Backpropagation and Gradient-Based Optimization in Multi-Layer Perceptrons

Atharv Bhatt

1 Introduction

Multi-Layer Perceptrons (MLPs) are feed-forward neural networks capable of learning complex nonlinear mappings between inputs and outputs. The learning process of an MLP is governed by two fundamental mechanisms: **backpropagation**, which computes gradients efficiently, and **gradient-based optimization**, which updates parameters to minimize prediction error.

This report provides a complete and detailed explanation of all concepts involved, including their mathematical formulation, motivation, and purpose.

2 Neural Networks as Parametric Models

An MLP represents a parametric function:

$$\hat{y} = f(x; \theta)$$

where:

- x denotes the input vector,
- $\hat{y}$ denotes the predicted output,
- $\theta = \{W, b\}$ represents all trainable parameters.

Learning consists of finding parameters θ that minimize a loss function measuring the discrepancy between $\hat{y}$ and the true target y.

3 Neuron Model and Linear Transformation

Each neuron computes:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

where:

- x_i is the i^{th} input feature,
- w_i is the corresponding weight,
- b is the bias term,
- z is the pre-activation value.

The bias term allows neurons to shift activation thresholds and prevents the network from being constrained to pass through the origin.

4 Activation Functions

Stacking linear transformations without nonlinear activation collapses the network into a single linear mapping. Activation functions introduce nonlinearity, enabling the network to approximate complex functions.

4.1 Sigmoid

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

The sigmoid maps values into $(0, 1)$ and is suitable for probability estimation. However, its derivative becomes very small for large $|z|$, causing vanishing gradients.

4.2 ReLU

$$f(z) = \max(0, z)$$

ReLU allows gradients to pass unchanged for positive activations, improving gradient flow, but may cause inactive neurons if inputs remain negative.

5 Forward Propagation

For layer l:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = f(z^{(l)})$$

Forward propagation computes the output by successively transforming inputs through network layers.

6 Loss Functions

Loss functions quantify prediction error as a scalar objective.

6.1 Mean Squared Error

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Used for regression, it penalizes large errors more strongly.

6.2 Binary Cross-Entropy

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

Derived from maximum likelihood estimation for Bernoulli distributions.

6.3 Categorical Cross-Entropy

$$\mathcal{L}_{CCE} = -\sum_{k} y_k \log(\hat{y}_k)$$

Measures divergence between true and predicted class distributions.

7 Why Backpropagation Is Necessary

To optimize the network, gradients of the loss with respect to every parameter must be computed:

$$\frac{\partial \mathcal{L}}{\partial \theta}$$

Since the loss depends on parameters through nested compositions of functions, backpropagation applies the chain rule efficiently.

8 Chain Rule Foundation

For composite mappings:

$$\mathcal{L} = f(g(h(x)))$$

$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}$$

Backpropagation systematically applies this principle layer by layer.

9 Error Signal Definition

The error signal at layer l is defined as:

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial z^{(l)}}$$

This quantity measures how sensitive the loss is to changes in the pre-activation values.

10 Backpropagation Equations

10.1 Output Layer

For softmax with cross-entropy loss:

$$\delta^{(L)} = \hat{y} - y$$

This result arises from algebraic cancellation and provides numerical stability.

10.2 Hidden Layers

$$\delta^{(l)} = (W^{(l+1)T}\delta^{(l+1)}) \odot f'(z^{(l)})$$

This equation propagates error backward while scaling it by local activation sensitivity.

11 Gradient Computation

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^T$$

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

These gradients determine how parameters should change to reduce loss.

12 Gradient Descent

Parameters are updated using:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}$$

The learning rate η controls update magnitude and stability.

13 Limitations of Vanilla Gradient Descent

Standard gradient descent:

- Converges slowly in narrow valleys,
- Oscillates in ill-conditioned loss surfaces,
- Uses a single global learning rate.

These limitations motivate advanced optimizers.

14 RMSProp Optimizer

RMSProp adapts the learning rate for each parameter individually.

14.1 Mathematical Formulation

Let $g_t = \nabla_\theta \mathcal{L}_t$ be the gradient at time step t.

$$s_t = \beta s_{t-1} + (1 - \beta) g_t^2$$

where:

- s_t is the exponentially weighted moving average of squared gradients,
- $\beta \in (0, 1)$ controls memory decay,
- g_t^2 is element-wise squaring.

Parameter update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_t + \epsilon}} g_t$$

14.2 Purpose of RMSProp

- Prevents exploding updates,
- Reduces learning rate for frequently changing parameters,
- Stabilizes training on non-stationary objectives.

15 Adam Optimizer

Adam (Adaptive Moment Estimation) combines momentum and RMSProp.

15.1 First Moment (Mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

15.2 Second Moment (Variance)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

15.3 Bias Correction

Since m_t and v_t are biased toward zero initially:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

15.4 Parameter Update

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

15.5 Why Adam Works

- Momentum accelerates convergence,
- Adaptive scaling stabilizes updates,
- Bias correction ensures reliable early training.

16 Vanishing and Exploding Gradients

Vanishing gradients arise when repeated multiplication of derivatives less than one causes gradients to shrink exponentially. Exploding gradients occur when derivatives greater than one amplify gradients.

17 Conclusion

Backpropagation provides an efficient mechanism for computing gradients in deep networks, while gradient-based optimizers such as RMSProp and Adam ensure stable and efficient parameter updates. Every component in this framework exists to preserve gradient information, stabilize learning, and minimize prediction error effectively.