W}$ and $\mathbf{b}$ at each layer to minimize prediction error.