**DEEP LEARNING**

**Pertemuan 1.**

**1. Program Dasar NumPy (Array dan Operasi Vektor)**

**NumPy** adalah pustaka fundamental untuk komputasi numerik di Python. Ini sangat penting dalam Deep Learning untuk menangani data dan bobot sebagai **array multidimensi (tensor)**.

Python

import numpy as np

# Program 1: Dasar NumPy (Array/Tensor)

# 1. Membuat Tensor/Array

data\_vektor = np.array([1.0, 2.0, 3.0])

data\_matriks = np.array([[1, 2], [3, 4]])

print(f"Vektor (Dimensi 1): {data\_vektor}")

print(f"Bentuk (Shape) Vektor: {data\_vektor.shape}")

print(f"\nMatriks (Dimensi 2):\n{data\_matriks}")

print(f"Bentuk (Shape) Matriks: {data\_matriks.shape}")

# 2. Operasi Dasar Vektor (Simulasi Operasi Node Saraf)

bobot = np.array([0.5, 0.2, -0.1])

bias = 0.8

# Penjumlahan Vektor (Input + Bobot)

input\_berbobot = data\_vektor \* bobot

print(f"\nInput Berbobot (Element-wise perkalian): {input\_berbobot}")

# Total input sebelum aktivasi (Sum + Bias)

total\_input = np.sum(input\_berbobot) + bias

print(f"Total Input (Z): {total\_input:.4f}")

**2. Implementasi Fungsi Aktivasi Sederhana (Sigmoid)**

Fungsi **Aktivasi** adalah komponen kunci dalam jaringan saraf yang memperkenalkan non-linearitas. Sigmoid adalah salah satu fungsi aktivasi klasik.

Python

import numpy as np

# Program 2: Fungsi Aktivasi Sigmoid

def sigmoid(z):

"""Menghitung fungsi aktivasi Sigmoid."""

return 1 / (1 + np.exp(-z))

# Input (Z) dari program sebelumnya

z\_value\_1 = 1.25 # Nilai Z positif

z\_value\_2 = -3.0 # Nilai Z negatif

z\_value\_3 = 0.0 # Nilai Z nol

# Menggunakan fungsi Sigmoid

output\_1 = sigmoid(z\_value\_1)

output\_2 = sigmoid(z\_value\_2)

output\_3 = sigmoid(z\_value\_3)

# Mencetak Hasil (Output akan selalu antara 0 dan 1)

print(f"Input Z = {z\_value\_1:.2f}, Output Sigmoid (A) = {output\_1:.4f}")

print(f"Input Z = {z\_value\_2:.2f}, Output Sigmoid (A) = {output\_2:.4f}")

print(f"Input Z = {z\_value\_3:.2f}, Output Sigmoid (A) = {output\_3:.4f}")

**3. Implementasi Fungsi Kerugian (Loss Function) - MSE**

**Fungsi Kerugian (Loss Function)** mengukur seberapa baik model memprediksi hasil. **Mean Squared Error (MSE)** umum digunakan untuk tugas regresi.

Python

import numpy as np

# Program 3: Fungsi Kerugian (Loss Function) - MSE

def mean\_squared\_error(y\_true, y\_pred):

"""Menghitung Mean Squared Error (MSE)."""

# Hitung selisih kuadrat

squared\_error = (y\_true - y\_pred) \*\* 2

# Rata-ratakan selisih kuadrat

mse = np.mean(squared\_error)

return mse

# Contoh 1: Prediksi yang cukup baik

y\_sebenarnya\_1 = np.array([10, 20, 30])

y\_prediksi\_1 = np.array([10.5, 19.8, 30.1])

# Contoh 2: Prediksi yang buruk

y\_sebenarnya\_2 = np.array([10, 20, 30])

y\_prediksi\_2 = np.array([5, 15, 25])

loss\_1 = mean\_squared\_error(y\_sebenarnya\_1, y\_prediksi\_1)

loss\_2 = mean\_squared\_error(y\_sebenarnya\_2, y\_prediksi\_2)

print(f"Loss (MSE) untuk Prediksi Baik: {loss\_1:.4f}")

print(f"Loss (MSE) untuk Prediksi Buruk: {loss\_2:.4f}")

print("\n\*Semakin kecil nilai Loss, semakin baik prediksi model.")

**4. Perhitungan Turunan Dasar (Untuk Backpropagation)**

**Turunan (Derivative)** adalah konsep matematika penting di balik *Backpropagation* dan optimasi. Program ini menunjukkan bagaimana turunan digunakan untuk mengukur laju perubahan.

Python

# Program 4: Perhitungan Turunan Dasar (Simulasi Gradient)

def fungsi\_biaya\_sederhana(w):

"""

Simulasi fungsi biaya sederhana (misalnya, w^2).

Kita ingin mencari nilai w yang meminimalkan biaya.

"""

return w \*\* 2

def turunan\_fungsi\_biaya(w):

"""

Turunan analitik dari fungsi biaya (2w).

Ini adalah Gradient.

"""

return 2 \* w

# Nilai bobot (Weight) saat ini

bobot\_saat\_ini = 3.0

# 1. Hitung biaya saat ini

biaya\_saat\_ini = fungsi\_biaya\_sederhana(bobot\_saat\_ini)

# 2. Hitung Gradient (Turunan)

gradient = turunan\_fungsi\_biaya(bobot\_saat\_ini)

print(f"Bobot saat ini (w): {bobot\_saat\_ini}")

print(f"Biaya saat ini: {biaya\_saat\_ini}")

print(f"Gradient (Turunan): {gradient}")

# Jika Gradient positif (+6.0), artinya bobot harus DIKURANGI untuk meminimalkan biaya.

# Jika Gradient negatif, bobot harus DITAMBAH.

**5. Simulasi Langkah Optimasi Sederhana (Gradient Descent)**

Menggabungkan konsep turunan (Gradient) dengan **Learning Rate** untuk mensimulasikan satu langkah optimasi **Gradient Descent**, yaitu cara model belajar.

Python

# Program 5: Simulasi Satu Langkah Gradient Descent

# Menggunakan fungsi dari Program 4

# def turunan\_fungsi\_biaya(w): return 2 \* w

def satu\_langkah\_gradient\_descent(w\_lama, gradient, learning\_rate):

"""

Menghitung bobot baru setelah satu langkah optimasi.

Bobot\_baru = Bobot\_lama - (Learning\_Rate \* Gradient)

"""

w\_baru = w\_lama - (learning\_rate \* gradient)

return w\_baru

# Parameter

w\_awal = 3.0

L\_R = 0.1 # Learning Rate

# Langkah 1: Hitung Gradient pada w\_awal

gradient\_awal = 2 \* w\_awal # Menggunakan 2w

print(f"Bobot Awal (w): {w\_awal}")

print(f"Gradient Awal: {gradient\_awal}")

print(f"Learning Rate (L.R): {L\_R}")

# Langkah 2: Hitung Bobot Baru

w\_baru\_1 = satu\_langkah\_gradient\_descent(w\_awal, gradient\_awal, L\_R)

print(f"\nBobot Baru (Setelah 1 iterasi): {w\_baru\_1:.4f}")

# Langkah 3 (Optional): Cek lagi Gradient-nya

gradient\_baru = 2 \* w\_baru\_1

print(f"Gradient Baru: {gradient\_baru:.4f}")

print("\n\*Perhatikan bahwa bobot (w) bergerak mendekati 0, dan Gradient juga mengecil.")