# Deep Learning Course – L 7 - Detailed Notes

# 📉 Gradient Descent and Learning Rate in Neural Networks

Gradient Descent is one of the most essential optimization algorithms used in machine learning and deep learning. In this lecture, we'll explore **how gradient descent works** and the **critical role of the learning rate** in optimizing a model's parameters.

---

## 🔧 What Is Gradient Descent?

# Forward Propagation and Backward Propagation in Neural Networks

---

## 🔁 1. Recap of Previous Concepts

Before diving into forward and backward propagation, let's briefly revisit what we've covered:

### ✅ Logistic Regression

- Logistic regression is used for binary classification.
- It forms the foundation of a simple neural network (a single-layer perceptron).
- The output is passed through a **sigmoid activation function** to predict probabilities.

### ✅ Weights and Bias

- **Weights (W):** Control the strength of the connection between neurons.
- **Bias (b):** Allows the model to shift the activation function to fit the data better.
- Both are learnable parameters.

### ✅ Loss Function

- Measures the difference between actual output (label $y$) and predicted output ($\hat{y}$).
- Common loss for classification: **Binary Cross-Entropy Loss**.

- Helps assess how well the model is performing.

### ✅ Gradient Descent

- Optimization algorithm used to minimize the loss.
- Updates weights and biases in the opposite direction of the gradient (steepest descent).
- Formula:

  W=W−α⋅∂Loss∂WW = W - \alpha \cdot \frac{\partial \text{Loss}}{\partial W}W=W−α⋅∂W∂Loss

  where α\alphaα is the learning rate.

---

## 🔄 2. Forward and Backward Propagation

These are the **core components of training** in neural networks.

---

### 🔷 Forward Propagation

*What is it?*

- The process of **passing input data through the network** to get a prediction.

*Steps:*

1. **Input layer:** Takes raw input xxx.
2. **Hidden layers:** Compute weighted sum → apply activation functions.
3. **Output layer:** Produces the prediction y^\hat{y}y^.
4. **Loss is calculated** by comparing y^\hat{y}y^ with actual output yyy.

*Example:*

For one hidden layer:

z[1]=W[1]x+b[1]a[1]=ReLU(z[1])z[2]=W[2]a[1]+b[2]y^=σ(z[2])z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \text{ReLU}(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ \hat{y} = \sigma(z^{[2]})z[1]=W[1]x+b[1]a[1]=ReLU(z[1])z[2]=W[2]a[1]+b[2]y^=σ(z[2])

*Output:*

- We get the predicted value y^\hat{y}y^.

- Next, we compute the loss.

---

### Backward Propagation

*What is it?*

- The process of **updating the weights and biases** to minimize the loss using derivatives.

*Goal:*

- Understand how changing each parameter (weight/bias) affects the loss.

*Uses:*

- **Chain Rule** to compute gradients from output layer back to input.

---

## 3. Toy Example: Step-by-Step

Let's understand forward and backward propagation using a very basic mathematical function:

### Function:

j=3×((a+b×c))j = 3 \times ((a + b \times c))j=3×((a+b×c))

### Step 1: Define Operations

- Let u=b×cu = b \times cu=b×c
- Let v=a+uv = a + uv=a+u
- Then j=3×vj = 3 \times vj=3×v

---

### Forward Propagation

Let's assign values:

- a = 1, b = 2, c = 3

Now compute:

1. u=2×3=6u = 2 \times 3 = 6u=2×3=6
2. v=1+6=7v = 1 + 6 = 7v=1+6=7
3. j=3×7=21j = 3 \times 7 = 21j=3×7=21

So, the output is **21**.

---

### Backward Propagation

Now we want to compute **how sensitive the output `j` is to each variable (a, b, c)**.

We use **derivatives**:

*Step 1: Derivative of j w.r.t v:*
∂j∂v=3\frac{\partial j}{\partial v} = 3∂v∂j=3

*Step 2: Derivative of v w.r.t a:*
∂v∂a=1⇒∂j∂a=∂j∂v·∂v∂a=3·1=3\frac{\partial v}{\partial a} = 1 \Rightarrow \frac{\partial j}{\partial a} = \frac{\partial j}{\partial v} \cdot \frac{\partial v}{\partial a} = 3 \cdot 1 = 3∂a∂v =1⇒∂a∂j=∂v∂j·∂a∂v=3·1=3

*Step 3: Derivative of v w.r.t u:*
∂v∂u=1\frac{\partial v}{\partial u} = 1∂u∂v=1

*Step 4: Derivative of u w.r.t b and c:*
∂u∂b=c=3,∂u∂c=b=2\frac{\partial u}{\partial b} = c = 3, \quad \frac{\partial u}{\partial c} = b = 2∂b∂u=c=3,∂c∂u=b=2

*Now use chain rule:*
∂j∂b=∂j∂v·∂v∂u·∂u∂b=3·1·3=9\frac{\partial j}{\partial b} = \frac{\partial j}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial b} = 3 \cdot 1 \cdot 3 = 9∂b∂j=∂v∂j ·∂u∂v·∂b∂u=3·1·3=9 ∂j∂c=3·1·2=6\frac{\partial j}{\partial c} = 3 \cdot 1 \cdot 2 = 6∂c∂j =3·1·2=6

---

## 4. Chain Rule in Backpropagation

The **chain rule** is the backbone of backward propagation. It allows us to calculate:

∂j∂x=∂j∂y·∂y∂x\frac{\partial j}{\partial x} = \frac{\partial j}{\partial y} \cdot \frac{\partial y}{\partial x}∂x∂j=∂y∂j·∂x∂y

In complex neural networks, we stack multiple functions together, so we apply the chain rule repeatedly through layers to calculate gradients.

---

## 🎯 5. Why Is Backpropagation Important?

- It gives us the **direction** to move weights to reduce the loss.
- Used along with gradient descent to **train** the neural network.
- Efficient for **deep networks** with many layers.

---

## 🧠 6. Summary of Key Points

| Concept | Description |
|---|---|
| Forward Propagation | Pass input through network to get output |
| Loss Function | Measures error between prediction and actual output |
| Backward Propagation | Calculates gradients of weights & biases |
| Chain Rule | Helps in computing gradients across multiple layers |
| Optimization | Gradients used to update parameters and reduce error |

---

## 📚 What’s Next?

In the next session, we’ll study **activation functions** (like Sigmoid, ReLU, Tanh), why they are essential, and how they add non-linearity to neural networks.

.