Neural Networks Class 2

Multi-layer Perceptrons & Backpropagation Theory

Learning Objectives:

- Understand limitations of single perceptron and MLP solutions
- Master forward propagation mathematics
- Learn backpropagation derivation and gradient computation

Limitations of Single Perceptron

XOR is not linearly separable:

- No single hyperplane can separate XOR classes, Single perceptron cannot solve XOR

Mathematical Impossibility:

For XOR with inputs (x_1, x_2) and output y:

- $(0, 0) \to 0$: requires $b < 0$
- $(0, 1) \to 1$: requires $w_2 + b > 0$
- $(1, 0) \to 1$: requires $w_1 + b > 0$
- $(1, 1) \to 0$: requires $w_1 + w_2 + b < 0$

Contradiction: Conditions 2 & 3 imply $w_1 + w_2 > -2b > 0$, but condition 4 requires $w_1 + w_2 < -b$

Multi-layer Architecture

The Solution: Hidden Layers

Key Insight: Multiple layers can create non-linear decision boundaries

MLP Architecture:

- **Input layer:** Features $\mathbf{x}$
- **Hidden layer(s):** Intermediate representations
- **Output layer:** Final predictions

Multi-layer Architecture

The Solution: Hidden Layers

XOR Solution with 2-layer MLP:

- Hidden layer creates two half-planes
- Output layer combines half-planes
- Result: Non-linear decision boundary

Mathematical Power: Multiple layers enable universal approximation

Universal Approximation Theorem

Theoretical Foundation

Theorem Statement:
A feedforward network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy.

Key Points:

- **Existence guarantee:** Solution exists for any continuous function
- **Not constructive:** Doesn't tell us how to find the solution
- **Practical limitations:** May require exponentially many neurons

Universal Approximation Theorem

Theoretical Foundation

Implications:

- MLPs are theoretically powerful
- Practical success depends on learning algorithm
- Deep networks may be more efficient than wide networks

Forward Propagation - Matrix Formulation

General Layer Equation:

$$\mathbf{A}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{A}^{(l-1)} + \mathbf{b}^{(l)})$$

Notation:

- $\mathbf{A}^{(l)}$: Activations at layer l
- $\mathbf{W}^{(l)}$: Weight matrix from layer $l-1$ to l
- $\mathbf{b}^{(l)}$: Bias vector at layer l
- $\sigma(\cdot)$: Element-wise activation function
- $\mathbf{A}^{(0)} = \mathbf{x}$: Input

Pre-activation: $\mathbf{Z}^{(l)} = \mathbf{W}^{(l)}\mathbf{A}^{(l-1)} + \mathbf{b}^{(l)}$

Forward Propagation

Detailed Computation Chain

2-Layer MLP Example:

Layer 1 (Hidden):

$$\mathbf{Z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{A}^{(1)} = \sigma(\mathbf{Z}^{(1)})$$

Layer 2 (Output):

$$\mathbf{Z}^{(2)} = \mathbf{W}^{(2)}\mathbf{A}^{(1)} + \mathbf{b}^{(2)}$$
$$\mathbf{A}^{(2)} = \sigma(\mathbf{Z}^{(2)}) = \hat{\mathbf{y}}$$

Forward Propagation

Detailed Computation Chain

Complete Forward Pass:

$$\hat{\mathbf{y}} = \sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

Dimensions and Matrix Shapes

Keeping Track of Sizes

For layer l:

- Input dimension: $n^{(l-1)}$
- Output dimension: $n^{(l)}$
- Weight matrix: $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$
- Bias vector: $\mathbf{b}^{(l)} \in \mathbb{R}^{n^{(l)}}$
- Activations: $\mathbf{A}^{(l)} \in \mathbb{R}^{n^{(l)}}$

Dimensions and Matrix Shapes

Keeping Track of Sizes

Example: 3-2-1 Network

- Input: $\mathbf{x} \in \mathbb{R}^3$
- $\mathbf{W}^{(1)} \in \mathbb{R}^{2\times 3}, \mathbf{b}^{(1)} \in \mathbb{R}^2$
- $\mathbf{W}^{(2)} \in \mathbb{R}^{1\times 2}, \mathbf{b}^{(2)} \in \mathbb{R}^1$
- Output: $\hat{y} \in \mathbb{R}^1$

Loss Functions

Mean Squared Error (MSE)

For Regression Problems:

$$L(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

Single Example:

$$L = \frac{1}{2} (\hat{y} - y)^2$$

Loss Functions

Mean Squared Error (MSE)

Properties:

- **Convex** for linear models
- **Smooth** everywhere
- **Large penalty** for large errors
- **Derivative:** $\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$

Loss Functions

Cross-Entropy Loss

For Binary Classification:

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

For Multi-class Classification:

$$L = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

Loss Functions

Cross-Entropy Loss

Properties:

- **Probabilistic interpretation**
- **Convex** in parameters
- **Derivative:** $\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}$ (binary)

Advantage: Faster convergence than MSE for classification

Backpropagation: The Chain Rule

Mathematical Foundation

Goal: Compute $\frac{\partial L}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial L}{\partial \mathbf{b}^{(l)}}$ for all layers

Chain Rule Application:

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \frac{\partial L}{\partial \mathbf{Z}^{(l)}} \frac{\partial \mathbf{Z}^{(l)}}{\partial \mathbf{W}^{(l)}}$$

Key Insight: Errors propagate backward through the network

Backpropagation: The Chain Rule

Mathematical Foundation

Define Error Signal:

$$\boldsymbol{\delta}^{(l)} = \frac{\partial L}{\partial \mathbf{Z}^{(l)}}$$

Recursive Relationship:

$$\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \odot \sigma'(\mathbf{Z}^{(l)})$$

Backpropagation: Step-by-Step

Output Layer

Step 1: Output Layer Error

For MSE loss and sigmoid output:

$$\boldsymbol{\delta}^{(L)} = \frac{\partial L}{\partial \mathbf{Z}^{(L)}} = \frac{\partial L}{\partial \mathbf{A}^{(L)}} \frac{\partial \mathbf{A}^{(L)}}{\partial \mathbf{Z}^{(L)}}$$
$$= (\mathbf{A}^{(L)} - \mathbf{y}) \odot \sigma'(\mathbf{Z}^{(L)})$$

For Sigmoid: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Simplified for Sigmoid + MSE:

$$\boldsymbol{\delta}^{(L)} = (\mathbf{A}^{(L)} - \mathbf{y}) \odot \mathbf{A}^{(L)} \odot (1 - \mathbf{A}^{(L)})$$

Backpropagation: Step-by-Step

Hidden Layers

Step 2: Hidden Layer Errors

$$\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \odot \sigma'(\mathbf{Z}^{(l)})$$

Intuition:

- $(\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}$: Error from next layer weighted by connections
- $\sigma'(\mathbf{Z}^{(l)})$: Local gradient of activation function
- $\odot$: Element-wise multiplication (Hadamard product)

Propagation Direction: Errors flow backward from output to input

Backpropagation: Step-by-Step

Step 3: Weight Gradients

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)} (\mathbf{A}^{(l-1)})^T$$

Step 4: Bias Gradients

$$\frac{\partial L}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$$

Matrix Dimensions Check:

- $\boldsymbol{\delta}^{(l)} : n^{(l)} \times 1$
- $\mathbf{A}^{(l-1)} : n^{(l-1)} \times 1$
- $\frac{\partial L}{\partial \mathbf{W}^{(l)}} : n^{(l)} \times n^{(l-1)}$ ✓

Complete Backpropagation Algorithm

2-Layer Example Walkthrough

Network: Input → Hidden (3 neurons) → Output (1 neuron)

Forward Pass:

1. $\mathbf{Z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$
2. $\mathbf{A}^{(1)} = \sigma(\mathbf{Z}^{(1)})$
3. $Z^{(2)} = \mathbf{W}^{(2)}\mathbf{A}^{(1)} + b^{(2)}$
4. $\hat{y} = \sigma(Z^{(2)})$
5. $L = \frac{1}{2}(\hat{y} - y)^2$

Complete Backpropagation Algorithm

2-Layer Example Walkthrough

Backward Pass:

1. $\delta^{(2)} = (\hat{y} - y) \cdot \sigma'(Z^{(2)})$
2. $\boldsymbol{\delta}^{(1)} = \mathbf{W}^{(2)T}\delta^{(2)} \odot \boldsymbol{\sigma}'(\mathbf{Z}^{(1)})$
3. $\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \delta^{(2)}(\mathbf{A}^{(1)})^T$
4. $\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \boldsymbol{\delta}^{(1)}\mathbf{x}^T$

Gradient Descent Update

Putting It All Together

Parameter Updates:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial L}{\partial \mathbf{W}^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial L}{\partial \mathbf{b}^{(l)}}$$

Gradient Descent Update

Putting It All Together

Complete Training Loop:

1. **Forward pass:** Compute predictions and loss
2. **Backward pass:** Compute gradients via backpropagation
3. **Update:** Adjust parameters using gradient descent
4. **Repeat:** Until convergence or max iterations

Learning Rate η: Controls step size in parameter space

Why Backpropagation Works

Computational Efficiency

Naive Approach: Compute each partial derivative independently

- **Time complexity:** $O(mn^2)$ for m parameters and n neurons
- **Computationally prohibitive** for large networks

Backpropagation Advantage:

- **Reuses computations** through chain rule
- **Time complexity:** $O(mn)$ - linear in parameters
- **Memory efficient:** Only store activations and errors

Key Insight: Dynamic programming applied to gradient computation

Common Activation Function Derivatives

Sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Tanh:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \quad \tanh'(z) = 1 - \tanh^{2}(z)$$

ReLU:

$$\mathrm{ReLU}(z) = \max(0, z), \quad \mathrm{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Implementation Tip: Compute derivatives using forward pass values

Summary: Class 2 Key Concepts

MLP Architecture:

- Multiple layers solve XOR and other non-linear problems
- Universal approximation theorem provides theoretical foundation

Forward Propagation:

- Matrix formulation: $\mathbf{A}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{A}^{(l-1)} + \mathbf{b}^{(l)})$
- Chain of computations from input to output

Backpropagation:

- Efficient gradient computation using chain rule
- Error signals propagate backward: $\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)})^T\boldsymbol{\delta}^{(l+1)} \odot \sigma'(\mathbf{Z}^{(l)})$
- Weight gradients: $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)}(\mathbf{A}^{(l-1)})^T$