**Overfitting & Underfitting in Deep Learning:**

**📌 Introduction**

Overfitting and underfitting are two common issues in machine learning. Understanding how to manage them is crucial for building models that generalize well.

**1️⃣ Project Steps: How to Do This Project in Google Colab**

**Step 1: Set Up the Environment**

1. Open **Google Colab**
2. Install required packages:
3. !pip install tensorflow tensorflow-hub tensorflow-datasets
4. Import necessary libraries:
5. import tensorflow as tf
6. from tensorflow.keras import layers, regularizers
7. import numpy as np
8. import matplotlib.pyplot as plt

**Step 2: Load the Dataset**

The dataset used is the **HIGGS dataset** with **28 features** and **binary labels**.

* **Download and load the dataset:**
* gz = tf.keras.utils.get\_file('HIGGS.csv.gz', 'http://mlphysics.ics.uci.edu/data/higgs/HIGGS.csv.gz')
* ds = tf.data.experimental.CsvDataset(gz, [float(),] \* (28 + 1), compression\_type="GZIP")

**Step 3: Data Preprocessing**

* Convert the dataset into **(features, label)** format:
* def pack\_row(\*row):
* label = row[0]
* features = tf.stack(row[1:], 1)
* return features, label
* packed\_ds = ds.batch(10000).map(pack\_row).unbatch()
* **Split data** into **training** and **validation** sets:
* N\_VALIDATION = 1000
* N\_TRAIN = 10000
* train\_ds = packed\_ds.skip(N\_VALIDATION).take(N\_TRAIN).cache()
* validate\_ds = packed\_ds.take(N\_VALIDATION).cache()

**Step 4: Create and Train Models**

**1️⃣ Tiny Model (Underfitting Example)**

tiny\_model = tf.keras.Sequential([

layers.Dense(16, activation='elu', input\_shape=(28,)),

layers.Dense(1)

])

**2️⃣ Large Model (Overfitting Example)**

large\_model = tf.keras.Sequential([

layers.Dense(512, activation='elu', input\_shape=(28,)),

layers.Dense(512, activation='elu'),

layers.Dense(512, activation='elu'),

layers.Dense(1)

])

**3️⃣ Regularized Model (Balanced Model)**

l2\_model = tf.keras.Sequential([

layers.Dense(512, activation='elu', kernel\_regularizer=regularizers.l2(0.001), input\_shape=(28,)),

layers.Dense(512, activation='elu', kernel\_regularizer=regularizers.l2(0.001)),

layers.Dense(1)

])

**Step 5: Train and Evaluate Models**

models = {"Tiny": tiny\_model, "Large": large\_model, "L2": l2\_model}

for name, model in models.items():

model.compile(optimizer=tf.keras.optimizers.Adam(0.001), loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(train\_ds.batch(500), epochs=100, validation\_data=validate\_ds.batch(500), verbose=1)

**Step 6: Save and Reload Model**

l2\_model.save("overfitting\_underfitting\_model.keras")

reloaded\_model = tf.keras.models.load\_model("overfitting\_underfitting\_model.keras")

**2️⃣ Must-Know Interview Questions & Answers**

**1️⃣ What is Overfitting?**

📌 Overfitting occurs when a model learns too much from training data, capturing **noise** instead of **general patterns**, leading to **poor performance** on unseen data.

**2️⃣ How Can You Detect Overfitting?**

✅ High accuracy on training data but **low accuracy** on validation/test data.

**3️⃣ What Causes Underfitting?**

🔹 A model that is **too simple**, not capturing key patterns in data, leading to **both training and validation errors**.

**4️⃣ How to Prevent Overfitting?**

✔️ Use **Regularization (L2, L1)**, **Dropout Layers**, **Early Stopping**, and **Data Augmentation**.

**5️⃣ What is Dropout in Neural Networks?**

💡 Dropout **randomly deactivates neurons** during training to prevent dependency on specific features.

**6️⃣ What is Early Stopping?**

🛑 Stops training when **validation loss increases**, preventing overfitting.

**7️⃣ What is the Purpose of L2 Regularization?**

🔎 L2 Regularization prevents **large weights** by adding a **penalty term** to the loss function.