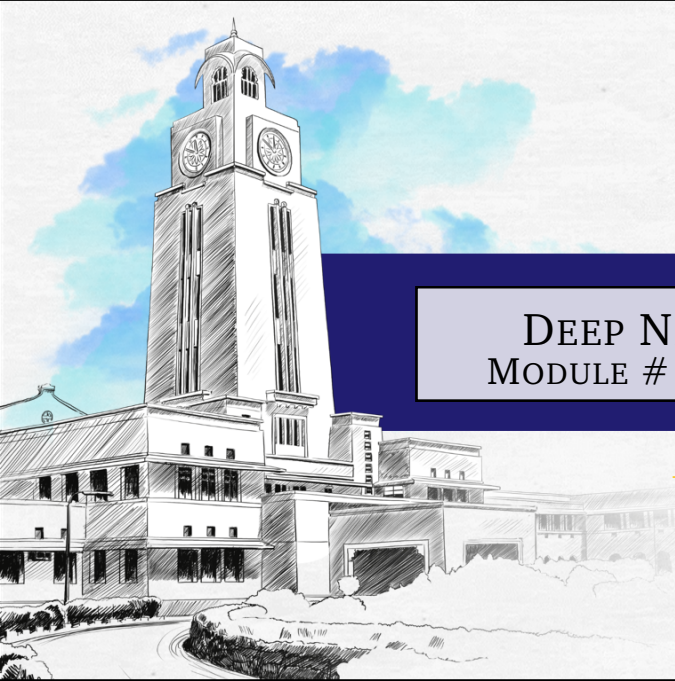


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Neural Network
Module # 5: DFNN Numerical

Seetha Parameswaran
BITS Pilani WILP

The instructor is gratefully acknowledging
the authors who made their course
materials freely available online.
This deck is prepared by Seetha Parameswaran.

Table of Contents

Numerical Problem : Forward Propagation

Problem: Consider a simple 2-layer neural network with the following parameters. Calculate the forward propagation step by step.

Network Architecture:

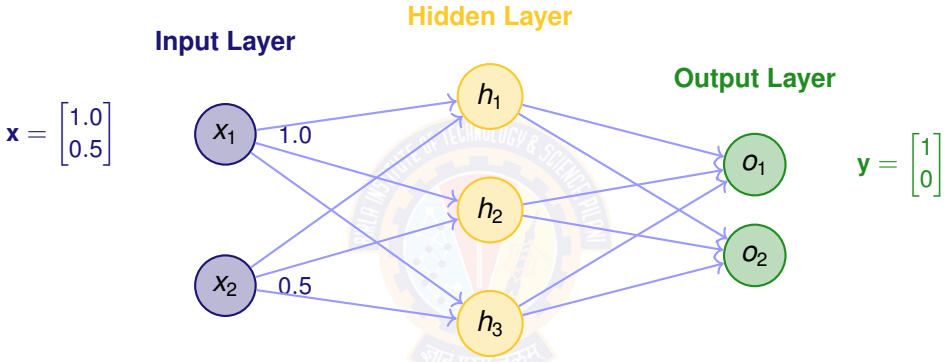
- Input layer: 2 neurons
- Hidden layer: 3 neurons (ReLU activation)
- Output layer: 2 neurons (Sigmoid activation)

Given Parameters:

$$x = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix} \quad W^{[1]} = \begin{bmatrix} 0.5 & 0.2 \\ -0.3 & 0.8 \\ 0.1 & -0.4 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} 0.4 & -0.1 & 0.6 \\ 0.2 & 0.7 & -0.3 \end{bmatrix}, \quad b^{[2]} = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$$

Forward Prop: Step 0 - Initial Network



Ready to compute forward propagation!

Solution: Forward Prop - Layer 1

Step 1: Calculate $z^{[1]}$

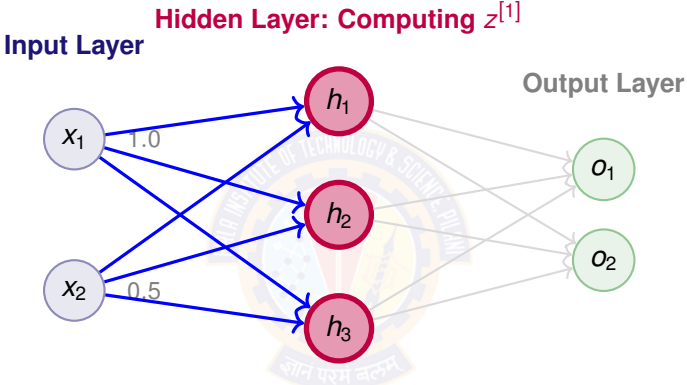
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$z^{[1]} = \begin{bmatrix} 0.5 & 0.2 \\ -0.3 & 0.8 \\ 0.1 & -0.4 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5(1.0) + 0.2(0.5) \\ -0.3(1.0) + 0.8(0.5) \\ 0.1(1.0) + (-0.4)(0.5) \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.7 \\ -0.1 \\ 0.2 \end{bmatrix}$$

Forward Prop: Step 1 - Computing $z^{[1]}$



$$z^{[1]} = W^{[1]}x + b^{[1]} = \begin{bmatrix} 0.5 & 0.2 \\ -0.3 & 0.8 \\ 0.1 & -0.4 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.7 \\ -0.1 \\ 0.2 \end{bmatrix}$$

Solution: Forward Prop - Layer 1 Activati

Step 2: Apply ReLU activation $a^{[1]} = \text{ReLU}(z^{[1]})$

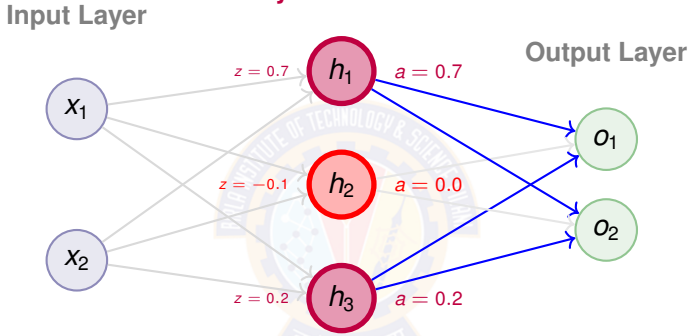
ReLU function: $\text{ReLU}(x) = \max(0, x)$

$$a^{[1]} = \text{ReLU}\begin{bmatrix} 0.7 \\ -0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} \max(0, 0.7) \\ \max(0, -0.1) \\ \max(0, 0.2) \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.0 \\ 0.2 \end{bmatrix}$$

Hidden layer activation: $a^{[1]} = \begin{bmatrix} 0.7 \\ 0.0 \\ 0.2 \end{bmatrix}$

Forward Prop: Step 2 - Applying ReLU

Hidden Layer: ReLU Activation



$$a^{[1]} = \text{ReLU}\begin{bmatrix} 0.7 \\ -0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.0 \\ 0.2 \end{bmatrix}$$

Neuron 2 is **dead** ($z < 0$)

Solution: Forward Prop - Layer 2

Step 3: Calculate $z^{[2]}$

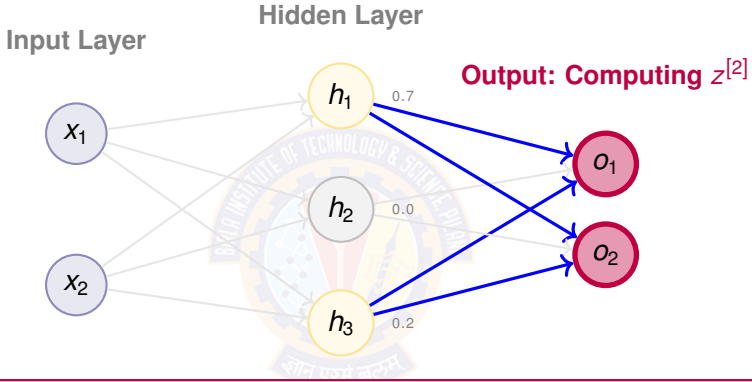
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$z^{[2]} = \begin{bmatrix} 0.4 & -0.1 & 0.6 \\ 0.2 & 0.7 & -0.3 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.0 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4(0.7) + (-0.1)(0.0) + 0.6(0.2) \\ 0.2(0.7) + 0.7(0.0) + (-0.3)(0.2) \end{bmatrix} + \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.28 + 0 + 0.12 \\ 0.14 + 0 - 0.06 \end{bmatrix} + \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} = \begin{bmatrix} 0.4 + 0.2 \\ 0.08 - 0.1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.02 \end{bmatrix}$$

Forward Prop: Step 3 - Computing $z^{[2]}$



$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} = \begin{bmatrix} 0.4 & -0.1 & 0.6 \\ 0.2 & 0.7 & -0.3 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.0 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.02 \end{bmatrix}$$

Solution: Forward Prop - Output Layer

Step 4: Apply Sigmoid activation $a^{[2]} = \sigma(z^{[2]})$

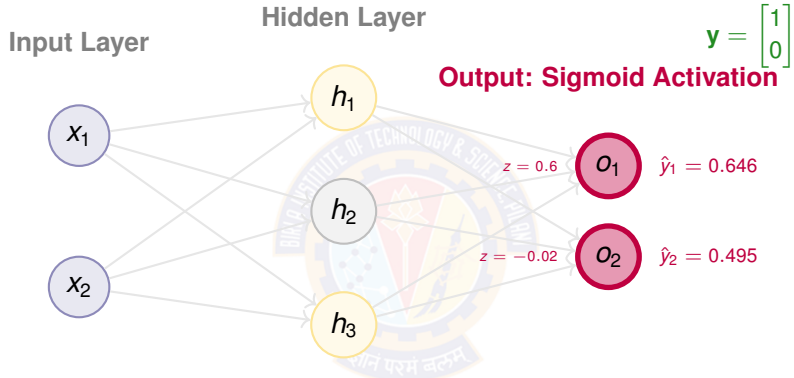
Sigmoid function: $\sigma(x) = \frac{1}{1 + e^{-x}}$

$$a^{[2]} = \sigma \begin{bmatrix} 0.6 \\ -0.02 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{1+e^{-0.6}} \\ \frac{1}{1+e^{0.02}} \end{bmatrix}$$

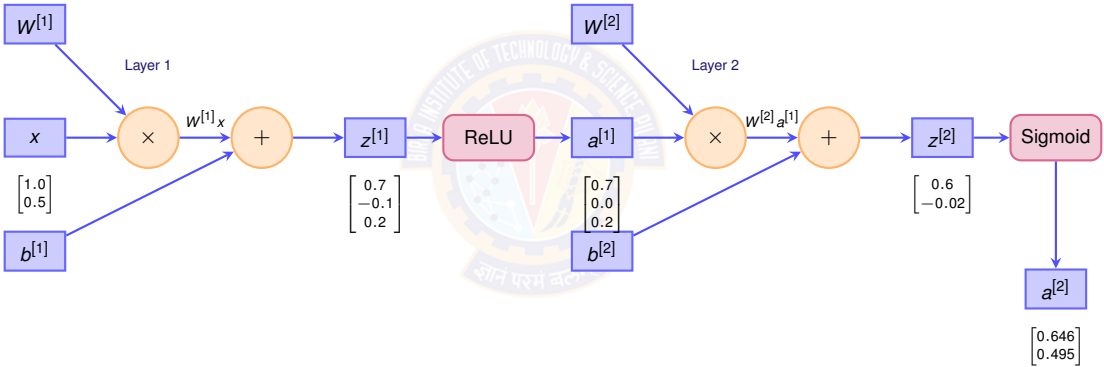
Output activation: $a^{[2]} = \begin{bmatrix} 0.646 \\ 0.495 \end{bmatrix}$

Forward Prop: Step 4 - Applying Sigmoid



Forward propagation complete! Prediction: $\hat{\mathbf{y}} = [0.646, 0.495]^T$

Computational Graph: Forward Propagatio



Numerical Problem: Backward Propagation

Problem: Using the previous network, calculate the backward propagation.

Given:

- Forward propagation results from previous problem
- Target output: $y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
- Loss function: Binary Cross-Entropy (BCE)

Calculate the gradients $\frac{\partial L}{\partial W^{[2]}}$, $\frac{\partial L}{\partial b^{[2]}}$, $\frac{\partial L}{\partial W^{[1]}}$, and $\frac{\partial L}{\partial b^{[1]}}$.

Solution: Backward Propagation - Loss

Step 1: Calculate the Binary Cross-Entropy loss

For our outputs: $a^{[2]} = \begin{bmatrix} 0.646 \\ 0.495 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

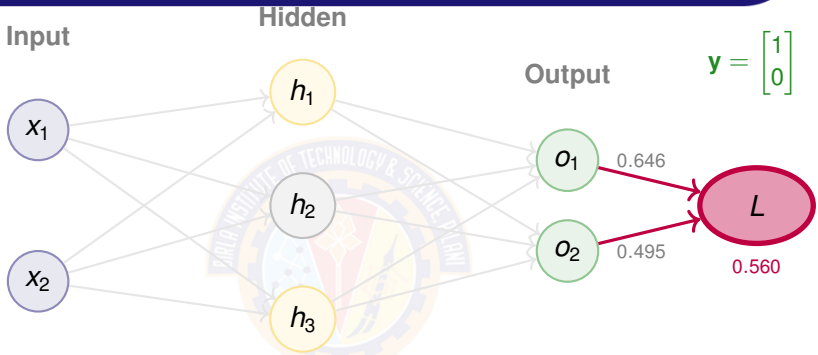
BCE Loss Formula: $L = \frac{-1}{n} \sum_{i=1}^{n} [y_i \log(a_i^{[2]}) + (1 - y_i) \log(1 - a_i^{[2]})]$

$$L = \frac{-1}{2} [1 \cdot \log(0.646) + (1 - 1) \cdot \log(1 - 0.646) + 0 \cdot \log(0.495) + (1 - 0) \cdot \log(1 - 0.495)]$$

$$= \frac{-1}{2} [\log(0.646) + \log(0.505)]$$

$$= \frac{-1}{2} [-0.437 + (-0.683)] = 0.560$$

Computing Loss



$$L = -\frac{1}{2}[y_1 \log(\hat{y}_1) + (1 - y_1) \log(1 - \hat{y}_1) + y_2 \log(\hat{y}_2) + (1 - y_2) \log(1 - \hat{y}_2)]$$
$$L = -\frac{1}{2}[\log(0.646) + \log(0.505)] = -\frac{1}{2}[-0.437 - 0.683] = 0.560$$

Now we backpropagate to update weights!

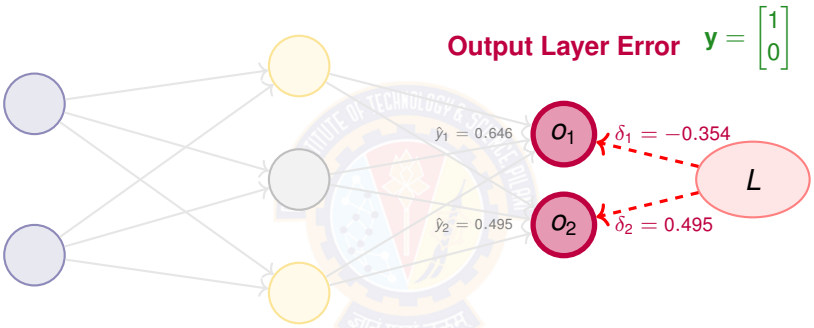
Solution: BackProp - Output Layer Error

Step 2: Calculate output layer error $\delta^{[2]}$

$$\frac{\partial L}{\partial z^{[2]}} = \delta^{[2]} = a^{[2]} - y$$

$$\delta^{[2]} = \begin{bmatrix} 0.646 \\ 0.495 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix}$$

Back-Prop Step 1 - Output Layer Error



$$\delta^{[2]} = \begin{bmatrix} 0.646 \\ 0.495 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix}$$

Solution: Backward Prop - Output Layer

Step 3: Calculate gradients for output layer

Recall: $a^{[1]} = \begin{bmatrix} 0.7 & 0.0 & 0.2 \end{bmatrix}^\top$

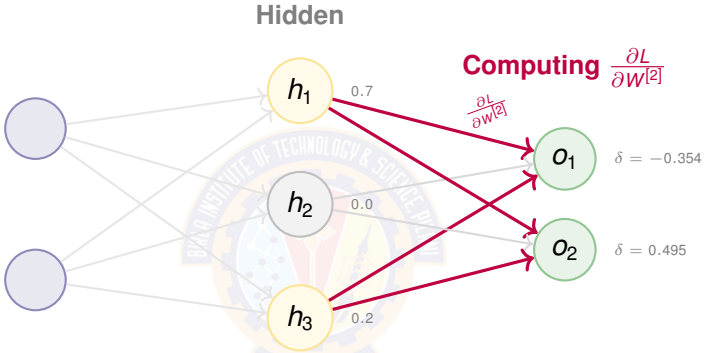
$$\frac{\partial L}{\partial W^{[2]}} = \delta^{[2]} (a^{[1]})^T = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix} \begin{bmatrix} 0.7 & 0.0 & 0.2 \end{bmatrix}$$

$$= \begin{bmatrix} -0.354 \times 0.7 & -0.354 \times 0.0 & -0.354 \times 0.2 \\ 0.495 \times 0.7 & 0.495 \times 0.0 & 0.495 \times 0.2 \end{bmatrix}$$

$$= \begin{bmatrix} -0.248 & 0.000 & -0.071 \\ 0.347 & 0.000 & 0.099 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{[2]}} = \delta^{[2]} = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix}$$

Back-Prop: Step 2 - Output Layer Gradient



$$\frac{\partial L}{\partial W^{[2]}} = \delta^{[2]}(a^{[1]})^T = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix} \begin{bmatrix} 0.7 & 0.0 & 0.2 \end{bmatrix} = \begin{bmatrix} -0.248 & 0.000 & -0.071 \\ 0.347 & 0.000 & 0.099 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{[2]}} = \delta^{[2]} = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix}$$

Solution: BackProp - Hidden Layer Error

Step 4: Calculate hidden layer error $\delta^{[1]}$

Backpropagate error to hidden layer: $\delta^{[1]} = (W^{[2]})^T \delta^{[2]} \odot g'^{[1]}(z^{[1]})$

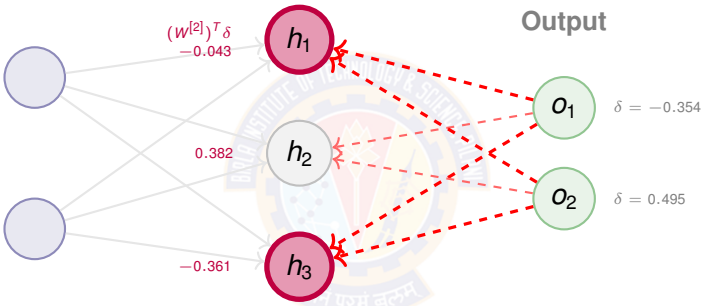
$$(W^{[2]})^T \delta^{[2]} = \begin{bmatrix} 0.4 & 0.2 \\ -0.1 & 0.7 \\ 0.6 & -0.3 \end{bmatrix} \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4(-0.354) + 0.2(0.495) \\ -0.1(-0.354) + 0.7(0.495) \\ 0.6(-0.354) + (-0.3)(0.495) \end{bmatrix}$$

$$= \begin{bmatrix} -0.142 + 0.099 \\ 0.035 + 0.347 \\ -0.212 - 0.149 \end{bmatrix} = \begin{bmatrix} -0.043 \\ 0.382 \\ -0.361 \end{bmatrix}$$

Back-Prop: Step 3 - Backprop to Hidden La

Backprop: $(W^{[2]})^T \delta^{[2]}$



$$(W^{[2]})^T \delta^{[2]} = \begin{bmatrix} 0.4 & 0.2 \\ -0.1 & 0.7 \\ 0.6 & -0.3 \end{bmatrix} \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix} = \begin{bmatrix} -0.043 \\ 0.382 \\ -0.361 \end{bmatrix}$$

Solution: BackProp - Hidden Layer Error

Step 4 (continued):

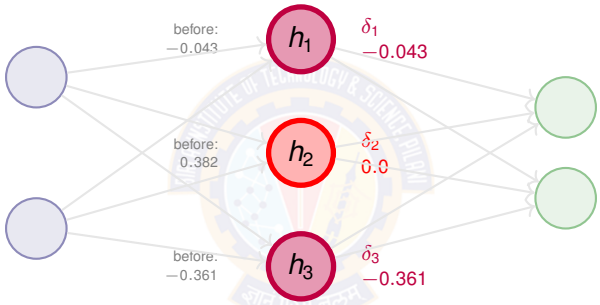
$$g'^{[1]}(z^{[1]}) = \begin{bmatrix} 1 & (\text{since } z_1^{[1]} = 0.7 > 0) \\ 0 & (\text{since } z_2^{[1]} = -0.1 \leq 0) \\ 1 & (\text{since } z_3^{[1]} = 0.2 > 0) \end{bmatrix}$$

Error to Layer 1: $\delta^{[1]} = (W^{[2]})^T \delta^{[2]} \odot g'^{[1]}(z^{[1]})$

$$\delta^{[1]} = \begin{bmatrix} -0.043 \\ 0.382 \\ -0.361 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.043 \\ 0.000 \\ -0.361 \end{bmatrix}$$

Back-Prop: Step 4 - Apply ReLU Derivative

Apply ReLU': $\delta^{[1]} = (W^{[2]})^T \delta^{[2]} \odot g'(z^{[1]})$



$$\delta^{[1]} = \begin{bmatrix} -0.043 \\ 0.382 \\ -0.361 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.043 \\ 0.0 \\ -0.361 \end{bmatrix}$$

Dead neuron gradient = 0 (no update!)

Step 5: Calculate gradients for hidden layer

$$\frac{\partial L}{\partial W^{[1]}} = \delta^{[1]} x^T = \begin{bmatrix} -0.043 \\ 0.000 \\ -0.361 \end{bmatrix} \begin{bmatrix} 1.0 & 0.5 \end{bmatrix}$$

$$\frac{\partial L}{\partial W^{[1]}} = \begin{bmatrix} -0.043 \times 1.0 & -0.043 \times 0.5 \\ 0.000 \times 1.0 & 0.000 \times 0.5 \\ -0.361 \times 1.0 & -0.361 \times 0.5 \end{bmatrix}$$

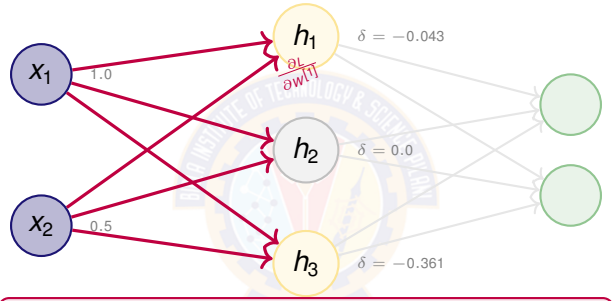
$$= \begin{bmatrix} -0.043 & -0.022 \\ 0.000 & 0.000 \\ -0.361 & -0.181 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{[1]}} = \delta^{[1]} = \begin{bmatrix} -0.043 \\ 0.000 \\ -0.361 \end{bmatrix}$$

Back-Prop: Step 5 - Hidden Layer Gradient

Computing $\frac{\partial L}{\partial W^{[1]}}$

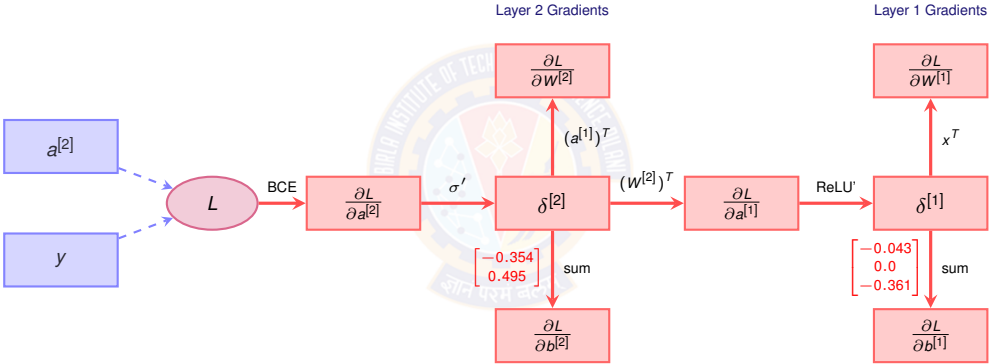
Input



$$\frac{\partial L}{\partial W^{[1]}} = \delta^{[1]}\mathbf{x}^T = \begin{bmatrix} -0.043 \\ 0.0 \\ -0.361 \end{bmatrix} \begin{bmatrix} 1.0 & 0.5 \end{bmatrix} = \begin{bmatrix} -0.043 & -0.022 \\ 0.0 & 0.0 \\ -0.361 & -0.181 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{[1]}} = \delta^{[1]} = \begin{bmatrix} -0.043 \\ 0.0 \\ -0.361 \end{bmatrix}$$

Computational Graph: Backward Propagat



Numerical Problem: Weight Updation

Problem: Using the gradients computed in previous problem, update the network parameters with learning rate $\alpha = 0.1$.

Given gradients:

$$\frac{\partial L}{\partial W^{[2]}} = \begin{bmatrix} -0.248 & 0.000 & -0.071 \\ 0.347 & 0.000 & 0.099 \end{bmatrix} \qquad \frac{\partial L}{\partial b^{[2]}} = \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix}$$

$$\frac{\partial L}{\partial W^{[1]}} = \begin{bmatrix} -0.043 & -0.022 \\ 0.000 & 0.000 \\ -0.361 & -0.181 \end{bmatrix} \qquad \frac{\partial L}{\partial b^{[1]}} = \begin{bmatrix} -0.043 \\ 0.000 \\ -0.361 \end{bmatrix}$$

Calculate the updated parameters.

Solution: Parameter Updates

Update rule: $\theta_{\text{new}} = \theta_{\text{old}} - \alpha \frac{\partial L}{\partial \theta}$

$$W^{[2]}_{\text{new}} = W^{[2]} - 0.1 \times \frac{\partial L}{\partial W^{[2]}}$$

$$W^{[2]}_{\text{new}} = \begin{bmatrix} 0.4 & -0.1 & 0.6 \\ 0.2 & 0.7 & -0.3 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.248 & 0.000 & -0.071 \\ 0.347 & 0.000 & 0.099 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4 + 0.025 & -0.1 + 0.000 & 0.6 + 0.007 \\ 0.2 - 0.035 & 0.7 + 0.000 & -0.3 - 0.010 \end{bmatrix}$$

$$= \begin{bmatrix} 0.425 & -0.100 & 0.607 \\ 0.165 & 0.700 & -0.310 \end{bmatrix}$$

$$b^{[2]}_{\text{new}} = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.354 \\ 0.495 \end{bmatrix} = \begin{bmatrix} 0.235 \\ -0.150 \end{bmatrix}$$

Solution: Parameter Updates (Continued)

$$W_{\text{new}}^{[1]} = W^{[1]} - 0.1 \times \frac{\partial L}{\partial W^{[1]}}$$

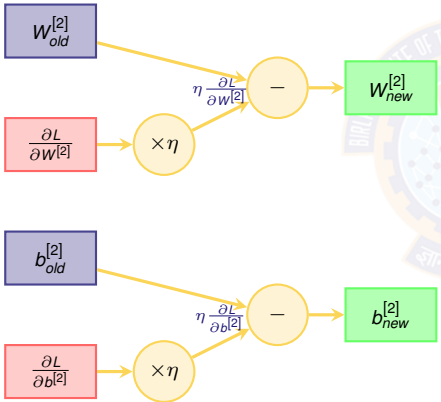
$$W_{\text{new}}^{[1]} = \begin{bmatrix} 0.5 & 0.2 \\ -0.3 & 0.8 \\ 0.1 & -0.4 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.043 & -0.022 \\ 0.000 & 0.000 \\ -0.361 & -0.181 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 + 0.004 & 0.2 + 0.002 \\ -0.3 + 0.000 & 0.8 + 0.000 \\ 0.1 + 0.036 & -0.4 + 0.018 \end{bmatrix} = \begin{bmatrix} 0.504 & 0.202 \\ -0.300 & 0.800 \\ 0.136 & -0.382 \end{bmatrix}$$

$$b_{\text{new}}^{[1]} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.043 \\ 0.000 \\ -0.361 \end{bmatrix} = \begin{bmatrix} 0.104 \\ -0.200 \\ 0.336 \end{bmatrix}$$

Computational Graph: Weight Update Step

Layer 2 Updates



Layer 1 Updates

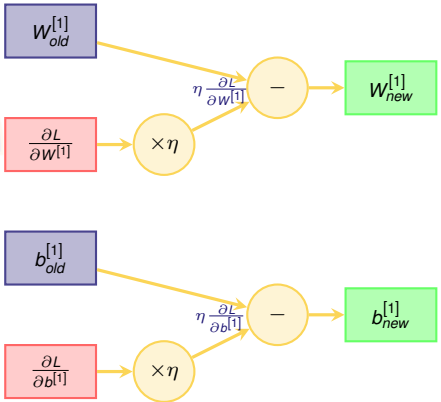


Table of Contents

Sigmoid Function: Definition and Properties

The sigmoid function maps any real number to the interval $(0, 1)$.

Definition:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

Properties:

- Output range: $(0, 1)$
- Smooth and differentiable everywhere
- $\sigma(0) = 0.5$
- $\sigma(-z) = 1 - \sigma(z)$ (symmetry)
- As $z \to \infty$: $\sigma(z) \to 1$
- As $z \to -\infty$: $\sigma(z) \to 0$

Use cases:

- Binary classification (output layer)
- Gate mechanisms in LSTMs

Key values:

- $\sigma(-2) \approx 0.12$
- $\sigma(0) = 0.5$
- $\sigma(2) \approx 0.88$

Sigmoid Function: Derivative Derivation

Goal: Find $\frac{d\sigma}{dz}$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

Step 1: Rewrite and apply chain rule

$$\sigma(z) = (1 + e^{-z})^{-1} \quad \Rightarrow \quad \frac{d\sigma}{dz} = -(1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2} \tag{2}$$

Step 2: Factor to express in terms of $\sigma(z)$

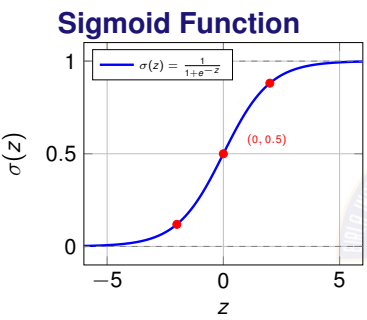
$$\frac{d\sigma}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} \tag{3}$$

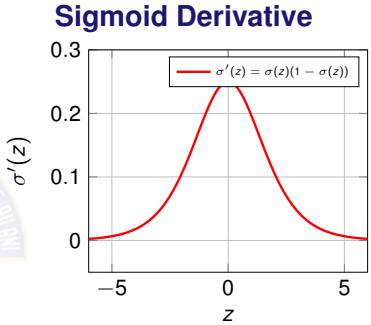
Step 3: Recognize that $\sigma(z) = \frac{1}{1+e^{-z}}$ and $1 - \sigma(z) = \frac{e^{-z}}{1+e^{-z}}$

$$\frac{d\sigma}{dz} = \sigma(z) \cdot (1 - \sigma(z)) \tag{4}$$

Beautiful property: Derivative computed using only the function value itself!

Sigmoid Function: Visualization





Key observations:

- Sigmoid squashes inputs to $(0, 1)$ - ideal for probabilities
- Derivative peaks at $z = 0$ with value 0.25
- Derivative vanishes for large $|z|$ (vanishing gradient problem!)

Softmax Function: Definition

The softmax function converts a vector of real numbers into a probability distribution.

For input vector $\mathbf{z} = [z_1, z_2, \ldots, z_C]^T$:

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}} \quad \text{for } j = 1, 2, \ldots, C \tag{5}$$

Properties:

- All outputs in $(0, 1)$: $0 < \text{softmax}(\mathbf{z})_j < 1$
- Outputs sum to 1: $\sum_{j=1}^{C} \text{softmax}(\mathbf{z})_j = 1$
- Monotonic: larger $z_j \rightarrow$ larger output
- Translation invariant: $\text{softmax}(\mathbf{z} + c) = \text{softmax}(\mathbf{z})$

Use case: Multi-class classification output layer

Softmax Function: Notation

Let's denote the softmax outputs as:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}} \quad \text{where} \quad \hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_C]^T \tag{6}$$

Example: $\mathbf{z} = [1.0, 2.0, 3.0]^T$

$$\hat{y}_1 = \frac{e^{1.0}}{e^{1.0} + e^{2.0} + e^{3.0}} = \frac{2.718}{2.718 + 7.389 + 20.086} \approx 0.090$$

$$\hat{y}_2 = \frac{e^{2.0}}{e^{1.0} + e^{2.0} + e^{3.0}} = \frac{7.389}{30.193} \approx 0.245$$

$$\hat{y}_3 = \frac{e^{3.0}}{e^{1.0} + e^{2.0} + e^{3.0}} = \frac{20.086}{30.193} \approx 0.665$$

Verify: $0.090 + 0.245 + 0.665 = 1.0$ ✓

Softmax + Cross-Entropy: Beautiful

Cross-entropy Loss: $\ell = -\sum_c y_c \log(\hat{y}_c)$ where true class is k (7)

Chain rule:
$$\frac{\partial \ell}{\partial z_j} = \sum_{i=1}^{C} \frac{\partial \ell}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_j} \tag{8}$$

$$\frac{\partial \ell}{\partial z_j} = -\frac{1}{\hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z_j} \quad \text{where} \quad \frac{\partial \ell}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \quad y_k = 1 \tag{9}$$

$$= -\frac{1}{\hat{y}_k}[\hat{y}_k(\delta_{kj} - \hat{y}_j)] \tag{10}$$

$$= -(\delta_{kj} - \hat{y}_j) \tag{11}$$

$$= \hat{y}_j - \delta_{kj} = \hat{y}_j - y_j \tag{12}$$

Tanh Function: Definition and Properties

The hyperbolic tangent (tanh) maps real numbers to the interval $(-1, 1)$.

Definition:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{13}$$

Properties:

- Output range: $(-1, 1)$
- Zero-centered: $\tanh(0) = 0$
- Odd function: $\tanh(-z) = -\tanh(z)$
- As $z \to \pm\infty$: $\tanh(z) \to \pm 1$

Advantages over sigmoid:

- Zero-centered
- Stronger gradients (max = 1)
- Faster convergence

Key values:

- $\tanh(-1) \approx -0.76$
- $\tanh(0) = 0$
- $\tanh(1) \approx 0.76$

Tanh Function: Derivative Derivation

Quotient Rule Method (where $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$)

Let $u = e^z - e^{-z}$, $v = e^z + e^{-z}$, so $u' = e^z + e^{-z}$, $v' = e^z - e^{-z}$

$$\frac{d\tanh}{dz} = \frac{u'v - uv'}{v^2} = \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2} = \frac{4}{(e^z + e^{-z})^2} \tag{14}$$

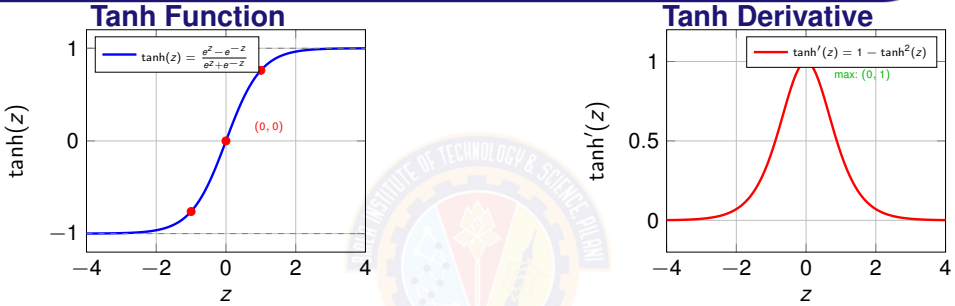
Note: $(e^z + e^{-z})^2 - (e^z - e^{-z})^2 = (e^{2z} + 2 + e^{-2z}) - (e^{2z} - 2 + e^{-2z}) = 4$

Tanh Derivative

$$\boxed{\tanh'(z) = 1 - \tanh^2(z) = \frac{4}{(e^z + e^{-z})^2}} \tag{15}$$

Beautiful property: Derivative computed using only the function value! Maximum at $z = 0$: $\tanh'(0) = 1$

Tanh Function: Visualization



Key observations:

- Tanh is zero-centered (unlike sigmoid) - better for hidden layers
- Derivative peaks at $z = 0$ with value 1 (4× larger than sigmoid!)
- Derivative vanishes for large $|z|$ but slower than sigmoid
- Symmetric derivative (even function): $\tanh'(-z) = \tanh'(z)$

Numerical Problem: Activation Functions

Problem: Given the input vector $z = \begin{bmatrix} 2.0 \\ -1.5 \\ 0.8 \\ -0.3 \end{bmatrix}$, calculate the output for each activation function. Also compute the derivatives for each function at the given input.

1. Sigmoid: $\sigma(z)$
2. Tanh: $\tanh(z)$
3. ReLU: $\text{ReLU}(z)$

Solution: Sigmoid Activation

Sigmoid Function: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$

$$\sigma(z) = \begin{bmatrix} \frac{1}{1+e^{-2.0}} \\ \frac{1}{1+e^{1.5}} \\ \frac{1}{1+e^{-0.8}} \\ \frac{1}{1+e^{0.3}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+0.135} = 0.881 \\ \frac{1}{1+4.482} = 0.182 \\ \frac{1}{1+0.449} = 0.690 \\ = \frac{1}{1+1.350} = 0.426 \end{bmatrix} = \begin{bmatrix} 0.881 \\ 0.182 \\ 0.690 \\ 0.426 \end{bmatrix}$$

Sigmoid Derivative: $\sigma'(z) = \begin{bmatrix} 0.881(1 - 0.881) \\ 0.182(1 - 0.182) \\ 0.690(1 - 0.690) \\ 0.426(1 - 0.426) \end{bmatrix} = \begin{bmatrix} 0.105 \\ 0.149 \\ 0.214 \\ 0.245 \end{bmatrix}$

Solution: Tanh Activation

Tanh Function: $\tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$

$$\tanh(z) = \begin{bmatrix} \tanh(2.0) \\ \tanh(-1.5) \\ \tanh(0.8) \\ \tanh(-0.3) \end{bmatrix} = \begin{bmatrix} \frac{7.389-0.135}{7.389+0.135} = 0.964 \\ \frac{0.223-4.482}{0.223+4.482} = -0.905 \\ \frac{2.226-0.449}{2.226+0.449} = 0.664 \\ \frac{0.741-1.350}{0.741+1.350} = -0.291 \end{bmatrix} = \begin{bmatrix} 0.964 \\ -0.905 \\ 0.664 \\ -0.291 \end{bmatrix}$$

Tanh Derivative: $\tanh'(z) = \begin{bmatrix} 1 - 0.964^2 \\ 1 - (-0.905)^2 \\ 1 - 0.664^2 \\ 1 - (-0.291)^2 \end{bmatrix} = \begin{bmatrix} 0.070 \\ 0.181 \\ 0.559 \\ 0.915 \end{bmatrix}$

Solution: ReLU

ReLU Function: $\text{ReLU}(z) = \max(0, z)$

$$= \begin{bmatrix} \max(0, 2.0) \\ \max(0, -1.5) \\ \max(0, 0.8) \\ \max(0, -0.3) \end{bmatrix} = \begin{bmatrix} 2.0 \\ 0.0 \\ 0.8 \\ 0.0 \end{bmatrix}$$

ReLU Derivative: $\text{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Numerical Problem: Softmax Function

Problem: Given the logit vector z, compute:

1. The softmax output
2. The cross-entropy loss with true label y
3. The gradient of the loss with respect to the logits

$$z = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \\ -1.0 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Cross-entropy Loss:

$$L = -\sum_{i=1}^{n} y_i \log(\text{softmax}(z_i))$$

Solution: Softmax Computation

Step 1: Compute softmax values First, for numerical stability, subtract the maximum:

$$z_{\text{stable}} = z - \max(z) = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \\ -1.0 \end{bmatrix} - 2.0 = \begin{bmatrix} 0.0 \\ -1.0 \\ -1.9 \\ -3.0 \end{bmatrix}$$

$$e^{z_{\text{stable}}} = \begin{bmatrix} e^{0.0} \\ e^{-1.0} \\ e^{-1.9} \\ e^{-3.0} \end{bmatrix} = \begin{bmatrix} 1.000 \\ 0.368 \\ 0.150 \\ 0.050 \end{bmatrix}$$

$$\sum_{j=1}^{4} e^{z_j - \max(z)} = 1.000 + 0.368 + 0.150 + 0.050 = 1.568$$

Solution: Softmax Computation

$$\text{softmax}(z) = \frac{1}{1.568}\begin{bmatrix}1.000\\0.368\\0.150\\0.050\end{bmatrix} = \begin{bmatrix}0.638\\0.235\\0.096\\0.032\end{bmatrix}$$

$$\text{Verification: } = 0.638 + 0.235 + 0.096 + 0.032 = 1.001 \approx 1$$

Solution: Cross-entropy Loss and Gradient

Step 2: Compute cross-entropy loss

$$L = -\sum_{i=1}^{4} y_i \log(\text{softmax}(z_i))$$

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ only the second term contributes

$$L = -y_2 \log(\text{softmax}(z_2)) = -1 \times \log(0.235) = -\log(0.235) = 1.447$$

Solution: Cross-entropy Loss and Gradient

Step 3: Compute gradient For cross-entropy loss with softmax, the gradient is:

$$\frac{\partial L}{\partial z_i} = \text{softmax}(z_i) - y_i$$

$$\frac{\partial L}{\partial z} = \begin{bmatrix} 0.638 \\ 0.235 \\ 0.096 \\ 0.032 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.638 \\ -0.765 \\ 0.096 \\ 0.032 \end{bmatrix}$$

This result shows that the gradient is simply the difference between predicted and true probabilities!

Table of Contents

Numerical Problem: Gradient Flow Analys

Problem: Consider a 4-layer network with sigmoid activations. Analyze gradient flow through the network.

Given:

- All weight matrices have spectral norm $\|W^{[l]}\| = 0.8$
- Sigmoid derivatives at each layer: $g'^{[l]} = 0.25$ (maximum sigmoid derivative)
- Initial gradient magnitude at output: $|\delta^{[4]}| = 1.0$

Calculate the gradient magnitude at each layer and discuss the implications.

Gradient Propagation Formula:

$$|\delta^{[l-1]}| = |\delta^{[l]}| \cdot \|W^{[l]}\| \cdot |g'^{[l-1]}|$$

Solution: Gradient Flow Analysis

Layer-by-layer gradient magnitude:

$$\textbf{Layer 4 (Output):} |\delta^{[4]}| = 1.0 \quad \text{(given)}$$

$$\textbf{Layer 3:} |\delta^{[3]}| = |\delta^{[4]}| \cdot \|W^{[4]}\| \cdot |g'^{[3]}| = 1.0 \times 0.8 \times 0.25 = 0.2$$

$$\textbf{Layer 2:} |\delta^{[2]}| = |\delta^{[3]}| \cdot \|W^{[3]}\| \cdot |g'^{[2]}| = 0.2 \times 0.8 \times 0.25 = 0.04$$

$$\textbf{Layer 1:} |\delta^{[1]}| = |\delta^{[2]}| \cdot \|W^{[2]}\| \cdot |g'^{[1]}| = 0.04 \times 0.8 \times 0.25 = 0.008$$

Analysis:

- Gradient magnitude decreases by factor of $0.8 \times 0.25 = 0.2$ per layer
- After 3 backward steps: gradient is reduced to 0.8% of original
- First layer receives extremely weak learning signal
- This demonstrates the vanishing gradient problem in deep sigmoid networks

Table of Contents

Example 1: Small Tabular Dataset

Problem Setup: Compute the number of layers and number of neurons in each layer

- $d = 20$ features (customer data)
- $M = 800$ training samples
- $c = 3$ classes (customer segments)

Example 1: Small Tabular Dataset

- Step 1: Apply constraints
  $L \leq \log_2(800/(20 \times 3)) + 1 = \log_2(13.33) + 1 \approx 4.7 \Rightarrow L \leq 4$
- Step 2: Choose starting architecture Since $M < 1000$, start with $L = 2$:
  $[20, 40, 3]$
- Step 3: Parameter count check Parameters
  $= 20 \times 40 + 40 + 40 \times 3 + 3 = 800 + 40 + 120 + 3 = 963$
  Constraint: $963 \leq 800/5 = 160$ ✗ Too many parameters!
- Step 4: Reduce complexity
  - Try $[20, 20, 3]$: Parameters = $20 \times 20 + 20 + 20 \times 3 + 3 = 483$ ✗ Still too many!
  - Try $[20, 10, 3]$: Parameters = $20 \times 10 + 10 + 10 \times 3 + 3 = 243$ ✗
  - Try $[20, 6, 3]$: Parameters = $20 \times 6 + 6 + 6 \times 3 + 3 = 147$ ✔
- Final recommendation: $[20, 6, 3]$ with regularization

Example 2: Medium Image Dataset

Problem Setup: Compute the number of layers and number of neurons in each layer

- $d = 784$ features (28×28 grayscale images)
- $M = 5000$ training samples
- $c = 10$ classes (digit classification)

Example 2: Medium Image Dataset

- **Step 1: Relaxed approach**
  For medium datasets, start with $L = 3$: $[784, 2352, 784, 10]$
- **Step 2: Parameter count check**
  Parameters = $784 \times 2352 + 2352 + 2352 \times 784 + 784 + 784 \times 10 + 10 \approx 3.7M$
  Constraint $5000/5 = 1000$ ✗ Way too many!
- **Step 3: Practical architecture**
  - Try $[784, 128, 64, 10]$:
    Parameters = $784 \times 128 + 128 + 128 \times 64 + 64 + 64 \times 10 + 10 = 109,000$ ✗
  - Try $[784, 64, 32, 10]$:
    Parameters = $784 \times 64 + 64 + 64 \times 32 + 32 + 32 \times 10 + 10 = 52,458$ ✗
  - Try $[784, 32, 10]$: Parameters = $784 \times 32 + 32 + 32 \times 10 + 10 = 25,450$ ✗
  - Try $[784, 16, 10]$: Parameters = $784 \times 16 + 16 + 16 \times 10 + 10 = 12,714$ ✗
- **Reality check:** Need to use regularization techniques (dropout, L2) with larger architectures.

Example 3: Large Text Dataset

Problem Setup: Compute the number of layers and number of neurons in each layer

- $d = 1000$ features (TF-IDF vectors)
- $M = 50000$ training samples
- $c = 5$ classes (sentiment categories)

Example 3: Large Text Dataset

- Step 1: Apply constraints
  $L \leq \log_2(50000/(1000 \times 5)) + 1 = \log_2(10) + 1 \approx 4.3 \Rightarrow L \leq 4$
- Step 2: Choose architecture
  Since $M \geq 10000$, can use $L = 4$: $[1000, 5000, 2000, 1000, 5]$
- Step 3: Parameter count check Parameters =
  $1000 \times 5000 + 5000 + 5000 \times 2000 + 2000 + 2000 \times 1000 + 1000 + 1000 \times 5 + 5 =$
  $5M + 10M + 2M + 5K = 17M$
  Constraint: $17M \leq 50000/5 = 10000$ ✗ Way too many!
- Step 4: Reasonable architecture
  Try $[1000, 512, 256, 128, 5]$:
  Parameters = $1000 \times 512 + 512 + 512 \times 256 + 256 + 256 \times 128 + 128 + 128 \times 5 + 5$
  $= 512K + 131K + 32K + 640 = 675K$ parameters ✔
  This is $675000/50000 = 13.5$ parameters per sample, which is reasonable.
- Final recommendation: $[1000, 512, 256, 128, 5]$ with batch normalization and dropout.

Table of Contents

Overview: Loss Functions

Loss functions measure the discrepancy between predicted and actual values.

We will include:

1. Mean Squared Error (MSE) Loss
2. Mean Absolute Error (MAE) Loss
3. Binary Cross-Entropy (BCE) Loss
4. Cross-Entropy Loss

Key Goal: Understand how to compute gradients for backpropagation

MSE Loss: Definition

Mean Squared Error (MSE) is commonly used for regression tasks.

$$\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \tag{16}$$

Properties:

- Always non-negative: $\ell \geq 0$
- Convex function (for linear models)
- Differentiable everywhere
- Penalizes large errors quadratically
- Factor of $\frac{1}{2}$ simplifies derivative

MSE Loss: Derivative Derivation

Goal: Find $\frac{\partial \ell}{\partial \hat{y}}$ where $\ell = \frac{1}{2}(\hat{y} - y)^2$

Let $u = \hat{y} - y$, so $\ell = \frac{1}{2}u^2$

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial u} \cdot \frac{\partial u}{\partial \hat{y}} \quad \text{(chain rule)} \tag{17}$$

$$= \left(\frac{1}{2} \cdot 2u\right) \cdot (1) \quad \text{(compute derivatives)} \tag{18}$$

$$= u = \hat{y} - y \quad \text{(substitute back)} \tag{19}$$

MSE Loss Derivative:

$$\boxed{\frac{\partial \ell}{\partial \hat{y}} = \hat{y} - y} \tag{20}$$

MSE Loss: Numerical Example

Example 1: $y = 3$, $\hat{y} = 5$

$$\ell = \frac{1}{2}(5-3)^2 = \frac{1}{2}(2)^2 = 2$$

$$\frac{\partial \ell}{\partial \hat{y}} = 5 - 3 = 2$$

Example 2: $y = 8$, $\hat{y} = 5$

$$\ell = \frac{1}{2}(5-8)^2 = \frac{1}{2}(-3)^2 = 4.5$$

$$\frac{\partial \ell}{\partial \hat{y}} = 5 - 8 = -3$$

Note: Gradient is negative, telling us to *increase* $\hat{y}$

MAE Loss: Definition

Mean Absolute Error (MAE) is more robust to outliers than MSE.

$$\ell(\hat{y}, y) = |\hat{y} - y| \tag{21}$$

Properties:

- Always non-negative: $\ell \geq 0$
- Less sensitive to outliers than MSE
- Not differentiable at $\hat{y} = y$
- Linear penalty for errors

MAE Loss: Derivative Derivation

Goal: Find $\frac{\partial \ell}{\partial \hat{y}}$ where $\ell = |\hat{y} - y|$

Recall: Derivative of absolute value function

$$\frac{d|x|}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0 \end{cases} \tag{22}$$

Apply to $\ell = |\hat{y} - y|$ using chain rule:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}|\hat{y} - y| = \begin{cases} 1 & \text{if } \hat{y} > y \\ -1 & \text{if } \hat{y} < y \\ 0 & \text{if } \hat{y} = y \text{ (convention)} \end{cases} = \text{sign}(\hat{y} - y) \tag{23}$$

MAE Loss Derivative:

$$\boxed{\frac{\partial \ell}{\partial \hat{y}} = \text{sign}(\hat{y} - y)} \tag{24}$$

MAE Loss: Numerical Examples

Example 1: $y = 3$, $\hat{y} = 5$

$$\ell = |5 - 3| = 2$$
$$\frac{\partial \ell}{\partial \hat{y}} = \text{sign}(5 - 3) = \text{sign}(2) = +1$$

Example 2: $y = 8$, $\hat{y} = 5$

$$\ell = |5 - 8| = 3$$
$$\frac{\partial \ell}{\partial \hat{y}} = \text{sign}(5 - 8) = \text{sign}(-3) = -1$$

Comparison with MSE:

- MAE gradient is constant: ± 1
- MSE gradient varies with error magnitude: $\hat{y} - y$

BCE Loss: Definition

Binary Cross-Entropy (BCE) is used for binary classification where $y \in \{0, 1\}$ and $\hat{y} \in (0, 1)$.

For a single example:

$$\ell(\hat{y}, y) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \tag{25}$$

Properties:

- Measures divergence between predicted and true probability distributions
- Always non-negative: $\ell \geq 0$
- Heavily penalizes confident wrong predictions

BCE Loss: Understanding the Formula

Why this form?

Since $y \in \{0, 1\}$, the loss simplifies based on the true label:

Case 1: When $y = 1$

$$\ell = -[1 \cdot \log(\hat{y}) + (1 - 1)\log(1 - \hat{y})] \tag{26}$$

$$= -\log(\hat{y}) \tag{27}$$

Case 2: When $y = 0$

$$\ell = -[0 \cdot \log(\hat{y}) + (1 - 0)\log(1 - \hat{y})] \tag{28}$$

$$= -\log(1 - \hat{y}) \tag{29}$$

Intuition:

- If $y = 1$, want $\hat{y} \to 1$ to minimize $-\log(\hat{y})$
- If $y = 0$, want $\hat{y} \to 0$ to minimize $-\log(1 - \hat{y})$

BCE Loss: Derivative Derivation

Goal: Find $\frac{\partial \ell}{\partial \hat{y}}$ where $\ell = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$

Differentiate both terms (using $\frac{d}{dx}\log(x) = \frac{1}{x}$ and chain rule):

$$\frac{\partial \ell}{\partial \hat{y}} = -\frac{\partial}{\partial \hat{y}}[y \log(\hat{y})] - \frac{\partial}{\partial \hat{y}}[(1-y)\log(1-\hat{y})] \tag{30}$$

$$= -\frac{y}{\hat{y}} - (1-y) \cdot \frac{1}{1-\hat{y}} \cdot (-1) = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \tag{31}$$

Simplify with common denominator:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{-y(1-\hat{y}) + (1-y)\hat{y}}{\hat{y}(1-\hat{y})} = \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1-\hat{y})} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \tag{32}$$

BCE Loss Derivative:

$$\boxed{\frac{\partial \ell}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}} \tag{33}$$

BCE Loss: Derivative Examples

Example 1: $y = 1, \hat{y} = 0.9$

$$\ell = -\log(0.9) \approx 0.105$$

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{0.9 - 1}{0.9(1 - 0.9)} = \frac{-0.1}{0.09} \approx -1.11$$

Example 2: $y = 1, \hat{y} = 0.1$

$$\ell = -\log(0.1) \approx 2.303$$

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{0.1 - 1}{0.1(1 - 0.1)} = \frac{-0.9}{0.09} = -10$$

Example 3: $y = 0, \hat{y} = 0.9$

$$\ell = -\log(0.1) \approx 2.303$$

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{0.9 - 0}{0.9(1 - 0.9)} = \frac{0.9}{0.09} = 10$$

Cross-Entropy Loss: Definition

Cross-Entropy Loss generalizes BCE to multi-class classification where y is one-hot encoded and $\hat{\mathbf{y}}$ is a probability distribution.

For C classes and single example:

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{34}$$

where:

- $\mathbf{y} = [y_1, y_2, \ldots, y_C]^T$ is one-hot encoded (one $y_c = 1$, rest are 0)
- $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_C]^T$ are predicted probabilities
- $\sum_{c=1}^{C} \hat{y}_c = 1$ (output of softmax)

Cross-Entropy Loss: Derivative Derivation

Goal: Find $\frac{\partial \ell}{\partial \hat{y}_j}$ for each class j, where $\ell = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$

Step 1: Take derivative with respect to $\hat{y}_j$

$$\frac{\partial \ell}{\partial \hat{y}_j} = -\frac{\partial}{\partial \hat{y}_j}\left[\sum_{c=1}^{C} y_c \log(\hat{y}_c)\right] \tag{35}$$

Step 2: Only the $c = j$ term depends on $\hat{y}_j$

$$\frac{\partial \ell}{\partial \hat{y}_j} = -\frac{\partial}{\partial \hat{y}_j}\left[y_j \log(\hat{y}_j)\right] = -y_j \cdot \frac{1}{\hat{y}_j} \tag{36}$$

Cross-Entropy Loss

$$\boxed{\frac{\partial \ell}{\partial \hat{y}_j} = -\frac{y_j}{\hat{y}_j}} \tag{37}$$

Cross-Entropy Loss: Gradient Vector

The gradient with respect to all outputs forms a vector:

$$\nabla_{\hat{\mathbf{y}}} \ell = \begin{bmatrix} \frac{\partial \ell}{\partial \hat{y}_1} \\ \frac{\partial \ell}{\partial \hat{y}_2} \\ \vdots \\ \frac{\partial \ell}{\partial \hat{y}_C} \end{bmatrix} = \begin{bmatrix} -\frac{y_1}{\hat{y}_1} \\ -\frac{y_2}{\hat{y}_2} \\ \vdots \\ -\frac{y_C}{\hat{y}_C} \end{bmatrix} \tag{38}$$

Since $\mathbf{y}$ is one-hot (only $y_k = 1$):

- For the true class k: $\frac{\partial \ell}{\partial \hat{y}_k} = -\frac{1}{\hat{y}_k}$
- For all other classes $j \neq k$: $\frac{\partial \ell}{\partial \hat{y}_j} = 0$

Important: This is the gradient w.r.t. softmax outputs. When combined with softmax derivative, we get a simpler form!

Table of Contents

Activation Functions: Comparison

Property	Sigmoid	Tanh	Softmax
Formula	$\frac{1}{1+e^{-z}}$	$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$\frac{e^{z_j}}{\sum_k e^{z_k}}$
Range	$(0, 1)$	$(-1, 1)$	$(0, 1)$
Zero-centered?	No	Yes	No
Derivative	$\sigma(1-\sigma)$	$1 - \tanh^2$	Complex
Max Derivative	0.25	1.0	Varies
Output Sum	N/A	N/A	1.0
Use Case	Binary class	Hidden layers	Multi-class

Key Advantages:

- **Sigmoid:** Output is probability for binary classification
- **Tanh:** Zero-centered, stronger gradients
- **Softmax:** Outputs form probability distribution

Loss Functions: Comparison

Loss	Formula	Derivative
MSE	$\frac{1}{2}(\hat{y} - y)^2$	$\hat{y} - y$
MAE	$\lvert\hat{y} - y\rvert$	$\text{sign}(\hat{y} - y)$
BCE	$-[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$	$\frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$
Cross-Entropy	$-\sum_c y_c \log \hat{y}_c$	$-\frac{y_j}{\hat{y}_j}$

Use Cases:

- **MSE:** Regression tasks
- **MAE:** Regression with outliers
- **BCE:** Binary classification ($y \in \{0, 1\}$)
- **Cross-Entropy:** Multi-class classification

Special case: Softmax + Cross-Entropy gradient: $\hat{y}_j - y_j$ (very clean!)

Summary: Key Takeaways

Loss Functions:

- MSE penalizes errors quadratically, derivative is linear
- MAE is robust to outliers, derivative is constant ± 1
- BCE for binary classification, heavily penalizes confident errors
- Cross-Entropy for multi-class, only cares about correct class probability

Activation Functions:

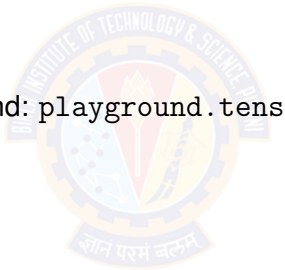
- Sigmoid squashes to $(0, 1)$, useful for probabilities
- Tanh squashes to $(-1, 1)$, zero-centered is better for hidden layers
- Softmax converts logits to probability distribution

Practical Tips:

- Regression: Use MSE or MAE loss
- Binary classification: Use sigmoid + BCE
- Multi-class: Use softmax + cross-entropy

Try

- Neural Network Playground: `playground.tensorflow.org`



Thank You!