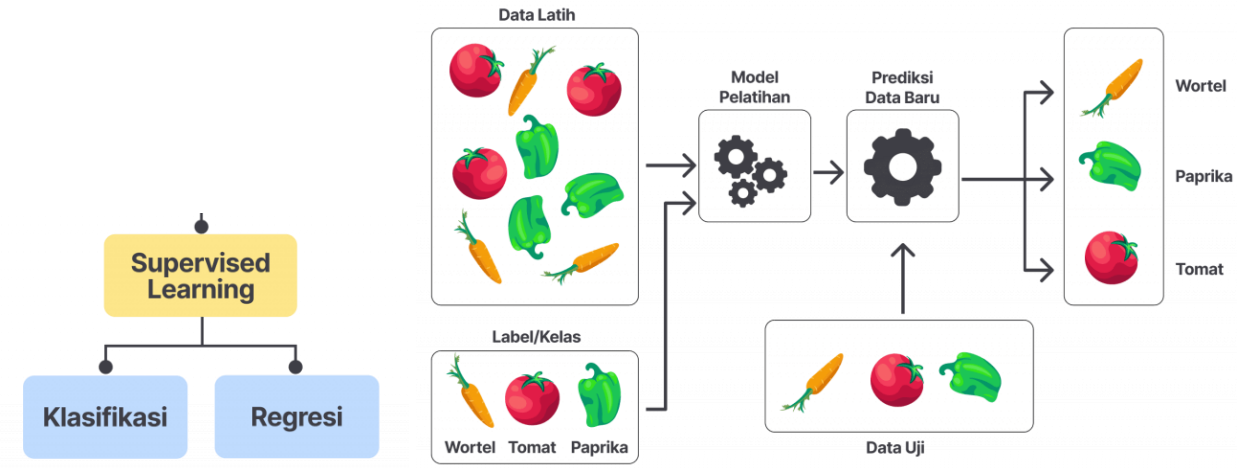
NEURAL NETWORK



normalisasi–denormalisasi

Analogi Sehari-hari

Bayangkan kamu punya **penggaris**:

- Panjang asli tanah = 50 m² sampai 150 m² → seperti garis dari 50 cm sampai 150 cm.
- Normalisasi = kita "menyusutkan" penggaris itu supaya panjangnya selalu **0 sampai 1**.
- Jadi angka 50 m² ditempatkan di titik 0, dan 150 m² ditempatkan di titik 1.
  Semua nilai lain otomatis jatuh di antara 0–1 sesuai posisinya.

👉 Normalisasi = **mengubah skala tanpa mengubah urutan**.

👉 Denormalisasi = **mengembalikan ke ukuran asli**.

Rumus Inti

- **Normalisasi (Min–Max):** $x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$
- **Denormalisasi (kembali ke data asli):** $\left[x = \left(x_{norm} \cdot (x_{max} - x_{min})\right) + x_{min}\right]$

Contoh Praktis

Normalisasi

```
import numpy as np

x = np.array([50, 100, 200, 300])
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
print(x_norm)

# Output: [0.   0.2  0.6  1. ]
```

- 50 → 0
- 100 → 0.2
- 200 → 0.6
- 300 → 1

Denormalisasi

```
y = np.array([300, 800])
y_norm = 0.5
y_denorm = (y_norm * (np.max(y) - np.min(y))) + np.min(y)
print(y_denorm)
# Output: 550
```

- Nilai normalisasi 0.5 → kembali jadi 550.

Kenapa Penting untuk Neural Network?

- **Menghindari dominasi angka besar**: misalnya harga rumah (jutaan) vs luas tanah (ratusan).
- **Mempercepat dan menstabilkan perhitungan gradien** saat training.
- **Memudahkan visualisasi**: grafik jadi lebih rapi karena semua data ada di rentang 0–1.

Kalimat Sederhana

- **Normalisasi** = "mengecilkan angka supaya pas di antara 0–1, tapi tetap menjaga perbandingan."
- **Denormalisasi** = "mengembalikan angka kecil itu ke ukuran aslinya."

Contoh 1: Tinggi Badan

Data tinggi (cm): `[150, 160, 180, 200]`

Normalisasi:

$$\left[x_{norm} = \frac{x - 150}{200 - 150} = \frac{x - 150}{50}\right]$$

```
import numpy as np
x = np.array([150, 160, 180, 200])
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
print(x_norm)
# [0.   0.2  0.6  1. ]
```

Denormalisasi (misal nilai 0.4):

$$\left[x = \left(0.4 \cdot (200 - 150)\right) + 150 = (0.4 \cdot 50) + 150 = 20 + 150 = 170\right]$$

Contoh 2: Nilai Ujian

Data nilai: `[40, 60, 80, 100]`

Normalisasi: $\left[x_{norm} = \frac{x-40}{100-40} = \frac{x-40}{60}\right]$

```
x = np.array([40, 60, 80, 100])
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
print(x_norm)
# [0.   0.33 0.67 1. ]
```

Denormalisasi (misal nilai 0.75):

$$\left[x = \left(0.75 \cdot (100 - 40)\right) + 40 = (0.75 \cdot 60) + 40 = 45 + 40 = 85\right]$$

Contoh 3: Suhu Ruangan

Data suhu (°C): `[20, 25, 30, 35]`

Normalisasi: $\left[x_{norm} = \frac{x-20}{35-20} = \frac{x-20}{15}\right]$

```
x = np.array([20, 25, 30, 35])
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
print(x_norm)
# [0.   0.33 0.67 1. ]
```

Denormalisasi (misal nilai 0.2): $\left[x = \left(0.2 \cdot (35 - 20)\right) + 20 = (0.2 \cdot 15) + 20 = 3 + 20 = 23\right]$

📊 Contoh 4: Harga Barang

Data harga (Rp ribu): `[10, 50, 100, 200]`

Normalisasi: $\left[x_{norm} = \frac{x-10}{200-10} = \frac{x-10}{190}\right]$

```
x = np.array([10, 50, 100, 200])
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
print(x_norm)
# [0.    0.21 0.47 1.  ]
```

Denormalisasi (misal nilai 0.5): $\left[x = \left(0.5 \cdot (200 - 10)\right) + 10 = (0.5 \cdot 190) + 10 = 95 + 10 = 105\right]$

✨ Jadi, pola selalu sama:

- **Normalisasi** → "menyusutkan" ke 0–1.
- **Denormalisasi** → "mengembalikan" ke skala asli.

COPY PASTE

melihat proses **bolak-balik normalisasi ↔ denormalisasi**:

```
import numpy as np
import matplotlib.pyplot as plt

# Data asli
x = np.array([50, 100, 200, 300])

# Normalisasi Min-Max
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))

# Contoh nilai normalisasi yang ingin dikembalikan
val_norm = 0.5
val_denorm = (val_norm * (np.max(x) - np.min(x))) + np.min(x)

# Buat figure dengan 3 subplot
fig, axs = plt.subplots(1, 3, figsize=(15, 4))

# Grafik batang data asli
axs[0].bar(range(len(x)), x, color='skyblue')
axs[0].set_title("Data Asli")
axs[0].set_xlabel("Index")
axs[0].set_ylabel("Nilai")
```

```
axs[0].set_xticks(range(len(x)))
axs[0].set_xticklabels([f"x{i}" for i in range(len(x))])

# Grafik batang data normalisasi
axs[1].bar(range(len(x_norm)), x_norm, color='orange')
axs[1].axhline(val_norm, color='red', linestyle='--',
label=f"val_norm={val_norm}")
axs[1].set_title("Data Setelah Normalisasi")
axs[1].set_xlabel("Index")
axs[1].set_ylabel("Nilai Normalisasi (0-1)")
axs[1].legend()
axs[1].set_xticks(range(len(x_norm)))
axs[1].set_xticklabels([f"x{i}" for i in range(len(x_norm))])

# Grafik batang data asli dengan titik hasil denormalisasi
axs[2].bar(range(len(x)), x, color='lightgreen')
axs[2].axhline(val_denorm, color='red', linestyle='--',
label=f"val_denorm={val_denorm}")
axs[2].set_title("Denormalisasi (kembali ke skala asli)")
axs[2].set_xlabel("Index")
axs[2].set_ylabel("Nilai")
axs[2].legend()
axs[2].set_xticks(range(len(x)))
axs[2].set_xticklabels([f"x{i}" for i in range(len(x))])

plt.tight_layout()
plt.show()
```

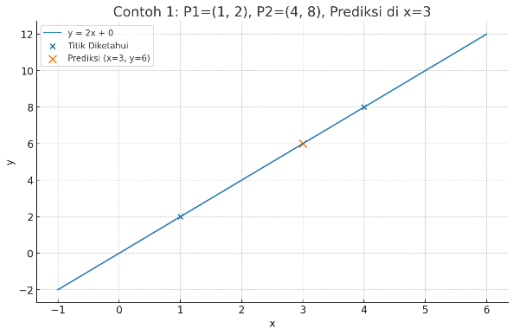
🔎 Penjelasan

- **Subplot 1**: data asli `[50, 100, 200, 300]`.
- **Subplot 2**: data setelah normalisasi (0–1), dengan garis merah di **0.5**.
- **Subplot 3**: data asli lagi, dengan garis merah di **nilai denormalisasi = 175** (hasil dari 0.5).

👉 Dengan ini, bisa melihat jelas bahwa angka **0.5 di dunia normalisasi** kembali menjadi **175 di dunia asli**.

---

1. PERSAMAAN GARIS LINEAR



$$y = mx + b$$

1. Definisi gradien (slope)

Sebuah garis lurus bisa ditentukan hanya dengan **dua titik**, misalnya (x_1, y_1) dan (x_2, y_2).

Gradien (kemiringan garis) didefinisikan sebagai:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Artinya: seberapa besar perubahan y setiap kali x bertambah satu satuan.

2. Persamaan garis dengan slope m

Jika kita sudah tahu slope m, maka persamaan garis yang melewati (x_1, y_1) bisa ditulis dalam bentuk **point-slope form**:

$$y - y_1 = m(x - x_1)$$

Ini artinya: untuk titik mana pun (x, y) di garis, selisih y terhadap y_1 sama dengan slope kali selisih x terhadap x_1.

# Hitung slope (m) dan intercept (b)
m = (y2 - y1) / (x2 - x1)
b = y1 - m * x1

# Persamaan garis: y = m*x + b
y_pred = m * x_pred + b

CONTOH PREDIKSI HARGA RUMAH
DATA

	LUAS TANAH(M^2)	HARGA (JT)
1	50	150
2	150	800

TUGAS PREDIKSI

	LUAS TANAH(M^2)	HARGA (JT)

1	100	??

Penjelasan langkah:

1. **Data input:** dua titik `(x1, y1)` dan `(x2, y2)` sebagai data harga rumah.

```
x = np.array([50, 150])   # luas tanah
print(x)#[ 50 150]
```

2. **Normalisasi:** agar nilai lebih terkontrol antara 0–1.

```
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
print (x_norm) #[0. 1.]
```

3. **Rumus regresi linier dua titik:**

$$m = \frac{y_2 - y_1}{x_2 - x_1}, b = y_1 - m \cdot x_1$$

```
x1, x2 = x_norm
y1, y2 = y_norm

m = (y2 - y1) / (x2 - x1)
b = y1 - m * x1
```

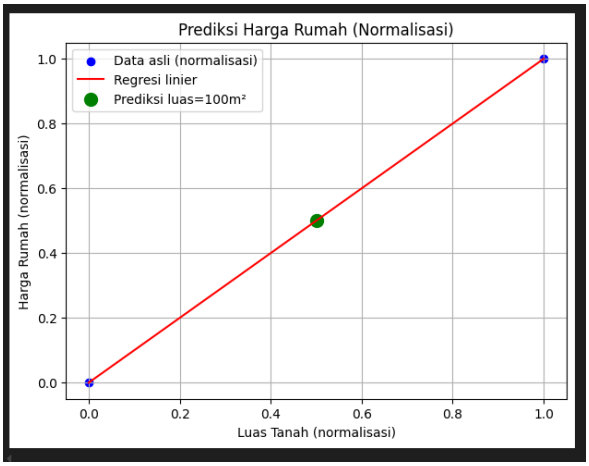
4. **Prediksi:** substitusi nilai `x_pred` ke persamaan.

```
# Misal mau prediksi rumah dengan luas 100 m2
x_pred = 100
x_pred_norm = (x_pred - np.min(x)) / (np.max(x) - np.min(x))  # normalisasi input
y_pred_norm = m * x_pred_norm + b                             # prediksi di ruang normalisasi
```

5. **Denormalisasi:** hasil dikembalikan ke satuan asli (juta rupiah).

```
# Denormalisasi hasil prediksi ke skala harga asli
y_pred = y_pred_norm * (np.max(y) - np.min(y)) + np.min(y)
print(f"Prediksi harga rumah (luas {x_pred} m2) = {y_pred:.2f} juta rupiah")
```

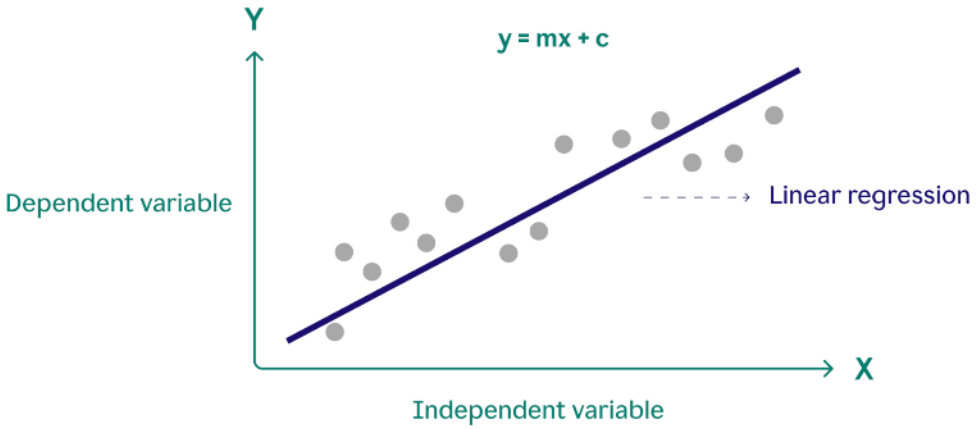
6. **Plot grafik:** menampilkan titik data, garis regresi, dan hasil prediksi.




```
import numpy as np
import matplotlib.pyplot as plt

# ---------- Data asli ----------
# Misal: luas tanah (m2) dan harga rumah (juta rupiah)
x = np.array([50, 150])   # luas tanah
y = np.array([300, 800])  # harga rumah

# ---------- Normalisasi ----------
x_norm = (x - np.min(x)) / (np.max(x) - np.min(x))
y_norm = (y - np.min(y)) / (np.max(y) - np.min(y))

print("x_norm:", x_norm)
print("y_norm:", y_norm)

# ---------- Hitung slope dan intercept pada data ter-normalisasi ----------
x1, x2 = x_norm
y1, y2 = y_norm

m = (y2 - y1) / (x2 - x1)
b = y1 - m * x1

print(f"Persamaan garis normalisasi: y = {m:.3f}x + {b:.3f}")

# ---------- Prediksi ----------
# Misal mau prediksi rumah dengan luas 100 m2
x_pred = 100
x_pred_norm = (x_pred - np.min(x)) / (np.max(x) - np.min(x))  # normalisasi input
y_pred_norm = m * x_pred_norm + b                             # prediksi di ruang
normalisasi
```

```
# Denormalisasi hasil prediksi ke skala harga asli
y_pred = y_pred_norm * (np.max(y) - np.min(y)) + np.min(y)
print(f"Prediksi harga rumah (luas {x_pred} m2) = {y_pred:.2f} juta rupiah")

# ---------- Visualisasi ----------
x_line = np.linspace(np.min(x_norm), np.max(x_norm), 100)
y_line = m * x_line + b

plt.figure(figsize=(7, 5))
plt.scatter(x_norm, y_norm, color='blue', label='Data asli (normalisasi)')
plt.plot(x_line, y_line, color='red', label='Regresi linier')
plt.scatter(x_pred_norm, y_pred_norm, color='green', s=100, label=f'Prediksi
luas={x_pred}m²')
plt.title("Prediksi Harga Rumah (Normalisasi)")
plt.xlabel("Luas Tanah (normalisasi)")
plt.ylabel("Harga Rumah (normalisasi)")
plt.legend()
plt.grid(True)
plt.show()
==========================================
x_norm: [0. 1.]
y_norm: [0. 1.]
Persamaan garis normalisasi: y = 1.000x + 0.000
Prediksi harga rumah (luas 100 m2) = 550.00 juta rupiah
```

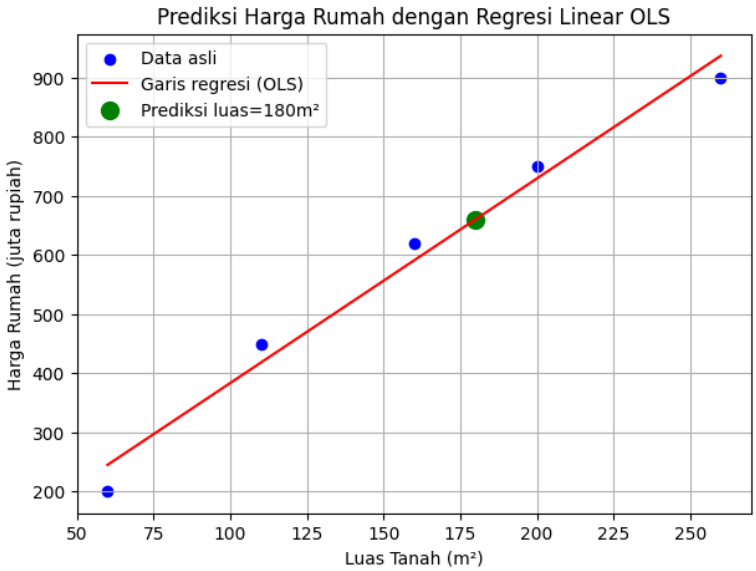
2. PERSAMAAN GARIS LINEAR OLS



$$m = \frac{\sum_{i=1}^{n} x_i y_i - n\,\bar{x}\,\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\,\bar{x}^2}.$$

$$\boxed{b = \bar{y} - m\bar{x}}$$

```
# jumlah data
n = len(x)
# rata-rata
x_mean = np.mean(x)
y_mean = np.mean(y)
# komponen pembilang dan penyebut
num = np.sum(x * y) - n * x_mean * y_mean
# Σ x_i y_i - n x̄ ȳ
den = np.sum(x**2) - n * x_mean**2
# Σ x_i^2 - n x̄^2
# slope
m = num / den
print("Slope m =", m)
# b
b = y_mean - m * x_mean
print("Intercept b =", b)
```



CONTOH PREDIKSI HARGA RUMAH
DATA

	LUAS TANAH(M^2) (x)	HARGA (JT) (y)
1	60	200
2	110	450
3	160	620
4	200	750
5	260	900

TUGAS PREDIKSI

	LUAS TANAH(M^2)	HARGA (JT)
1	180	??

```
# 1. Import Library
import numpy as np
import matplotlib.pyplot as plt

# 2. Data Raw
# ---------- Data harga rumah ----------
# Data (x = luas tanah dalam m², y = harga rumah dalam juta rupiah)
x = np.array([60, 110, 160, 200, 260])
y = np.array([200, 450, 620, 750, 900])

print("=== DATA ASLI ===")
print(f"Luas tanah (m²): {x}")
print(f"Harga rumah (juta): {y}")

# 3. Data Normalisasi
# ---------- Normalisasi ----------
x_min, x_max = np.min(x), np.max(x)
y_min, y_max = np.min(y), np.max(y)

x_norm = (x - x_min) / (x_max - x_min)
y_norm = (y - y_min) / (y_max - y_min)

print("\n=== DATA NORMALISASI ===")
print(f"x_norm = {np.round(x_norm, 4)}")
print(f"y_norm = {np.round(y_norm, 4)}")

# 4. Metode OLS
# ---------- Metode OLS pada Data Normalisasi ----------
n = len(x_norm)
x_norm_mean = np.mean(x_norm)
y_norm_mean = np.mean(y_norm)
```

```
# komponen pembilang dan penyebut untuk data normalisasi
num = np.sum(x_norm * y_norm) - n * x_norm_mean * y_norm_mean     # Σ
x_i y_i - n x̄ ȳ
den = np.sum(x_norm**2) - n * x_norm_mean**2                      # Σ
x_i² - n x̄²

# slope dan intercept di ruang normalisasi
m_norm = num / den
b_norm = y_norm_mean - m_norm * x_norm_mean

print(f"\n=== HASIL REGRESI DI RUANG NORMALISASI ===")
print(f"Slope (m_norm)     = {m_norm:.6f}")
print(f"Intercept (b_norm) = {b_norm:.6f}")
print(f"Persamaan garis normalisasi: y_norm = {m_norm:.6f} * x_norm +
{b_norm:.6f}")

# 5. KONVERSI KE SKALA ASLI (TAMBAHAN BARU)
# ---------- Konversi parameter ke skala asli ----------
m_actual = m_norm * (y_max - y_min) / (x_max - x_min)
b_actual = b_norm * (y_max - y_min) + y_min - m_actual * x_min

print(f"\n=== KONVERSI KE SKALA ASLI ===")
print(f"Slope (m_actual)     = {m_actual:.6f}")
print(f"Intercept (b_actual) = {b_actual:.6f}")
print(f"Persamaan garis asli: y = {m_actual:.6f} * x + {b_actual:.6f}")

# 6. Prediksi Normalisasi - Denormalisasi
# ---------- Prediksi dengan Dua Cara ----------
x_pred = 180

print(f"\n=== PREDIKSI UNTUK LUAS {x_pred} m² ===")

# Cara 1: Prediksi di ruang normalisasi lalu denormalisasi
x_pred_norm = (x_pred - x_min) / (x_max - x_min)
y_pred_norm = m_norm * x_pred_norm + b_norm
y_pred_1 = y_pred_norm * (y_max - y_min) + y_min

print("Cara 1 (Normalisasi → Prediksi → Denormalisasi):")
print(f"  x_pred = {x_pred} → x_norm = {x_pred_norm:.4f}")
print(f"  y_norm_pred = {m_norm:.4f} × {x_pred_norm:.4f} + {b_norm:.4f}
= {y_pred_norm:.4f}")
print(f"  y_pred = {y_pred_norm:.4f} × {y_max-y_min} + {y_min} =
{y_pred_1:.2f} juta")

# Cara 2: Langsung menggunakan persamaan di skala asli
```

```
y_pred_2 = m_actual * x_pred + b_actual
print(f"\nCara 2 (Langsung di skala asli):")
print(f"  y_pred = {m_actual:.6f} × {x_pred} + {b_actual:.6f} =
{y_pred_2:.2f} juta")

# 7. Visualisasi
plt.figure(figsize=(12, 5))

# Plot 1: Data asli dengan regresi
plt.subplot(1, 2, 1)
x_line = np.linspace(x_min, x_max, 100)
y_line = m_actual * x_line + b_actual

plt.scatter(x, y, color='blue', s=80, label='Data asli', alpha=0.7)
plt.plot(x_line, y_line, color='red', linewidth=2, label=f'Regresi: y =
{m_actual:.3f}x + {b_actual:.3f}')
plt.scatter(x_pred, y_pred_2, color='green', s=120, marker='*',
label=f'Prediksi: {y_pred_2:.1f} juta', zorder=5)

plt.title("Regresi Linear - Skala Asli")
plt.xlabel("Luas Tanah (m²)")
plt.ylabel("Harga Rumah (juta Rp)")
plt.legend()
plt.grid(True, alpha=0.3)

# Plot 2: Data normalisasi dengan regresi
plt.subplot(1, 2, 2)
x_norm_line = np.linspace(0, 1, 100)
y_norm_line = m_norm * x_norm_line + b_norm

plt.scatter(x_norm, y_norm, color='blue', s=80, label='Data
normalisasi', alpha=0.7)
plt.plot(x_norm_line, y_norm_line, color='red', linewidth=2,
label=f'Regresi: y = {m_norm:.3f}x + {b_norm:.3f}')
plt.scatter(x_pred_norm, y_pred_norm, color='green', s=120, marker='*',
label=f'Prediksi norm: {y_pred_norm:.3f}', zorder=5)

plt.title("Regresi Linear - Skala Normalisasi")
plt.xlabel("Luas Tanah (Normalisasi)")
plt.ylabel("Harga Rumah (Normalisasi)")
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```
# 8. Evaluasi Model
# Prediksi untuk semua data
y_pred_all = m_actual * x + b_actual

# Hitung MSE dan R-squared
mse = np.mean((y - y_pred_all) ** 2)
ss_total = np.sum((y - np.mean(y)) ** 2)
ss_residual = np.sum((y - y_pred_all) ** 2)
r_squared = 1 - (ss_residual / ss_total)

print(f"\n=== EVALUASI MODEL ===")
print(f"MSE (Mean Squared Error): {mse:.2f}")
print(f"R-squared: {r_squared:.4f}")
print(f"Akurasi: {r_squared*100:.2f}%")

# Tampilkan perbandingan prediksi vs aktual
print(f"\nPerbandingan Prediksi vs Aktual:")
print("Luas\tAktual\tPrediksi\tError")
for i in range(len(x)):
    error = y_pred_all[i] - y[i]
    print(f"{x[i]}m²\t{y[i]}jt\t{y_pred_all[i]:.1f}jt\t{error:+.1f}jt")
```

DASAR TEORI OLS

1. Masalahnya apa?

Kita punya beberapa titik data (x_i, y_i).

Contoh: tinggi badan (x) vs berat badan (y) 5 orang. Titik-titik ini **tidak pasti segaris lurus**, tapi kita ingin membuat **garis lurus terbaik** $y = mx + b$ untuk **memprediksi**.

2. Bagaimana garis disebut "terbaik"?

Kalau kita pilih sembarang garis, biasanya titik-titik tidak pas menempel di garis.

Maka kita ukur **error (kesalahan)** tiap titik:

$$\text{error}_i = y_i - (mx_i + b)$$

Yaitu selisih antara **nilai asli** y_i dengan **nilai prediksi** dari garis.

Kalau error besar → garisnya jelek.

Kalau error kecil → garis lebih cocok.

3. Kenapa pakai kuadrat error?

Karena ada error positif (titik di atas garis) dan error negatif (titik di bawah garis).

Kalau dijumlah langsung, bisa saling hapus.

Solusi: **pakai kuadrat error** → selalu positif.

$$\text{Total Error} = \sum (y_i - (mx_i + b))^2$$

Itulah **Least Squares (kuadrat terkecil)**: kita cari m dan b supaya jumlah kuadrat error ini **sekecil mungkin**.

Kita mulai dari **model linear**:

$$\hat{y}_i = mx_i + b$$

dan definisikan **Sum of Squared Errors (SSE)**:

$$SSE(m,b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2.$$

1. Turunan parsial dan persamaan normal

Ambil turunan parsial terhadap m dan b, lalu set sama dengan nol.

Turunan terhadap m:

$$\frac{\partial SSE}{\partial m} = \sum_{i=1}^{n} -2x_i(y_i - mx_i - b) = 0.$$

Turunan terhadap b:

Sederhanakan (hilangkan faktor -2) → kita dapat **persamaan normal**:

(1) $$\sum_{i=1}^{n} x_i y_i - m\sum_{i=1}^{n} x_i^2 - b\sum_{i=1}^{n} x_i = 0.$$

(2) $$\sum_{i=1}^{n} y_i - m\sum_{i=1}^{n} x_i - nb = 0.$$

$$f(a) = (7 - 3a - 2)^2.$$

Langkah:

- Anggap $u = 7 - 3a - 2 = 5 - 3a$.
- Turunan: $\frac{df}{da} = 2u \cdot \frac{du}{da}$.
- $\frac{du}{da} = -3$.

Hasil:

$$\frac{df}{da} = 2(5 - 3a)(-3) = -6(5 - 3a).$$

Ini persis pola $-2X(Y - aX - b)$ dengan $X = 3,\ Y = 7,\ b = 2$.

2. Selesaikan untuk b (dari persamaan (2))

Dari (2):

$$nb = \sum_{i=1}^{n} y_i - m\sum_{i=1}^{n} x_i$$

bagi n:

$$b = \frac{1}{n}\sum_{i=1}^{n} y_i - m\frac{1}{n}\sum_{i=1}^{n} x_i = \bar{y} - m\bar{x}.$$

Jadi diperoleh:

$$\boxed{b = \bar{y} - m\bar{x}}$$

3. Masukkan b ke persamaan (1) untuk cari m

Masukkan $b = \bar{y} - m\bar{x}$ ke (1):

$$\sum x_i y_i - m \sum x_i^2 - (\bar{y} - m\bar{x}) \sum x_i = 0.$$

Gunakan $\sum x_i = n\bar{x}$:

$$\sum x_i y_i - m \sum x_i^2 - \bar{y}(n\bar{x}) + m\bar{x}(n\bar{x}) = 0.$$

Susun suku-suku ber-m dan bukan-m:

$$\sum x_i y_i - n\bar{x}\bar{y} - m\left(\sum x_i^2 - n\bar{x}^2\right) = 0.$$

Sehingga

$$m = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}.$$

Jadi rumus m menjadi sangat sederhana:

$$\boxed{m = \frac{S_{xy}}{S_{xx}}}$$

dan sebelumnya kita sudah punya

$$\boxed{b = \bar{y} - m\bar{x}}.$$

```
# jumlah data
n = len(x)
# rata-rata
x_mean = np.mean(x)
y_mean = np.mean(y)
num = np.sum(x * y) - n * x_mean * y_mean
den = np.sum(x**2) - n * x_mean**2
m = num / den
b = y_mean - m * x_mean
```

3. PERSAMAAN GARIS LINEAR OLS MATRIX

$$B = (X^T X)^{-1} X^T y$$

Tujuan OLS

Kita ingin mencari parameter **B = [b₀, b₁, b₂, b₃]ᵀ** sehingga prediksi:

$$\hat{y} = XB$$

mendekati nilai sebenarnya **y** sebaik mungkin (error minimal).
Error (residual):

$$e = y - \hat{y} = y - XB$$

Untuk mendapatkan B terbaik, kita meminimalkan **jumlah kuadrat error (Least Squares)**:

$$J(B) = e^T e = (y - XB)^T (y - XB)$$

Menurunkan Rumus

Ambil turunan terhadap B, set ke nol untuk mencari titik minimum:

$$\frac{\partial}{\partial B}(y - XB)^T (y - XB) = 0$$

Hasilnya menjadi persamaan **Normal Equation**:

$$X^T XB = X^T y$$

Sehingga:

$$B = (X^T X)^{-1} X^T y$$

Implementasi dengan NumPy

```
XT = X.T
B = np.linalg.inv(XT @ X) @ XT @ y
y_new_pred = x_pred @ B
```

Dimensi Matriks agar Mudah Dipahami

Variabel	Bentuk	Keterangan
X	(n × 4)	n data, 3 fitur + kolom 1 untuk bias
y	(n × 1)	target output
X^T	(4 × n)	transpose
$X^T X$	(4 × 4)	square matrix, bisa dibalik
$(X^T X)^{-1}$	(4 × 4)	inverse
$(X^T X)^{-1} X^T$	(4 × n)	operator transform data
B	(4 × 1)	parameter hasil training
y_pred	(n × 1)	hasil prediksi

Contoh Konkret Perhitungan y_pred

Jika:

$$B = [b_0, b_1, b_2, b_3]^T$$

Dan baris pertama dari X:

$$[1, x_{11}, x_{12}, x_{13}]$$

Maka:

$$y_1 = b_0 + b_1 x_{11} + b_2 x_{12} + b_3 x_{13}$$

Begitu juga baris berikutnya:

$$y_2 = b_0 + b_1 x_{21} + b_2 x_{22} + b_3 x_{23}$$

Dan seterusnya sampai y_n.

Intuisi Visual

Bayangkan kita mencari garis atau bidang terbaik yang melewati sekumpulan titik.
OLS memilih parameter B yang membuat semua titik **paling dekat** ke bidang prediksi.

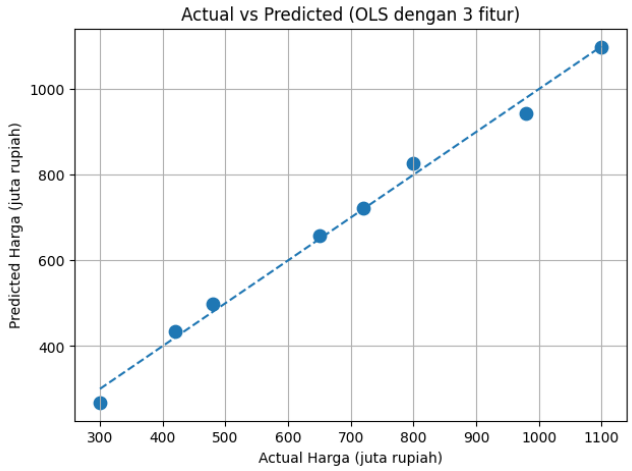
Kenapa harus pakai bentuk matriks?

1. Bisa menangani banyak fitur sekaligus
2. Komputasi cepat
3. Bisa digeneralisasi untuk ratusan fitur
4. Mudah diparalelkan dan dipakai di ML modern

Kesimpulan Sederhana

- OLS mencari parameter **B** terbaik
- Menggunakan minimisasi error kuadrat
- Solusi tertutupnya:

$$B = (X^T X)^{-1} X^T y$$



contoh lengkap bentuk Matrix X, y, dan B untuk regresi linear dengan 3 fitur.
Misalkan kita punya dataset kecil dengan **3 fitur**:

No	x_1 (Luas Rumah)	x_2 (Jumlah Kamar)	x_3 (Umur Bangunan)	y (Harga Rumah)
1	100	2	10	500
2	120	3	5	650
3	80	2	20	400
4	150	4	7	800

📌 Bentuk Matriks

Vector output y (n × 1)

$$y = \begin{bmatrix} 500 \\ 650 \\ 400 \\ 800 \end{bmatrix}$$

Matrix input X (n × 4):

Kolom pertama = **1** sebagai bias (intercept)

$$X = \begin{bmatrix} 1 & 100 & 2 & 10 \\ 1 & 120 & 3 & 5 \\ 1 & 80 & 2 & 20 \\ 1 & 150 & 4 & 7 \end{bmatrix}$$

Dimensi: **4 × 4** (4 data × 1 bias + 3 fitur)

Parameter (koefisien) yang ingin dicari

$$B = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- b_0 = intercept / bias
- b_1 = koef luas rumah
- b_2 = koef jumlah kamar
- b_3 = koef umur bangunan

Persamaan Matrix untuk prediksi

$$\hat{y} = XB$$

🔍 Ekspansi persamaan untuk masing-masing baris

$$y_1 = 1 \cdot b_0 + 100 \cdot b_1 + 2 \cdot b_2 + 10 \cdot b_3$$
$$y_2 = 1 \cdot b_0 + 120 \cdot b_1 + 3 \cdot b_2 + 5 \cdot b_3$$
$$y_3 = 1 \cdot b_0 + 80 \cdot b_1 + 2 \cdot b_2 + 20 \cdot b_3$$
$$y_4 = 1 \cdot b_0 + 150 \cdot b_1 + 4 \cdot b_2 + 7 \cdot b_3$$

Tujuan OLS

Mencari **B terbaik** dengan persamaan:

$$B = (X^T X)^{-1} X^T y$$

Kesimpulan

Dengan 3 fitur + 1 bias:

- X memiliki dimensi **n × 4**
- B memiliki dimensi **4 × 1**
- y memiliki dimensi **n × 1**
- Prediksi menggunakan **perkalian matrix**: $\hat{y} = XB$

CONTOH PREDIKSI HARGA RUMAH

DATA

	LUAS TANAH(M^2)	JUMLAH KAMAR	UMUR RUMAH	HARGA (JT)
1	50	2	10	300
2	80	3	5	420
3	120	3	20	650
4	150	4	15	800
5	200	5	30	1100
6	90	2	8	480
7	170	4	12	980
8	130	3	7	720

TUGAS PREDIKSI

	LUAS TANAH(M^2)	JUMLAH KAMAR	UMUR RUMAH	HARGA (JT)
1	100	3	10	??
2				

1. Import Library

```
import numpy as np
import matplotlib.pyplot as plt
```

2. Data Latih >> fitur x dan target y

```
# ---------- Contoh data ----------
# Fitur: [luas (m2), jumlah_kamar, umur_rumah (tahun)]
X_features = np.array([
    [50, 2, 10],
    [80, 3, 5],
    [120, 3, 20],
    [150, 4, 15],
    [200, 5, 30],
    [90, 2, 8],
    [170, 4, 12],
```

```
    [130, 3, 7]
], dtype=float)

# Target: harga rumah (juta rupiah)
y = np.array([300, 420, 650, 800, 1100, 480, 980, 720], dtype=float)
```

3. Normalisasi data Min-Max (0-1)

```
# ---------- Fungsi Normalisasi Min-Max ----------
def min_max_normalize(data):
    """Normalisasi data menggunakan metode Min-Max [0,1]"""
    min_vals = np.min(data, axis=0)
    max_vals = np.max(data, axis=0)
    normalized_data = (data - min_vals) / (max_vals - min_vals)
    return normalized_data, min_vals, max_vals

def min_max_denormalize(normalized_data, min_vals, max_vals):
    """Denormalisasi data dari range [0,1] ke nilai asli"""
    return normalized_data * (max_vals - min_vals) + min_vals

# ---------- Normalisasi Fitur ----------
print("=== NORMALISASI DATA ===")
X_normalized, X_min, X_max = min_max_normalize(X_features)
print("Data fitur asli:")
print(X_features)
print("\nData fitur ternormalisasi:")
print(X_normalized)
print(f"\nMin values: {X_min}")
print(f"Max values: {X_max}")

# ---------- Normalisasi Target ----------
y_normalized, y_min, y_max = min_max_normalize(y.reshape(-1, 1))
y_normalized = y_normalized.flatten()
print(f"\nTarget asli: {y}")
print(f"Target ternormalisasi: {y_normalized}")
print(f"Min target: {y_min[0]}, Max target: {y_max[0]}")
```

```
======================
=== NORMALISASI DATA ===
Data fitur asli:
[[ 50.   2.  10.]
 [ 80.   3.   5.]
 [120.   3.  20.]
 [150.   4.  15.]
 [200.   5.  30.]
 [ 90.   2.   8.]
 [170.   4.  12.]
 [130.   3.   7.]]

Data fitur ternormalisasi:
[[0.    0.    0.2  ]
 [0.2   0.333 0.   ]
```

```
 [0.467 0.333 0.6  ]
 [0.667 0.667 0.4  ]
 [1.    1.    1.   ]
 [0.267 0.    0.12 ]
 [0.8   0.667 0.28 ]
 [0.533 0.333 0.08 ]]

Min values: [50.  2.  5.]
Max values: [200.   5.  30.]

Target asli: [ 300.  420.  650.  800. 1100.  480.  980.  720.]
Target ternormalisasi: [0.     0.15   0.4375 0.625  1.     0.225  0.85   0.525 ]
Min target: 300.0, Max target: 1100.0
```

```
   4. Pers1iapan data
```

menambahkan kolom intercept (bias):

Konsep Intercept (Bias)

Intercept (bias) adalah koefisien `b0` dalam persamaan regresi linear:

text

```
y = b0 + b1*x1 + b2*x2 + ... + bn*xn
```

- `b0` adalah nilai y ketika semua variabel x = 0
- Penting untuk memberikan fleksibilitas pada model

Breakdown Kode:

python

```python
# ---------- Tambah kolom intercept (bias) ----------
n = X_normalized.shape[0]  # Jumlah sampel/data points
intercept = np.ones((n, 1))  # Membuat vektor berisi angka 1
X = np.hstack([intercept, X_normalized])  # Gabungkan intercept dengan data
```

1. `n = X_normalized.shape[0]`
   - `shape[0]` mengembalikan jumlah baris (sampel) dalam data
   - Contoh: jika `X_normalized` berukuran 100×3, maka `n = 100`
2. `intercept = np.ones((n, 1))`
   - Membuat array berisi angka 1 dengan dimensi n×1
   - Contoh untuk n=100: array [[1], [1], [1], ..., [1]] (100 baris, 1 kolom)
3. `np.hstack([intercept, X_normalized])`
   - Menggabungkan array secara horizontal (column-wise)
   - Hasilnya: kolom pertama semua 1, diikuti kolom-kolom data asli

Contoh Ilustrasi:

Sebelum:

```
X_normalized = [[0.1, 0.5],
                [0.3, 0.8],
                [0.7, 0.2]]
```

Setelah:

```
X = [[1.0, 0.1, 0.5],
     [1.0, 0.3, 0.8],
     [1.0, 0.7, 0.2]]
```

```
    ↑     ↑     ↑
   b0    b1    b2
```

Manfaat:

- Memungkinkan model memiliki intercept yang tidak harus melalui origin (0,0)
- Membuat model lebih fleksibel dalam menyesuaikan data
- Standar dalam implementasi regresi linear dan model linear lainnya

Dengan struktur ini, kita bisa menghitung prediksi menggunakan: `y_pred = X @ coefficients` dimana `coefficients = [b0, b1, b2, ...]`.

```
# ---------- Tambah kolom intercept (bias) ----------
n = X_normalized.shape[0]
intercept = np.ones((n, 1))
X = np.hstack([intercept, X_normalized])  # kolom pertama = 1 (b0)
print("Data fitur Persiapan data X:")
print(np.round(X, 3))
print(f"Target ternormalisasi y : {y_normalized}")
```

```
==============================
Data fitur Persiapan data X:
[[1.    0.    0.    0.2  ]
 [1.    0.2   0.333 0.   ]
 [1.    0.467 0.333 0.6  ]
 [1.    0.667 0.667 0.4  ]
 [1.    1.    1.    1.   ]
 [1.    0.267 0.    0.12 ]
 [1.    0.8   0.667 0.28 ]
 [1.    0.533 0.333 0.08 ]]
Target ternormalisasi y : [0.     0.15   0.4375 0.625  1.     0.225  0.85   0.525
]
```

5. OLS Matrix

```
# ---------- Hitung OLS (normal equation) ----------
print("\n=== REGRESI LINEAR DENGAN DATA NORMALISASI ===")
# XTX = Xᵀ × X
XT = X.T
XTX = XT @ X
print("XTX:",np.round(XTX, 3))
```

```
=== REGRESI LINEAR DENGAN DATA NORMALISASI ===
XTX: [[8.    3.933 3.333 2.68 ]
 [3.933 2.698 2.378 1.845]
 [3.333 2.378 2.222 1.68 ]
 [2.68  1.845 1.68  1.659]]
```

6. Mencari nilai B

```
# B_normalized berisi [b0, b1, b2, b3] untuk data ternormalisasi
# B = (XTX)⁻¹ × Xᵀ × y = [b0, b1, b2]
B_normalized = np.linalg.inv(XTX) @ XT @ y_normalized
print("B_normalized:",np.round(B_normalized, 3))

#B_normalized: [-0.038  1.085 -0.033 -0.015]
```

7. Prediksi menggunakan data awal utk mendapatkan hasil evaluasi

y_pred = b0*1 + b1*x1 + b2*x2 + b3*x3 + ...

```
# ---------- Prediksi dengan data ternormalisasi ----------
# y_pred = X × B
y_pred_normalized = X @ B_normalized
print("y_pred_normalized:",np.round(y_pred_normalized, 3))
================================
# y_pred_normalized: [-0.041  0.168  0.448  0.657  0.999  0.249  0.804  0.528]
```

8. Denormalisasi Prediksi

```
# ---------- Denormalisasi prediksi ----------
y_pred = min_max_denormalize(y_pred_normalized, y_min, y_max).flatten()
print("y_pred:",np.round(y_pred, 3))
================================
# y_pred: [ 267.168  434.457  658.493  825.782 1098.983  499.507  942.928
722.682]
```

9. Evaluasi

```
# ---------- Evaluasi dengan data asli ----------
mse = np.mean((y - y_pred)**2)
ss_tot = np.sum((y - np.mean(y))**2)
ss_res = np.sum((y - y_pred)**2)
r2 = 1 - ss_res / ss_tot

print(f"\n=== EVALUASI MODEL (DATA ASLI) ===")
print(f"MSE = {mse:.3f}")
print(f"R^2  = {r2:.3f}")


=== EVALUASI MODEL (DATA ASLI) ===
MSE = 473.356
R^2  = 0.993
```

Evaluasi dilakukan untuk mengukur seberapa baik model memprediksi data **asli** (bukan yang ternormalisasi).

1. Mean Squared Error (MSE)

```
mse = np.mean((y - y_pred)**2)
```

Rumus:

MSE = (1/n) × Σ(y_aktual - y_prediksi)²

Interpretasi:

- Mengukur rata-rata kuadrat selisih antara nilai aktual dan prediksi
- Semakin kecil MSE semakin baik
- Satuan sama dengan kuadrat satuan y (dalam contoh: dollar^2)

2. R-squared (R²) - Koefisien Determinasi

```
ss_tot = np.sum((y - np.mean(y))**2)  # Total Sum of Squares
ss_res = np.sum((y - y_pred)**2)      # Residual Sum of Squares
r2 = 1 - ss_res / ss_tot
```

Komponen:

- **SS_tot** (Total Sum of Squares): Variabilitas total data aktual
- **SS_res** (Residual Sum of Squares): Variabilitas yang tidak dijelaskan model
- **R^2**: Proporsi variabilitas yang dijelaskan model

Rumus:

R² = 1 - (SS_res / SS_tot)

Interpretasi Hasil:

```
=== EVALUASI MODEL (DATA ASLI) ===
MSE = 473.356
R^2  = 0.993
```

1. MSE = 473.356
   - Rata-rata kuadrat error adalah 473.356
   - Untuk konteks harga rumah (dalam ribuan dollar), ini berarti rata-rata error kuadrat adalah $473,356
   - **RMSE** (Root Mean Squared Error) = √473.356 ≈ $21.76 (error rata-rata)
2. R² = 0.993
   - **Nilai 0.993 sangat tinggi** (mendekati 1.0)
   - Artinya: **99.3% variasi dalam harga rumah** dapat dijelaskan oleh model regresi
   - Hanya **0.7% variasi** yang tidak dijelaskan model

Visualisasi Konsep:

```
Data Aktual (y):    [100, 200, 300, 400, 500]
Prediksi (y_pred):  [102, 198, 302, 398, 502]
Error:              [ -2,   2,  -2,   2,  -2]
```

Kualitas Model Berdasarkan R²:

- **R^2 = 0.993 → Excellent** (sangat baik)
- 0.8 - 0.99 → Very Good
- 0.6 - 0.8 → Good
- 0.4 - 0.6 → Fair

- < 0.4 → Poor

Contoh Perbandingan:

Model Sempurna:

- MSE = 0, R^2 = 1.0

Model Bagus:

- MSE = 473.356, R^2 = 0.993 ← **HASIL ANDA**

Model Buruk:

- MSE = 10,000, R^2 = 0.4

Kesimpulan untuk Model Anda:

1. **Sangat Akurat**: R^2 99.3% menunjukkan model menjelaskan hampir semua variasi data
2. **Error Kecil**: MSE 473.356 relatif kecil terhadap skala data
3. **Model Reliabel**: Dapat digunakan untuk prediksi dengan confidence tinggi

Catatan: Nilai R^2 yang sangat tinggi seperti ini bisa mengindikasikan:

- Model yang sangat well-fitted
- Kemungkinan overfitting (perlu dicek dengan data test)
- Data yang memiliki hubungan linear yang sangat kuat

`10.Prediksi` y_pred = X × B

kedua vektor di sini berbentuk **1D dengan panjang yang sama**, maka hasilnya adalah **perkalian dot product**:

$$[a, b, c]@[d, e, f] = a \times d + b \times e + c \times f$$

Contoh:

Jika

$$[a, b, c] = [1,2,3], [d, e, f] = [4,5,6]$$

Maka:

$$1 \times 4 + 2 \times 5 + 3 \times 6 = 4 + 10 + 18 = 32$$

✅ **Jadi hasil umumnya:**

$$[a, b, c]@[d, e, f] = a \cdot d + b \cdot e + c \cdot f$$

1. Bentuk Persamaan Regresi

Dalam bentuk skalar:

$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \varepsilon$

Dalam bentuk vektor:

$y = X \times B$

2.Dekomposisi Matriks

Vektor target y:

$y = [y_1, y_2, y_3, ..., y_n]^T$ *# n × 1*

Matrix fitur X (dengan intercept):

$X = [[1, x_{11}, x_{12}, x_{13}],$
$[1, x_{21}, x_{22}, x_{23}],$
$[1, x_{31}, x_{32}, x_{33}],$
$...$
$[1, x_{n1}, x_{n2}, x_{n3}]]$ *# n × 4*

Vektor koefisien B:

$B = [b_0, b_1, b_2, b_3]^T$ *# 4 × 1*

3. Operasi Perkalian Matriks

$y\_pred = X @ B$

Dijabarkan per baris:

$y_1 = 1 \times b_0 + x_{11} \times b_1 + x_{12} \times b_2 + x_{13} \times b_3$
$y_2 = 1 \times b_0 + x_{21} \times b_1 + x_{22} \times b_2 + x_{23} \times b_3$
$y_3 = 1 \times b_0 + x_{31} \times b_1 + x_{32} \times b_2 + x_{33} \times b_3$
$...$
$y_n = 1 \times b_0 + x_{n1} \times b_1 + x_{n2} \times b_2 + x_{n3} \times b_3$

4. Bentuk Umum untuk m Fitur

 # Untuk m fitur ($x_1, x_2, ..., x_m$)

$X = [[1, x_{11}, x_{12}, ..., x_{1m}],$
$[1, x_{21}, x_{22}, ..., x_{2m}],$
$...$
$[1, x_{n1}, x_{n2}, ..., x_{nm}]]$

$B = [b_0, b_1, b_2, ..., b_m]^T$

$y\_pred = X @ B = [b_0 + b_1x_{11} + b_2x_{12} + ... + b_mx_{1m},$
$b_0 + b_1x_{21} + b_2x_{22} + ... + b_mx_{2m},$
$...$
$b_0 + b_1x_{n1} + b_2x_{n2} + ... + b_mx_{nm}]$

5. Contoh Numerik Konkret

Misalkan kita memiliki:

 # Data fitur (setelah normalisasi)
$X = [[1, 0.1, 0.5, 0.3],$ *# sampel 1*
$[1, 0.3, 0.8, 0.6],$ *# sampel 2*
$[1, 0.7, 0.2, 0.9],$ *# sampel 3*
$[1, 0.4, 0.6, 0.1]]$ *# sampel 4*

 # Koefisien yang dihitung
$B = [b_0, b_1, b_2, b_3] = [0.5, 2.1, -1.3, 0.8]$

Perhitungan prediksi:

$y\_pred = X @ B$

Hasil per sampel:

Sampel 1: $y_1 = 1 \times 0.5 + 0.1 \times 2.1 + 0.5 \times (-1.3) + 0.3 \times 0.8 = 0.5 + 0.21 - 0.65 + 0.24 = 0.30$
Sampel 2: $y_2 = 1 \times 0.5 + 0.3 \times 2.1 + 0.8 \times (-1.3) + 0.6 \times 0.8 = 0.5 + 0.63 - 1.04 + 0.48 = 0.57$
Sampel 3: $y_3 = 1 \times 0.5 + 0.7 \times 2.1 + 0.2 \times (-1.3) + 0.9 \times 0.8 = 0.5 + 1.47 - 0.26 + 0.72 = 2.43$
Sampel 4: $y_4 = 1 \times 0.5 + 0.4 \times 2.1 + 0.6 \times (-1.3) + 0.1 \times 0.8 = 0.5 + 0.84 - 0.78 + 0.08 = 0.64$

```
# ---------- Contoh prediksi untuk data baru ----------
print("\n=== CONTOH PREDIKSI ===")
# Data baru yang akan diprediksi
x_new_features = np.array([100, 3, 10])  # [luas=100, kamar=3, umur=10]

# Normalisasi data baru menggunakan parameter yang sama
x_new_normalized = (x_new_features - X_min) / (X_max - X_min)

# Tambah intercept
x_new_with_intercept = np.hstack([1, x_new_normalized])

# Prediksi dengan model ternormalisasi
y_new_pred_normalized = x_new_with_intercept @ B_normalized

# Denormalisasi prediksi
y_new_pred = y_new_pred_normalized * (y_max - y_min) + y_min

print(f"Data baru: luas={x_new_features[0]}m2, kamar={x_new_features[1]},
umur={x_new_features[2]}th")
print(f"Data baru ternormalisasi: {np.round(x_new_normalized, 3)}")
print(f"x_new_with_intercept: {np.round(x_new_with_intercept, 3)}")
print(f"B_normalized: {np.round(B_normalized, 3)}")
print(f"Prediksi ternormalisasi: x_new_with_intercept @ B_normalized =
{y_new_pred_normalized:.4f}")
print(f"Prediksi harga = {y_new_pred[0]:.2f} juta rupiah")
```

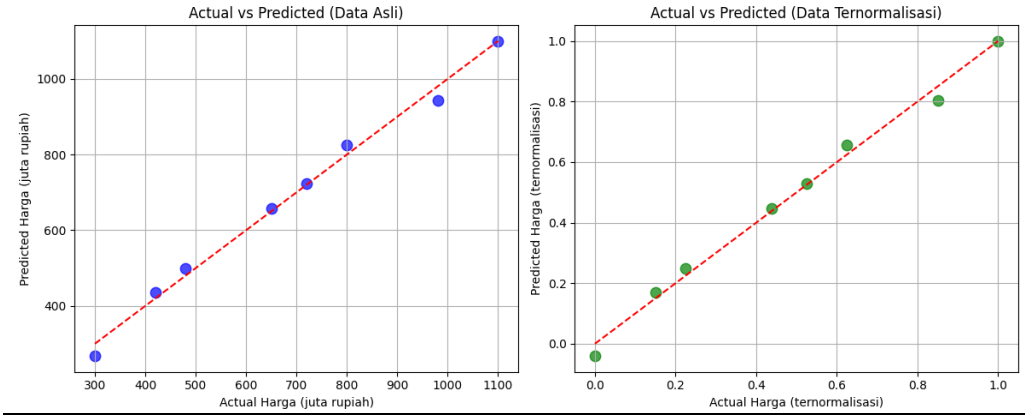
```
=== CONTOH PREDIKSI ===
Data baru: luas=100m2, kamar=3, umur=10th
Data baru ternormalisasi: [0.333 0.333 0.2  ]
x_new_with_intercept: [1.    0.333 0.333 0.2  ]
B_normalized: [-0.038  1.085 -0.033 -0.015]
Prediksi ternormalisasi: x_new_with_intercept @ B_normalized = 0.3096
Prediksi harga = 547.70 juta rupiah
```

11.Visualisasi

```
# ---------- Visualisasi: Actual vs Predicted ----------
plt.figure(figsize=(12, 5))

# Subplot 1: Data asli
plt.subplot(1, 2, 1)
plt.scatter(y, y_pred, s=80, color='blue', alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], linestyle='--', color='red')
plt.xlabel("Actual Harga (juta rupiah)")
```

```
plt.ylabel("Predicted Harga (juta rupiah)")
plt.title("Actual vs Predicted (Data Asli)")
plt.grid(True)

# Subplot 2: Data ternormalisasi
plt.subplot(1, 2, 2)
plt.scatter(y_normalized, y_pred_normalized, s=80, color='green', alpha=0.7)
plt.plot([y_normalized.min(), y_normalized.max()],
         [y_normalized.min(), y_normalized.max()], linestyle='--', color='red')
plt.xlabel("Actual Harga (ternormalisasi)")
plt.ylabel("Predicted Harga (ternormalisasi)")
plt.title("Actual vs Predicted (Data Ternormalisasi)")
plt.grid(True)

plt.tight_layout()
plt.show()

# ---------- Perbandingan dengan model tanpa normalisasi ----------
print("\n=== PERBANDINGAN DENGAN MODEL TANPA NORMALISASI ===")
# Model tanpa normalisasi (seperti script asli)
X_original = np.hstack([np.ones((n, 1)), X_features])
if np.linalg.cond(X_original.T @ X_original) < 1 / np.finfo(XTX.dtype).eps:
    B_original = np.linalg.inv(X_original.T @ X_original) @ X_original.T @ y
else:
    B_original = np.linalg.pinv(X_original) @ y

y_pred_original = X_original @ B_original
mse_original = np.mean((y - y_pred_original)**2)

print("Koefisien model tanpa normalisasi:")
for i, coef in enumerate(B_original):
    if i == 0:
        print(f" b0 (intercept) = {coef:.4f}")
    else:
        print(f" b{i} = {coef:.4f}")

print(f"MSE tanpa normalisasi: {mse_original:.3f}")
print(f"MSE dengan normalisasi: {mse:.3f}")
```

```
=======================================================
=== PERBANDINGAN DENGAN MODEL TANPA NORMALISASI ===
Koefisien model tanpa normalisasi:
 b0 (intercept) = 0.1992
 b1 = 5.7840
 b2 = -8.6742
 b3 = -0.4884
MSE tanpa normalisasi: 473.356
MSE dengan normalisasi: 473.356
```



DASAR TEORI OLS MATRIX

Sekarang kita punya **dua fitur (x_1 dan x_2)**, jadi persamaan regresinya:

$$y_i = b_0 + b_1 x_{i1} + b_2 x_{i2}$$

Mari kita tulis dalam bentuk matriks dan lihat proses perkaliannya.

2 Matriks Desain X

Sekarang kita perlu **tiga kolom**:

- **Kolom 1**: semua 1 (untuk intercept b_0)
- **Kolom 2**: nilai fitur pertama x_{i1}
- **Kolom 3**: nilai fitur kedua x_{i2}

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{bmatrix}$$

Ukuran X adalah $n \times 3$.

3 Vektor Koefisien B

$$B = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Ukuran 3×1.

4 Vektor Target Y

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Ukuran $n \times 1$.

5 Perkalian Matriks

Perkalian XB sah karena:

- Kolom X = 3
- Baris B = 3

→ hasil $n \times 1$.

Baris ke-i dari hasil:

$$(XB)_i = [1,\ x_{i1},\ x_{i2}] \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = b_0 + b_1 x_{i1} + b_2 x_{i2}$$

Jadi:

$$XB = \begin{bmatrix} b_0 + b_1 x_{11} + b_2 x_{12} \\ b_0 + b_1 x_{21} + b_2 x_{22} \\ \vdots \\ b_0 + b_1 x_{n1} + b_2 x_{n2} \end{bmatrix}$$

Mulai dari ide dasar regresi

Kita punya data:

$$y_i \quad \text{(nilai asli/observasi)}$$

$$\hat{y}_i = x_i^T \beta \quad \text{(prediksi dari model linear)}$$

Error (residual) tiap titik:

$$e_i = y_i - \hat{y}_i = y_i - x_i^T \beta$$

Kenapa pakai kuadrat error?

Kalau hanya menjumlahkan error:

$$\sum_i e_i = \sum_i (y_i - x_i^T \beta)$$

→ ini bisa nol karena error positif & negatif bisa saling meniadakan.
Maka dipilih **kuadrat error**:

$$J(\beta) = \sum_{i=1}^{n} e_i^2$$

Itulah prinsip **Ordinary Least Squares (OLS)** → meminimalkan jumlah kuadrat error.

Fungsi Kerugian (Loss Function) yang paling dasar adalah **Sum of Squared Errors (SSE)**, yang dirumuskan sebagai:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} e_i^2$$

Ternyata, ini persis sama dengan $\mathbf{e}^T\mathbf{e}$!

$$\text{SSE} = \mathbf{e}^T\mathbf{e}$$

Apa itu $\mathbf{e}^T\mathbf{e}$?

$\mathbf{e}^T\mathbf{e}$ adalah hasil perkalian dot product (perkalian matrix) antara transpose sebuah vektor $\mathbf{e}$ dengan vektor $\mathbf{e}$ itu sendiri.

---

1. Breaking Down Komponennya

- $\mathbf{e}$ adalah sebuah **vektor kolom** yang berisi `n` elemen (atau error/kesalahan).

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

- $\mathbf{e}^T$ adalah **transpose** dari vektor $\mathbf{e}$. Transpose mengubah vektor kolom menjadi **vektor baris**.

$$\mathbf{e}^T = \begin{bmatrix} e_1, e_2, \ldots, e_n \end{bmatrix}$$

2. Melakukan Perkalian $\mathbf{e}^T\mathbf{e}$

Sekarang kita kalikan vektor baris $\mathbf{e}^T$ dengan vektor kolom $\mathbf{e}$.

Aturan Perkalian: Karena $\mathbf{e}^T$ berukuran `(1 x n)` dan $\mathbf{e}$ berukuran `(n x 1)`, hasil perkaliannya akan menjadi sebuah **scalar** (bilangan tunggal) dengan ukuran `(1 x 1)`.

Cara Kalkulasi (Dot Product):

$$\mathbf{e}^T\mathbf{e} = \begin{bmatrix} e_1, e_2, \dots, e_n \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = (e_1 \times e_1) + (e_2 \times e_2) + \dots + (e_n \times e_n)$$

$$\mathbf{e}^T\mathbf{e} = e_1^2 + e_2^2 + \cdots + e_n^2 = \sum_{i=1}^{n} e_i^2$$

Jadi, $\mathbf{e}^T\mathbf{e}$ adalah *penjumlahan dari kuadrat* setiap elemen dalam vektor $\mathbf{e}$.

- **OLS** = cari parameter β yang membuat total kuadrat error minimum.
- Error didefinisikan: $e = y - X\beta$.
- Jumlah kuadrat error → ditulis ringkas:

$$J(\beta) = e^T e = (y - X\beta)^T (y - X\beta)$$

4. Ekspansi fungsi error

Kembangkan:

$$J(\beta) = (y^T - \beta^T X^T)(y - X\beta)$$
$$= y^T y - y^T X\beta - \beta^T X^T y + \beta^T X^T X\beta$$

Karena $y^T X\beta$ dan $\beta^T X^T y$ adalah skalar yang sama, maka:

$$J(\beta) = y^T y - 2\beta^T X^T y + \beta^T X^T X\beta$$

Kita punya tiga objek:

- y : **vektor kolom** ukuran $n \times 1$
- X : **matriks desain** ukuran $n \times p$
- B : **vektor kolom** ukuran $p \times 1$

Kita ingin membandingkan dua bentuk:

$$\boxed{y^T XB} \quad \text{dan} \quad \boxed{B^T X^T y}$$

dan menunjukkan bahwa keduanya adalah **skalar yang sama**.

1️⃣ Periksa Ukuran

- y^T: $1 \times n$
- X: $n \times p$
- B: $p \times 1$

Urutan $y^T XB$:

- $y^T X \rightarrow$ ukuran $1 \times p$
- $(y^T X)B \rightarrow$ ukuran 1×1 (skalar)

Urutan $B^T X^T y$:

- $X^T y \rightarrow$ ukuran $p \times 1$
- $B^T (X^T y) \rightarrow$ ukuran 1×1 (skalar)

Keduanya jelas menghasilkan **skalar**.

2️⃣ Sifat Transposisi

Ingat sifat transpose:

$$(AB)^T = B^T A^T$$

Maka

$$(y^T XB)^T = B^T X^T y$$

Tetapi $y^T XB$ adalah skalar 1×1.
Untuk skalar, transpose tidak mengubah nilainya:

$$s^T = s$$

Sehingga

$$y^T XB = (y^T XB)^T = B^T X^T y$$

3️⃣ Intuisi

Secara "dot product":

- XB adalah vektor prediksi $n \times 1$.
- $y^T (XB)$ berarti dot product antara y dan XB.
- $B^T (X^T y)$ berarti dot product antara B dan $X^T y$.

Dot product **bersifat komutatif** bila dikemas dengan transpose yang tepat, jadi hasilnya identik.

✅ Kesimpulan

Karena hasilnya skalar, dan karena $(y^T XB)^T = B^T X^T y$, maka

$$\boxed{y^T XB = B^T X^T y}.$$

1 Catat Ukuran

- B: $p \times 1$ (vektor kolom)
- X: $n \times p$
- y: $n \times 1$

---

2 Turunan Tiap Suku

a. $y^T y$

- Tidak bergantung pada B.
- Turunan w.r.t. B = **0**.

---

b. $-2B^T X^T y$

Gunakan fakta:

$$\frac{\partial}{\partial B}(B^T c) = c$$

untuk vektor konstan c.

Di sini $c = X^T y$, jadi

$$\frac{\partial}{\partial B}\left[-2B^T X^T y\right] = -2X^T y.$$

Rumus sakti:

d(uv)/dx = u'v + uv'

Anggap $u(B) = B^T$, $v(B) = AB$, maka $f = uv$ (perkalian skalar? tidak, tapi hati-hati).

Lebih baik pakai aturan:

$$d(B^T AB) = (dB)^T AB + B^T A(dB)$$

Sekarang, $(dB)^T AB$ adalah skalar, transpose tidak mengubah nilai skalar:

$(dB)^T AB = B^T A^T dB$?? Hati-hati dengan layout.

Mari kita terapkan aturan:
Jika $f = B^T MB$, maka $df = (dB)^T MB + B^T MdB$.

Tapi $(dB)^T MB$ adalah skalar, sama dengan transpose-nya: $B^T M^T dB$.

Jadi:

$$df = B^T M^T dB + B^T MdB = B^T (M^T + M)dB$$

Dalam kasus kita $M = A$, jadi:

$$df = B^T (A^T + A)dB$$

Dalam kalkulus matriks (layout numerator), gradien $\frac{\partial f}{\partial B}$ adalah transpose dari koefisien dB, yaitu:

$$\frac{\partial f}{\partial B} = (A^T + A)B$$

c. $B^T X^T XB$

Gunakan hasil standar:

$$\frac{\partial}{\partial B}\left(B^T AB\right) = (A + A^T)B.$$

Karena $A = X^T X$ dan itu **simetris** ($A^T = A$),

$$\frac{\partial}{\partial B}\left(B^T X^T XB\right) = 2X^T XB.$$

---

3 Gabungkan

Gradien total:

$$\nabla_B J(B) = 0 \; - \; 2X^T y \; + \; 2X^T XB$$

atau disederhanakan:

$$\boxed{\nabla_B J(B) = 2X^T XB - 2X^T y}.$$

4️⃣ Interpretasi

- Titik minimum diperoleh dengan menset gradien = 0:

$$2X^TXB - 2X^Ty = 0$$
$$\implies X^TXB = X^Ty$$
$$\implies \hat{B} = (X^TX)^{-1}X^Ty$$

(asumsi X^TX invertibel).

Itu adalah **normal equation** untuk OLS regresi linear.

7. Solusi untuk β

Kalau X^TX bisa di-invers:

$$\beta = (X^TX)^{-1}X^Ty$$

👉 Inilah yang disebut **Normal Equation** — rumus OLS umum.

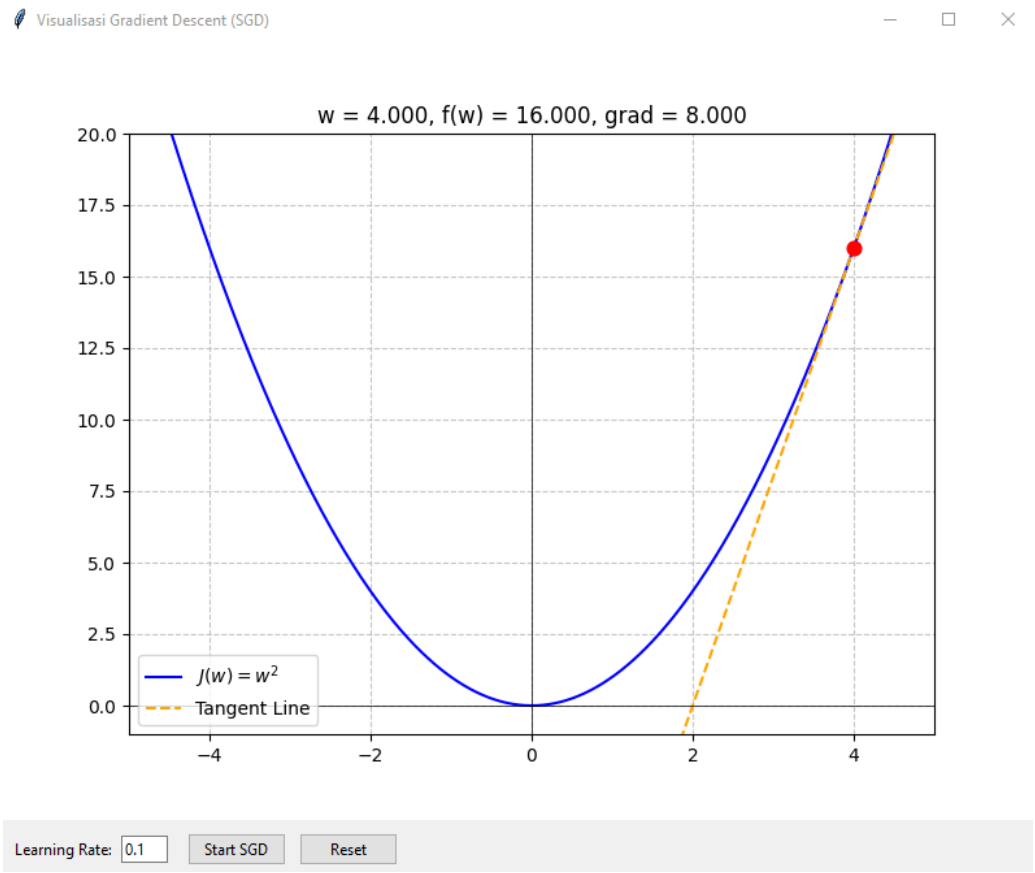
🔑 Kesimpulan

- Mulai dari **model linear**: $y = X\beta + e$.
- Definisikan **error kuadrat total**: $J(\beta) = (y - X\beta)^T(y - X\beta)$.
- **Turunkan** fungsi tersebut → dapat persamaan normal.
- Selesaikan → ketemu rumus:

$$\beta = (X^TX)^{-1}X^Ty$$

```
# ---------- Hitung OLS (normal equation) ----------
XT = X.T
XTX = XT @ X
B = np.linalg.inv(XTX) @ XT @ y

# ---------- Prediksi ----------
y_pred = X @ B
```

4. $f(x) = x^2$



1. Fungsi Objective (Target)

$$f(x) = x^2$$

Tujuan: Mencari nilai x yang meminimalkan $f(x)$

---

2. Konsep Error dan Loss

$$\text{target} = 0 (\text{kita ingin x mendekati } 0)$$
$$\text{error} = x - \text{target} = x - 0 = x$$
$$\text{loss} = (\text{error})^2 = x^2 = f(x)$$

---

3. Turunan (Gradient)

Turunan pertama dari $f(x)$ terhadap x:

$$f'(x) = \frac{d}{dx}(x^2) = 2x$$

Gradient:

$$\nabla f(x) = 2x$$

---

4. Update Rule (Aturan Pembaruan)

Formula Umum Gradient Descent:

$$x_{\text{new}} = x_{\text{old}} - \eta \cdot \nabla f(x_{\text{old}})$$

Substitusi untuk f(x) = x^2:

$$x_{\text{new}} = x_{\text{old}} - \eta \cdot (2x_{\text{old}})$$

---

5. Notasi dalam Programming

```
# Inisialisasi
x = x_initial
learning_rate = η

# Loop gradient descent
for epoch in range(num_epochs):
    gradient = 2 * x          # ∇f(x) = 2x
    x = x - learning_rate * gradient  # x_new = x_old - η·∇f(x)
```

---

6. Contoh Numerik Lengkap

Dengan: $x_0 = 3, \eta = 0.1$

Iterasi 1**:**

$$x_0 = 3$$
$$\nabla f(x_0) = 2 \times 3 = 6$$
$$x_1 = 3 - 0.1 \times 6 = 3 - 0.6 = 2.4$$

Iterasi 2**:**

$$x_1 = 2.4$$
$$\nabla f(x_1) = 2 \times 2.4 = 4.8$$
$$x_2 = 2.4 - 0.1 \times 4.8 = 2.4 - 0.48 = 1.92$$

Iterasi 3**:**

$$x_2 = 1.92$$
$$\nabla f(x_2) = 2 \times 1.92 = 3.84$$
$$x_3 = 1.92 - 0.1 \times 3.84 = 1.92 - 0.384 = 1.536$$

---

7. Deret Matematika Lengkap

Secara umum, setelah n iterasi:

$$x_n = x_0 \times (1 - 2\eta)^n$$

Bukti:

- $x_1 = x_0 - 2\eta x_0 = x_0(1 - 2\eta)$
- $x_2 = x_1(1 - 2\eta) = x_0(1 - 2\eta)^2$
- $x_3 = x_2(1 - 2\eta) = x_0(1 - 2\eta)^3$
- ...

- $x_n = x_0(1 - 2\eta)^n$

---

8. Kondisi Konvergensi

Agar gradient descent **konvergen** (tidak divergen):

$$|\ 1 - 2\eta\ | < 1$$
$$-1 < 1 - 2\eta < 1$$
$$0 < \eta < 1$$

Artinya: learning rate harus antara 0 dan 1 agar konvergen ke minimum.

---

9. Visualisasi Proses

```
Iterasi:    x       f(x)     ∇f(x)    Update
0          3.000    9.000    6.000    -0.600
1          2.400    5.760    4.800    -0.480
2          1.920    3.686    3.840    -0.384
3          1.536    2.359    3.072    -0.307
4          1.229    1.510    2.458    -0.246
5          0.983    0.966    1.966    -0.197
...        ...      ...      ...      ...
→ Menuju x = 0, f(x) = 0
```

Dengan memahami matematika di balik **f(x) = x²** ini, Anda telah menguasai fundamental dari **backpropagation** yang digunakan dalam neural networks!

10. Gradient Descent sebagai "Pencarian Buta"

```python
# Anda sedang berdiri di posisi x = 3
x = 3                    # Posisi awal Anda
learning_rate = 0.1      # "Langkah kaki" Anda
target = 0               # Target kita (titik terendah)

for step in range(5):
    # 1. Rasakan kemiringan tanah (hitung gradient)
    gradient = 2 * x     # f'(x) = 2x

    # 2. Tentukan arah berjalan
    # Jika gradient positif → tanah menanjak ke kanan → jalan ke kiri
    # Jika gradient negatif → tanah menanjak ke kiri → jalan ke kanan
    direction = -gradient

    # 3. Ambil langkah
    step_size = learning_rate * gradient
    x = x - step_size    # x_new = x - η * gradient

    # 4. Lihat posisi baru
    loss = x * x
    print(f"Step {step+1}: x = {x:.3f}, gradient = {gradient:.3f}, loss =
{loss:.3f}")
```

Output:

```
Step 1: x = 2.400, gradient = 6.000, loss = 5.760
Step 2: x = 1.920, gradient = 4.800, loss = 3.686
Step 3: x = 1.536, gradient = 3.840, loss = 2.359
```

```
Step 4: x = 1.229, gradient = 3.072, loss = 1.510
Step 5: x = 0.983, gradient = 2.458, loss = 0.966
```

2. Visualisasi Prosesnya

```
x-axis: -3 -2 -1  0  1  2  3

Step 1:          🚶        (x=3, loss=9)
Step 2:       🚶           (x=2.4, loss=5.76)
Step 3:    🚶              (x=1.92, loss=3.69)
Step 4:  🚶                (x=1.54, loss=2.37)
Step 5:🚶                 (x=1.23, loss=1.51)
```

3. Koneksi ke Machine Learning

```
# Kasus Linear Regression vs f(x)=x²
# ==================================

# LINEAR REGRESSION:
# y_pred = w*x + b
# error = y_pred - y_true
# loss = error² = (w*x + b - y_true)²

# f(x) = x²:
# "prediction" = x
# "target" = 0
# error = x - 0 = x
# loss = error² = x²
```

4. Mengapa Turunan f'(x) = 2x?

Cara intuitif memahami turunan:

- Turunan = "seberapa sensitif loss terhadap perubahan x"
- Di x = 3: loss = 9, jika x naik jadi 3.1 → loss = 9.61 (naik 0.61)
- Di x = 1: loss = 1, jika x naik jadi 1.1 → loss = 1.21 (naik 0.21)
- Di x = 0: loss = 0, jika x naik jadi 0.1 → loss = 0.01 (naik sangat kecil)

Pattern: Perubahan loss lebih besar di x yang besar → gradient = 2x

5. Learning Rate (η) - "Ukuran Langkah"

```
# Learning Rate terlalu besar (η = 1.0)
x = 3
gradient = 2 * 3 = 6
x_new = 3 - 1.0 * 6 = -3  # Terlalu jauh! Bolak-balik tidak konvergen

# Learning Rate terlalu kecil (η = 0.01)
x = 3
gradient = 6
x_new = 3 - 0.01 * 6 = 2.94  # Perlahan sekali, butuh banyak step

# Learning Rate optimal (η = 0.1)
x = 3
gradient = 6
x_new = 3 - 0.1 * 6 = 2.4  # Cukup cepat menuju minimum
```

6. Exercise Praktis

Coba implementasikan ini dan lihat bagaimana perilaku gradient descent:

```
def gradient_descent_viz(start_x, learning_rate, steps):
    x = start_x
    print(f"Starting at x = {x}, loss = {x*x}")

    for i in range(steps):
        gradient = 2 * x
        x_old = x
        x = x - learning_rate * gradient
        loss = x * x

        print(f"Step {i+1}:")
        print(f"  Gradient at x={x_old:.2f}: {gradient:.2f}")
        print(f"  Step size: {learning_rate * gradient:.2f}")
        print(f"  New x: {x:.2f}, New loss: {loss:.2f}")
        print(f"  {'→' if x < x_old else '←'} Moving {'left' if x < x_old else
'right'}")
        print()

# Coba dengan parameter berbeda
gradient_descent_viz(start_x=3, learning_rate=0.1, steps=5)
```

Dengan memahami $f(x) = x^2$ ini, Anda sudah menguasai 80% konsep gradient descent yang digunakan di neural networks!

```
#COPY - PASTE
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
from tkinter import ttk

# Fungsi Loss dan Turunan
def f(w):
    return w**2

def df(w):
    return 2 * w

# Garis singgung
def tangent_line(x, a):
    return df(a) * (x - a) + f(a)

class SGDVisualizer:
    def __init__(self, root):
        self.root = root
        self.root.title("Visualisasi Gradient Descent (SGD)")

        # Setup figure
```

```
        self.fig, self.ax = plt.subplots(figsize=(8, 6))
        self.canvas = FigureCanvasTkAgg(self.fig, master=root)
        self.canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

        # Kontrol frame
        control_frame = ttk.Frame(root)
        control_frame.pack(side=tk.BOTTOM, fill=tk.X, padx=10, pady=10)

        # Learning rate
        ttk.Label(control_frame, text="Learning Rate:").pack(side=tk.LEFT)
        self.lr_var = tk.DoubleVar(value=0.1)
        self.lr_entry = ttk.Entry(control_frame, textvariable=self.lr_var,
width=5)
        self.lr_entry.pack(side=tk.LEFT, padx=5)

        # Tombol start
        self.start_btn = ttk.Button(control_frame, text="Start SGD",
command=self.start_sgd)
        self.start_btn.pack(side=tk.LEFT, padx=10)

        # Reset
        self.reset_btn = ttk.Button(control_frame, text="Reset",
command=self.reset)
        self.reset_btn.pack(side=tk.LEFT)

        # Variabel SGD
        self.x = np.linspace(-5, 5, 400)
        self.y = f(self.x)
        self.w = 4.0   # titik awal
        self.running = False

        # Plot awal
        self.plot()

    def plot(self):
        self.ax.clear()
        self.ax.plot(self.x, self.y, label="$J(w) = w^2$", color="blue")

        # Titik saat ini
        self.ax.scatter(self.w, f(self.w), color="red", s=60, zorder=5)

        # Garis singgung
        tangent = tangent_line(self.x, self.w)
        self.ax.plot(self.x, tangent, "--", color="orange", label="Tangent Line")
```

```
        # Info
        self.ax.set_title(f"w = {self.w:.3f}, f(w) = {f(self.w):.3f}, grad =
{df(self.w):.3f}")
        self.ax.axhline(0, color="black", linewidth=0.5)
        self.ax.axvline(0, color="black", linewidth=0.5)
        self.ax.set_xlim(-5, 5)
        self.ax.set_ylim(-1, 20)
        self.ax.legend()
        self.ax.grid(linestyle="--", alpha=0.7)

        self.canvas.draw()

    def step(self):
        if not self.running:
            return
        lr = self.lr_var.get()
        grad = df(self.w)
        self.w = self.w - lr * grad   # Update rule: w ← w - η * grad

        self.plot()

        # Stop kalau sudah dekat 0
        if abs(self.w) < 0.01:
            self.running = False
            return

        # Loop animasi
        self.root.after(500, self.step)

    def start_sgd(self):
        if not self.running:
            self.running = True
            self.step()

    def reset(self):
        self.running = False
        self.w = 4.0
        self.plot()

if __name__ == "__main__":
    root = tk.Tk()
    app = SGDVisualizer(root)
    root.mainloop()
```

5. GRADIEN DECENT

1. Mulai dari fungsi garis

$$\hat{y} = mx + b$$

Kita ingin membuat garis ini mendekati data (x, y).

Pada saat kita menggunakan data yang banyak 2 fitur/2d data(x,y) atau bahkan banyak fitur atau ndimensi data. Dan jumlah data yang sangat banyak maka untuk menggambar garis tersebut sangat sulit dengan metode OLS maka menggunakan metode lain yaitu MSE dan SGD dengan cara memprediksi garis dengan membandingkan data yang di berikan.

- y_i = nilai asli, $\hat{y}_i$ = prediksi garis.

---

2. Definisikan error (selisih)

$$\text{error} = y - \hat{y} = y - (mx + b)$$

Error ini dipakai di dalam fungsi biaya (loss function).

---

3. Fungsi biaya (Mean Squared Error, MSE)

Untuk satu titik data:

$$J(m, b) = (y - \hat{y})^2$$

Kalau banyak data, biasanya dirata-rata ($\frac{1}{N}\sum$), tapi di *stochastic gradient descent* kita cukup ambil satu titik per langkah.

Fungsi Jmengukur seberapa baik garis regresi $y = mx + b$cocok dengan data.

- Kuadrat biar error negatif tidak meng-cancel error positif.

---

4. Turunan pakai Power Rule

Dari

$$J(m, b) = (y - (mx + b))^2$$

Gunakan power rule: turunan $(\_\_)^2 \to 2(\_\_)$.

Kita punya fungsi biaya (untuk **1 titik data saja**):

$$J(m, b) = (y - (mx + b))^2$$

dengan:

- y = target asli,
- $mx + b$ = prediksi model,
- m = slope,
- b = bias/intercept.

Tujuan Fungsi LOST/COST: mencari **nilai m dan b** yang membuat fungsi ini **sekecil mungkin** (artinya garis mendekati titik data).

🔷 Turunan J terhadap m

1. Tulis ulang:

$$J(m, b) = (y - (mx + b))^2$$

2. Misalkan

$$E = y - (mx + b)$$

maka

$$J = E^2$$

3. Turunkan J terhadap m:

$$\frac{\partial J}{\partial m} = 2E \cdot \frac{\partial E}{\partial m}$$

4. Karena

$$E = y - (mx + b) \quad \Rightarrow \quad \frac{\partial E}{\partial m} = -x$$

🔷 2. Turunan Error terhadap m

$$\frac{\partial E}{\partial m} = \frac{\partial}{\partial m}(y - (mx + b))$$

- y adalah konstanta → turunan = 0
- $-(mx)$ diturunkan terhadap m → $-x$
- $-b$ adalah konstanta (tidak tergantung m) → turunan = 0

Sehingga:

$$\frac{\partial E}{\partial m} = -x$$

5. Substitusi:

$$\frac{\partial J}{\partial m} = 2(y - (mx + b)) \cdot (-x)$$

6. Sederhanakan:

$$\frac{\partial J}{\partial m} = -2x(y - (mx + b))$$

$$\frac{\partial J}{\partial m} = -2x(y - (mx + b)) = -2x \cdot error$$

🔷 Intuisi

- Jika prediksi (mx+b) terlalu **kecil dibanding y**, maka turunan positif → m harus dinaikkan.
- Jika prediksi terlalu **besar**, turunan negatif → m harus diturunkan.

Inilah dasar kenapa **gradient descent** bisa “mengarahkan” m ke nilai optimal.

$$J(m, b) = (y - (mx + b))^2$$

🔷 Turunan J terhadap b

1. Ulangi definisi error:

$$E = y - (mx + b)$$

$$J = E^2$$

2. Turunkan J terhadap b:

$$\frac{\partial J}{\partial b} = 2E \cdot \frac{\partial E}{\partial b}$$

3. Karena

$$E = y - (mx + b) \quad \Rightarrow \quad \frac{\partial E}{\partial b} = -1$$

4. Substitusi:

$$\frac{\partial J}{\partial b} = 2(y - (mx + b)) \cdot (-1)$$

5. Sederhanakan:

$$\frac{\partial J}{\partial b} = -2\big(y - (mx + b)\big)$$

atau:

$$\frac{\partial J}{\partial b} = 2\big((mx + b) - y\big)$$

🔷 3. Turunan Error terhadap b

$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b}\big(y - (mx + b)\big)$$

- y konstanta → 0
- $-(mx)$ tidak bergantung pada b → 0
- $-b$ → turunan = -1

$$\frac{\partial E}{\partial b} = -1$$

🔷 Ringkasan Gradien

Untuk **1 data point (x, y)**:

$$\frac{\partial J}{\partial m} = -2x(y - (mx + b))$$

$$\frac{\partial J}{\partial b} = -2(y - (mx + b))$$

🔷 4. Hubungan ke Loss (MSE)

Biasanya yang kita minimalkan adalah **MSE (Mean Squared Error)**:

$$J = E^2 = (y - (mx + b))^2$$

Turunan:

$$\frac{\partial J}{\partial m} = 2E \cdot \frac{\partial E}{\partial m} = 2E(-x) = -2xE$$
$$\frac{\partial J}{\partial b} = 2E \cdot \frac{\partial E}{\partial b} = 2E(-1) = -2E$$

Definisikan dulu:

$$error = y - (mx + b)$$

Sehingga:

$$\frac{\partial J}{\partial m} = -2x \cdot error$$
$$\frac{\partial J}{\partial b} = -2 \cdot error$$

🔷 3. Aturan Update (Gradient Descent)

Secara umum:

$$parameter \leftarrow parameter - \eta \cdot \frac{\partial J}{\partial parameter}$$

khusus untuk m dan b:

$$m \leftarrow m - \eta \cdot \frac{\partial J}{\partial m}$$
$$b \leftarrow b - \eta \cdot \frac{\partial J}{\partial b}$$

dengan:

- η = **learning rate** (kecepatan belajar, seberapa besar langkah tiap update)
- $\frac{\partial J}{\partial m}, \frac{\partial J}{\partial b}$ = arah gradien

🔷 4. Substitusi

Masukkan hasil turunan:

$$m \leftarrow m - \eta(-2x \cdot error) = m + 2\eta\, x\, error$$
$$b \leftarrow b - \eta(-2 \cdot error) = b + 2\eta\, error$$

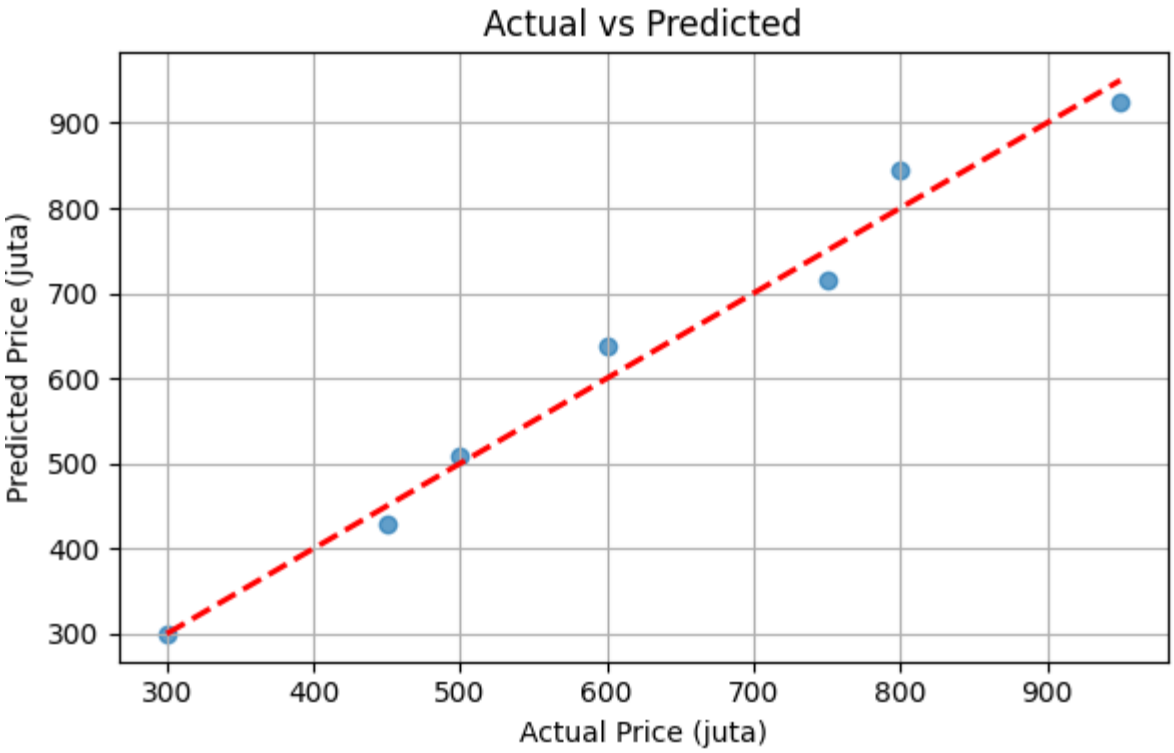
👉 inilah versi lengkapnya.

Kalau faktor 2 dimasukkan ke learning rate, hasilnya jadi:

$$m \leftarrow m + \eta\, x\, error$$
$$b \leftarrow b + \eta\, error$$

Dalam banyak implementasi, **konstanta 2** sering diabaikan dengan cara **menyerapnya ke dalam learning rate**.

```
m += error * x * learning_rate
b += error * learning_rate
```



CONTOH PREDIKSI HARGA RUMAH

DATA

	LUAS TANAH(M^2)	JUMLAH KAMAR	HARGA (JT)
1	60	2	300
2	80	3	450
3	100	3	500
4	120	4	600
5	140	4	750
6	160	5	800
7	180	5	950
8			

TUGAS PREDIKSI

	LUAS TANAH(M^2)	JUMLAH KAMAR	HARGA (JT)
1	90	3	??
2	150	4	??
3	200	5	??

```
import numpy as np
import matplotlib.pyplot as plt

# Dataset: [luas_tanah(m²), jumlah_kamar, harga(juta)]
data = np.array([
    [60, 2, 300],
```

```
    [80, 3, 450],
    [100, 3, 500],
    [120, 4, 600],
    [140, 4, 750],
    [160, 5, 800],
    [180, 5, 950]
])

# Pisahkan features dan target
X = data[:, :2]  # [luas_tanah, jumlah_kamar]
y = data[:, 2]   # harga

# Fungsi normalisasi Min-Max
def min_max_normalize(X):
    """Normalisasi data ke range [0,1]"""
    X_min = np.min(X, axis=0)
    X_max = np.max(X, axis=0)
    X_range = X_max - X_min
    X_normalized = (X - X_min) / X_range
    return X_normalized, X_min, X_max, X_range

def min_max_denormalize(X_normalized, X_min, X_max):
    """Denormalisasi data dari range [0,1] ke skala asli"""
    X_range = X_max - X_min
    return X_normalized * X_range + X_min

# Fungsi prediksi
def predict(X, w, b):
    """Prediksi: y = X @ w + b"""
    return np.dot(X, w) + b

# Fungsi training SGD
def train_sgd(X, y, learning_rate=0.1, epochs=1000):
    """Training model dengan Stochastic Gradient Descent"""

    # Inisialisasi weights dan bias secara random
    w = np.random.randn(X.shape[1])
    b = np.random.randn()

    losses = []

    print("Sebelum training:")
    print(f"Weights: {w}, Bias: {b:.4f}")

    for epoch in range(epochs):
```

```
        total_loss = 0

        # SGD: update untuk setiap data point
        for i in range(len(X)):

            # Prediksi untuk satu data
            prediction = predict(X[i], w, b)
            error = prediction - y[i]

            # Gradients
            grad_w = error * X[i]  # error * x
            grad_b = error         # error * 1

            # Update weights & bias
            w -= learning_rate * grad_w
            b -= learning_rate * grad_b

            # Hitung loss untuk monitoring
            total_loss += error**2

        # Rata-rata loss per epoch
        avg_loss = total_loss / len(X)
        losses.append(avg_loss)

        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Loss = {avg_loss:.4f}")

    print("\nSetelah training:")
    print(f"Weights: {w}, Bias: {b:.4f}")

    return w, b, losses

# Fungsi evaluasi
def evaluate_model(X, y, w, b):
    """Evaluasi model"""
    predictions = predict(X, w, b)
    mse = np.mean((predictions - y)**2)
    r2 = 1 - np.sum((y - predictions)**2) / np.sum((y - np.mean(y))**2)

    print(f"Mean Squared Error: {mse:.4f}")
    print(f"R-squared (R²): {r2:.4f}")

    # Tampilkan prediksi vs actual
    print("\nPrediksi vs Aktual:")
    for i in range(len(X)):
```

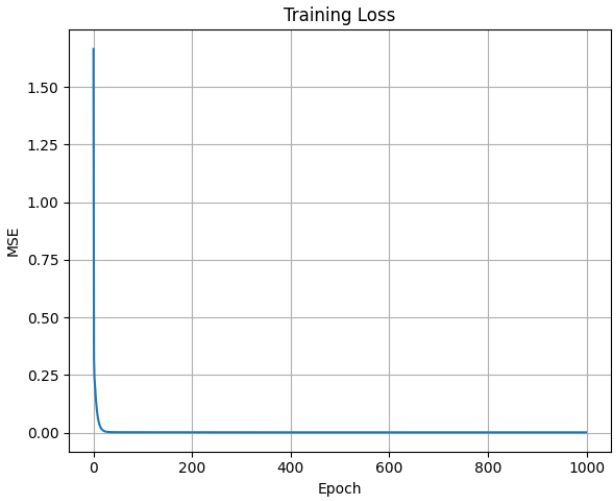
```
        print(f"Data {i+1}: Predicted = {predictions[i]:.3f}, Actual =
{y[i]:.3f}, Error = {predictions[i]-y[i]:.3f}")

    return predictions, mse, r2

# Fungsi prediksi rumah baru
def predict_new_house(luas, kamar, w, b, X_min, X_max, X_range, y_min, y_max,
y_range):
    """Prediksi harga rumah baru dengan normalisasi Min-Max"""
    # Normalisasi input
    luas_norm = (luas - X_min[0]) / X_range[0]
    kamar_norm = (kamar - X_min[1]) / X_range[1]
    X_new_normalized = np.array([luas_norm, kamar_norm])

    # Prediksi dalam skala normalisasi
    y_pred_normalized = predict(X_new_normalized, w, b)

    # Denormalize ke skala asli
    y_pred = y_pred_normalized * y_range + y_min

    return y_pred

# ========== MAIN PROGRAM ==========

print("=== NORMALISASI MIN-MAX ===")

# Normalisasi fitur X
X_normalized, X_min, X_max, X_range = min_max_normalize(X)
print(f"X min: {X_min}")
print(f"X max: {X_max}")
print(f"X range: {X_range}")

# Normalisasi target y
y_normalized, y_min, y_max, y_range = min_max_normalize(y.reshape(-1, 1))
y_normalized = y_normalized.flatten()
y_min = y_min[0]
y_max = y_max[0]
y_range = y_range[0]

print(f"y min: {y_min}")
print(f"y max: {y_max}")
print(f"y range: {y_range}")

print("\nData setelah normalisasi:")
for i in range(len(X_normalized)):
```

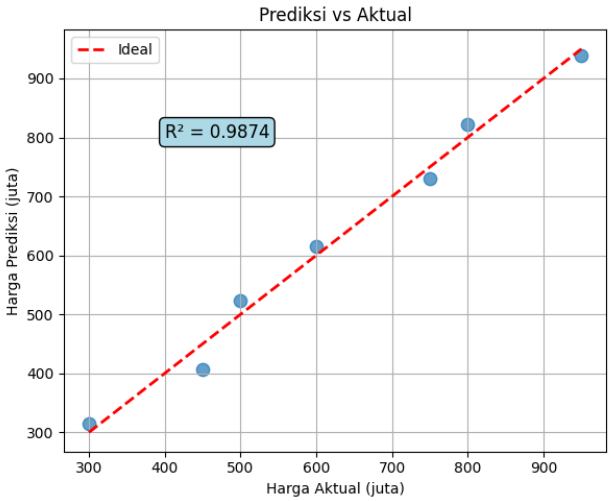
```
    print(f"Data {i+1}: X = {X_normalized[i]}, y = {y_normalized[i]:.4f}")

# Training model
print("\n=== TRAINING MODEL ===")
w, b, losses = train_sgd(X_normalized, y_normalized,
                         learning_rate=0.1, epochs=1000)

# Evaluasi model dengan data ternormalisasi
print("\n=== EVALUASI (DATA NORMALISASI) ===")
predictions_normalized, mse, r2 = evaluate_model(X_normalized, y_normalized, w,
b)

# Interpretasi koefisien dalam skala asli
print("\n=== INTERPRETASI MODEL (SKALA ASLI) ===")

# Konversi koefisien ke skala asli
w1_denorm = w[0] * y_range / X_range[0]
w2_denorm = w[1] * y_range / X_range[1]
b_denorm = b * y_range + y_min

print(f"Koefisien luas tanah: {w1_denorm:.1f} juta/m²")
print(f"Koefisien jumlah kamar: {w2_denorm:.1f} juta/kamar")
print(f"Bias (harga dasar): {b_denorm:.1f} juta")

print(f"\nRumus prediksi (skala asli):")
print(f"Harga = {w1_denorm:.1f} × luas + {w2_denorm:.1f} × kamar +
{b_denorm:.1f}")

# Contoh prediksi rumah baru
print("\n=== PREDIKSI RUMAH BARU ===")
rumah_baru = [
    (90, 3),   # luas 90m², 3 kamar
    (150, 4),  # luas 150m², 4 kamar
    (200, 5)   # luas 200m², 5 kamar
]

for luas, kamar in rumah_baru:
    harga_pred = predict_new_house(luas, kamar, w, b, X_min, X_max, X_range,
y_min, y_max, y_range)
    print(f"Rumah {luas}m², {kamar} kamar → Prediksi: {harga_pred:.0f} juta")

# Validasi dengan rumus langsung
print("\n=== VALIDASI DENGAN RUMUS LANGSUNG ===")
for i in range(len(X)):
    luas = X[i, 0]
```

```
    kamar = X[i, 1]
    harga_pred = w1_denorm * luas + w2_denorm * kamar + b_denorm
    harga_aktual = y[i]
    error = harga_pred - harga_aktual
    print(f"Data {i+1}: Prediksi = {harga_pred:.0f}, Aktual = {harga_aktual},
Error = {error:.0f}")

# Visualisasi hasil
plt.figure(figsize=(12, 5))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.grid(True)

# Plot prediksi vs aktual
plt.subplot(1, 2, 2)
predictions_actual = predictions_normalized * y_range + y_min

plt.scatter(y, predictions_actual, alpha=0.7, s=80)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2, label='Ideal')
plt.xlabel('Harga Aktual (juta)')
plt.ylabel('Harga Prediksi (juta)')
plt.title('Prediksi vs Aktual')
plt.legend()
plt.grid(True)

# Tambahkan informasi R² pada plot
plt.text(400, 800, f'R² = {r2:.4f}', fontsize=12,
         bbox=dict(boxstyle="round,pad=0.3", facecolor="lightblue"))

plt.tight_layout()
plt.show()

# Analisis residual
print("\n=== ANALISIS RESIDUAL ===")
residuals = predictions_actual - y
print(f"Mean Residual: {np.mean(residuals):.2f}")
print(f"Std Residual: {np.std(residuals):.2f}")
print(f"Max Residual: {np.max(residuals):.2f}")
print(f"Min Residual: {np.min(residuals):.2f}")
```

```python
# Prediksi untuk range luas tanah
print("\n=== PENGARUH LUAS TANAH ===")
luas_range = np.arange(60, 200, 20)
for luas in luas_range:
    # Asumsi 3 kamar
    harga_pred = predict_new_house(luas, 3, w, b, X_min, X_max, X_range, y_min,
y_max, y_range)
    print(f"Luas {luas}m², 3 kamar → Prediksi: {harga_pred:.0f} juta")
```

```
==================================
=== NORMALISASI MIN-MAX ===
X min: [60  2]
X max: [180   5]
X range: [120   3]
y min: 300
y max: 950
y range: 650

Data setelah normalisasi:
Data 1: X = [0. 0.], y = 0.0000
Data 2: X = [0.16666667 0.33333333], y = 0.2308
Data 3: X = [0.33333333 0.33333333], y = 0.3077
Data 4: X = [0.5        0.66666667], y = 0.4615
Data 5: X = [0.66666667 0.66666667], y = 0.6923
Data 6: X = [0.83333333 1.        ], y = 0.7692
Data 7: X = [1. 1.], y = 1.0000

=== TRAINING MODEL ===
Sebelum training:
Weights: [-0.04563284 -1.9007882 ], Bias: -0.0244
Epoch 0: Loss = 1.6643
Epoch 100: Loss = 0.0020
Epoch 200: Loss = 0.0018
Epoch 300: Loss = 0.0017
Epoch 400: Loss = 0.0016
...
Data 4: Prediksi = 914, Aktual = 600, Error = 314
Data 5: Prediksi = 1031, Aktual = 750, Error = 281
Data 6: Prediksi = 1122, Aktual = 800, Error = 322
Data 7: Prediksi = 1239, Aktual = 950, Error = 289
```






```
=== ANALISIS RESIDUAL ===
Mean Residual: 0.18
Std Residual: 23.44
Max Residual: 22.85
Min Residual: -43.67

=== PENGARUH LUAS TANAH ===
Luas 60m², 3 kamar → Prediksi: 290 juta
Luas 80m², 3 kamar → Prediksi: 406 juta
Luas 100m², 3 kamar → Prediksi: 523 juta
Luas 120m², 3 kamar → Prediksi: 639 juta
Luas 140m², 3 kamar → Prediksi: 756 juta
Luas 160m², 3 kamar → Prediksi: 872 juta
Luas 180m², 3 kamar → Prediksi: 989 juta
```

```
# Sebagai benchmark
X_with_bias = np.column_stack([X, np.ones(len(X))])
w_analytical = np.linalg.inv(X_with_bias.T @ X_with_bias) @ X_with_bias.T @ y

print("\n=== PERBANDINGAN DENGAN SOLUSI ANALITIK ===")
print(f"SGD Weights: {w_denormalized[0]:.0f}, {w_denormalized[1]:.0f}, Bias:
{b_denormalized:.0f}")
print(f"Analytical:  {w_analytical[0]:.0f}, {w_analytical[1]:.0f}, Bias:
{w_analytical[2]:.0f}")
```

```
=== PERBANDINGAN DENGAN SOLUSI ANALITIK ===
SGD Weights: 4, 51, Bias: -37
Analytical:  6, -17, Bias: 12
```

COPY PASTE SIMULASI SGD

```
# SIMULASI SGD
import numpy as np
import tkinter as tk
from tkinter import ttk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import time

# ==========================
# SGD Animation (Tkinter)
# ==========================
# This script trains a simple linear model (using SGD) on the provided dataset
# and animates the model updates inside a Tkinter window. For visualization we
# plot Harga (juta) vs Luas (m²) and draw the predicted line using the current
# model parameters. To keep the visualization meaningful while training on two
# features (luas, kamar) we fix "kamar" as the dataset median when plotting the line.
#
# To run locally: python sgd_tkinter_animation.py
# (Requires matplotlib)

# Dataset: [luas_tanah(m²), jumlah_kamar, harga(juta)]
data = np.array([
    [60, 2, 300],
    [80, 3, 450],
    [100, 3, 500],
    [120, 4, 600],
    [140, 4, 750],
    [160, 5, 800],
    [180, 5, 950]
])

X = data[:, :2].astype(float)
y = data[:, 2].astype(float)

# --- Min-Max normalize helpers ---
def min_max_normalize(data):
    min_val = np.min(data, axis=0)
    max_val = np.max(data, axis=0)
    norm = (data - min_val) / (max_val - min_val)
    return norm, min_val, max_val

def min_max_denormalize(norm_data, min_val, max_val):
    return norm_data * (max_val - min_val) + min_val

X_norm, X_min, X_max = min_max_normalize(X)
y_norm, y_min, y_max = min_max_normalize(y.reshape(-1,1))
```

```
y_norm = y_norm.flatten()

# Initialize weights and bias
np.random.seed(42)
w = np.random.randn(2)
b = np.random.randn()

# SGD hyperparams
learning_rate = 0.1
epochs = 20  # small number for demo; animation shows per-sample updates

# Prepare plotting grid (for line)
luas_grid = np.linspace(X[:,0].min(), X[:,0].max(), 200)  # land area in original scale
median_kamar = np.median(X[:,1])

# --- prediction helpers using normalized model ---
def predict_norm(x_norm, w, b):
    return np.dot(x_norm, w) + b

def predict_denorm_from_luas(luas_val, kamar_val, w, b):
    # normalize input
    luas_n = (luas_val - X_min[0]) / (X_max[0] - X_min[0])
    kamar_n = (kamar_val - X_min[1]) / (X_max[1] - X_min[1])
    pred_n = predict_norm(np.array([luas_n, kamar_n]), w, b)
    pred = min_max_denormalize(pred_n, y_min, y_max)[0]
    return pred

# --- Tkinter + Matplotlib setup ---
root = tk.Tk()
root.title("Animasi SGD - Linear Regression (Visualisasi)")
root.geometry("900x600")

mainframe = ttk.Frame(root, padding="6 6 6 6")
mainframe.pack(fill=tk.BOTH, expand=True)

# Matplotlib figure
fig, ax = plt.subplots(figsize=(7,5))
canvas = FigureCanvasTkAgg(fig, master=mainframe)
canvas_widget = canvas.get_tk_widget()
canvas_widget.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Right-side controls & info
control_frame = ttk.Frame(mainframe, width=250)
control_frame.pack(side=tk.RIGHT, fill=tk.Y)
status_label = ttk.Label(control_frame, text="Inisialisasi...", justify=tk.LEFT)
status_label.pack(padx=8, pady=8)
```

```
info_text = tk.Text(control_frame, width=30, height=15, wrap=tk.WORD)
info_text.pack(padx=8, pady=8)
info_text.insert(tk.END, "Informasi training akan muncul di sini.\n")

# Plot initial scatter (actual data in original scale)
ax.clear()
ax.set_title("Harga vs Luas - Animasi SGD")
ax.set_xlabel("Luas tanah (m²)")
ax.set_ylabel("Harga (juta)")
ax.scatter(X[:,0], y, label="Data (aktual)", s=50)
line_plot, = ax.plot([], [], 'r-', linewidth=2, label="Prediksi (current)")
ax.legend()
ax.grid(True)

# Force canvas draw once
canvas.draw()

# Training iterator state (we'll step manually to animate)
training_steps = []  # each step stores (w_copy, b_copy, loss, i, epoch)
# We'll prepare steps by actually performing SGD but collecting snapshots after each sample update.
w_copy = w.copy()
b_copy = float(b)

for epoch in range(epochs):
    # optionally shuffle indices for SGD
    indices = np.arange(len(X_norm))
    # np.random.shuffle(indices)  # optional: uncomment to randomize order per epoch
    for i in indices:
        x_i = X_norm[i]
        y_i = y_norm[i]
        pred = predict_norm(x_i, w_copy, b_copy)
        error = pred - y_i
        grad_w = error * x_i
        grad_b = error
        # update
        w_copy = w_copy - learning_rate * grad_w
        b_copy = b_copy - learning_rate * grad_b
        loss = (error**2)
        # store snapshot
        training_steps.append((w_copy.copy(), float(b_copy), float(loss), int(i), int(epoch)))

# Animation control variables
step_index = 0
max_steps = len(training_steps)
```

```
playing = False
delay_ms = 25  # ms between frames (fast); adjust to slow down

# Functions to update UI and plot
def update_plot_for_step(step_idx):
    global line_plot
    w_step, b_step, loss_step, i_step, epoch_step = training_steps[step_idx]
    # compute prediction line (use median kamar for line)
    y_line = [predict_denorm_from_luas(l, median_kamar, w_step, b_step) for l in luas_grid]
    line_plot.set_data(luas_grid, y_line)
    # update status box
    status = f"Step: {step_idx+1}/{max_steps}\nEpoch: {epoch_step}, Sample idx:
{i_step}\nLoss (sample): {loss_step:.6f}\nWeights: [{w_step[0]:.4f}, {w_step[1]:.4f}]\nBias:
{b_step:.4f}"
    status_label.config(text=status)
    # update info text (append first few, keep short)
    if step_idx % 30 == 0 or step_idx == max_steps-1:
        info_text.insert(tk.END, f"Step {step_idx+1}: loss={loss_step:.6f}\\n")
        info_text.see(tk.END)
    canvas.draw_idle()

def play_animation():
    global playing, step_index
    playing = True
    _play_step()

def _play_step():
    global step_index, playing
    if not playing:
        return
    if step_index < max_steps:
        update_plot_for_step(step_index)
        step_index += 1
        root.after(delay_ms, _play_step)
    else:
        playing = False

def pause_animation():
    global playing
    playing = False

def reset_animation():
    global step_index, playing
    playing = False
    step_index = 0
    update_plot_for_step(0)
```

```
def step_forward():
    global step_index
    if step_index < max_steps-1:
        step_index += 1
        update_plot_for_step(step_index)

def step_backward():
    global step_index
    if step_index > 0:
        step_index -= 1
        update_plot_for_step(step_index)

# Controls
btn_play = ttk.Button(control_frame, text="Play ▶", command=play_animation)
btn_pause = ttk.Button(control_frame, text="Pause ▮▮", command=pause_animation)
btn_reset = ttk.Button(control_frame, text="Reset ↺", command=reset_animation)
btn_step_f = ttk.Button(control_frame, text="Step →", command=step_forward)
btn_step_b = ttk.Button(control_frame, text="← Step", command=step_backward)

btn_play.pack(fill=tk.X, padx=6, pady=4)
btn_pause.pack(fill=tk.X, padx=6, pady=4)
btn_reset.pack(fill=tk.X, padx=6, pady=4)
btn_step_f.pack(fill=tk.X, padx=6, pady=4)
btn_step_b.pack(fill=tk.X, padx=6, pady=4)

# Initialize with first step
if max_steps > 0:
    update_plot_for_step(0)

# Run Tkinter mainloop
root.mainloop()
```

6. BATCH GRADIENT DECENT

1. Konsep Matematis BGD vs SGD

SGD (Yang Sudah Anda Pahami)

```python
# Update untuk SETIAP data point
for i in range(len(X)):
    prediction = model(X[i])
    error = prediction - y[i]

    # Update weights berdasarkan SATU data point
    w -= learning_rate * (error * X[i])
    b -= learning_rate * error
```

BGD (Batch Gradient Descent)
python
```python
# Hitung RATA-RATA gradient dari SEMUA data
total_grad_w = 0
total_grad_b = 0

for i in range(len(X)):
    prediction = model(X[i])
    error = prediction - y[i]

    # Akumulasi gradient
    total_grad_w += error * X[i]
    total_grad_b += error

# Update weights berdasarkan RATA-RATA semua data
w -= learning_rate * (total_grad_w / len(X))
b -= learning_rate * (total_grad_b / len(X))
```

2. Formulasi Matematis BGD

Fungsi Loss (MSE):

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m} \quad (h_w(x^{(i)}) - y^{(i)})^2$$

Gradients:

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} \quad \left(h_w\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} \quad \left(h_w\left(x^{(i)}\right) - y^{(i)}\right)$$

Update Rules:

$$w := w - \alpha \cdot \frac{\partial J}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial J}{\partial b}$$

3. Implementasi BGD untuk Prediksi Rumah

```python
#BGD
# ================================================================
# PREDIKSI HARGA RUMAH DENGAN BATCH GRADIENT DESCENT (BGD)
# ================================================================

import numpy as np
import matplotlib.pyplot as plt

# ----------------------------
# 1. Dataset (luas, kamar, harga)
```

```python
# ----------------------------
data = np.array([
    [60, 2, 300],
    [80, 3, 450],
    [100, 3, 500],
    [120, 4, 600],
    [140, 4, 750],
    [160, 5, 800],
    [180, 5, 950]
])

# Pisahkan fitur (X) dan target (y)
X = data[:, :2]   # fitur = [luas, kamar]
y = data[:, 2]    # target = harga (dalam juta)

# ----------------------------
# 2. Normalisasi data (Min-Max)
# ----------------------------
def min_max_normalize(data):
    """Mengubah nilai data ke rentang [0,1] agar stabil saat training."""
    min_vals = np.min(data, axis=0)
    max_vals = np.max(data, axis=0)
    normalized = (data - min_vals) / (max_vals - min_vals)
    return normalized, min_vals, max_vals

# Normalisasi X dan y
X_norm, X_min, X_max = min_max_normalize(X)
y_norm, y_min, y_max = min_max_normalize(y.reshape(-1, 1))
y_norm = y_norm.flatten()

# ----------------------------
# 3. Inisialisasi parameter awal
# ----------------------------
np.random.seed(42)     # agar hasil acak konsisten
w = np.random.randn(2) # dua bobot: w1(luas), w2(kamar)
b = np.random.randn()  # bias (intercept)

print(f"Weights awal: {w}, Bias awal: {b:.4f}")

# ----------------------------
# 4. Fungsi model dan loss
# ----------------------------
def predict(X, w, b):
    """Prediksi harga dengan model linear sederhana"""
    return np.dot(X, w) + b  # w1*x1 + w2*x2 + b
```

```python
def calculate_loss(y_true, y_pred):
    """Menghitung rata-rata error kuadrat (MSE)"""
    return np.mean((y_true - y_pred)**2)

# ----------------------------
# 5. Fungsi Training (Batch GD)
# ----------------------------
def train_batch_gradient_descent(X, y, w, b, lr=0.1, epochs=1000):
    """
    Melatih model menggunakan Batch Gradient Descent:
    - Menggunakan seluruh data untuk menghitung gradien
    - Update dilakukan sekali per epoch
    """
    losses = []
    n = len(X)

    for epoch in range(epochs):
        # --- Feedforward ---
        y_pred = predict(X, w, b)

        # --- Error dan gradien ---
        error = y_pred - y
        grad_w = (1/n) * np.dot(X.T, error)  # turunan terhadap w
        grad_b = (1/n) * np.sum(error)       # turunan terhadap b

        # --- Update parameter ---
        w -= lr * grad_w
        b -= lr * grad_b

        # --- Hitung loss untuk monitoring ---
        loss = calculate_loss(y, y_pred)
        losses.append(loss)

        # Cetak setiap 100 epoch
        if epoch % 100 == 0:
            print(f"Epoch {epoch:4d} | Loss={loss:.6f}")

    return w, b, losses

# ----------------------------
# 6. Jalankan training
# ----------------------------
print("\n=== TRAINING DIMULAI ===")
w_trained, b_trained, losses = train_batch_gradient_descent(
```

```
    X_norm, y_norm, w, b, lr=1.0, epochs=1000
)

print("\n=== TRAINING SELESAI ===")
print(f"Weights akhir: {w_trained}")
print(f"Bias akhir: {b_trained:.6f}")

# ------------------------------
# 7. Denormalisasi hasil agar bisa diinterpretasikan
# ------------------------------
def denormalize_coefficients(w, b, X_min, X_max, y_min, y_max):
    """Kembalikan koefisien ke skala asli (juta rupiah)"""
    w1 = w[0] * (y_max - y_min) / (X_max[0] - X_min[0])
    w2 = w[1] * (y_max - y_min) / (X_max[1] - X_min[1])
    b_ = y_min + (y_max - y_min)*b - w1*X_min[0] - w2*X_min[1]
    return w1[0], w2[0], b_[0]

w1_real, w2_real, b_real = denormalize_coefficients(
    w_trained, b_trained, X_min, X_max, y_min, y_max
)

print(f"\n=== INTERPRETASI MODEL (SKALA ASLI) ===")
print(f"Setiap +1 m² menambah harga: {w1_real:.1f} juta")
print(f"Setiap +1 kamar menambah harga: {w2_real:.1f} juta")
print(f"Harga dasar (bias): {b_real:.1f} juta")

# Rumus akhir:
print(f"\nHarga = {w1_real:.1f} * luas + {w2_real:.1f} * kamar + {b_real:.1f}")

# ------------------------------
# 8. Prediksi rumah baru
# ------------------------------
def predict_new(luas, kamar, w, b, X_min, X_max, y_min, y_max):
    """Prediksi harga rumah baru (input skala asli)"""
    luas_n = (luas - X_min[0]) / (X_max[0] - X_min[0])
    kamar_n = (kamar - X_min[1]) / (X_max[1] - X_min[1])
    y_pred_n = predict(np.array([luas_n, kamar_n]), w, b)
    return y_pred_n * (y_max - y_min) + y_min

print("\n=== PREDIKSI RUMAH BARU ===")
rumah_baru = [(90, 3), (150, 4), (200, 5)]
for luas, kamar in rumah_baru:
    harga = predict_new(luas, kamar, w_trained, b_trained,
                        X_min, X_max, y_min, y_max)
    print(f"Rumah {luas}m², {kamar} kamar → Prediksi: {harga[0]:.0f} juta")
```

```python
# ----------------------------
# 9. Visualisasi hasil training
# ----------------------------
plt.figure(figsize=(10,4))

# Grafik konvergensi loss
plt.subplot(1, 2, 1)
plt.plot(losses)
plt.title("Konvergensi Batch Gradient Descent")
plt.xlabel("Epoch")
plt.ylabel("Loss (MSE)")
plt.grid(True)

# Grafik prediksi vs aktual (denormalisasi)
y_pred_norm = predict(X_norm, w_trained, b_trained)
y_pred = y_pred_norm * (y_max - y_min) + y_min
plt.subplot(1, 2, 2)
plt.scatter(y, y_pred, color='blue', alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.title("Prediksi vs Aktual (Skala Asli)")
plt.xlabel("Harga Aktual (juta)")
plt.ylabel("Harga Prediksi (juta)")
plt.grid(True)

plt.tight_layout()
plt.show()

# ----------------------------
# 10. Kesimpulan ringkas
# ----------------------------
print("\n=== KESIMPULAN ===")
print("✅ Batch Gradient Descent memperbarui bobot menggunakan seluruh data per langkah.")
print("✅ Konvergensi stabil, cocok untuk dataset kecil-menengah.")
print("✅ Hasil mendekati solusi analitik regresi linear.")
```

Weights awal: [0.49671415 -0.1382643], Bias awal: 0.6477

=== TRAINING DIMULAI ===
Epoch 0 | Loss=0.145020
Epoch 100 | Loss=0.001460
Epoch 200 | Loss=0.001376
Epoch 300 | Loss=0.001333
Epoch 400 | Loss=0.001311

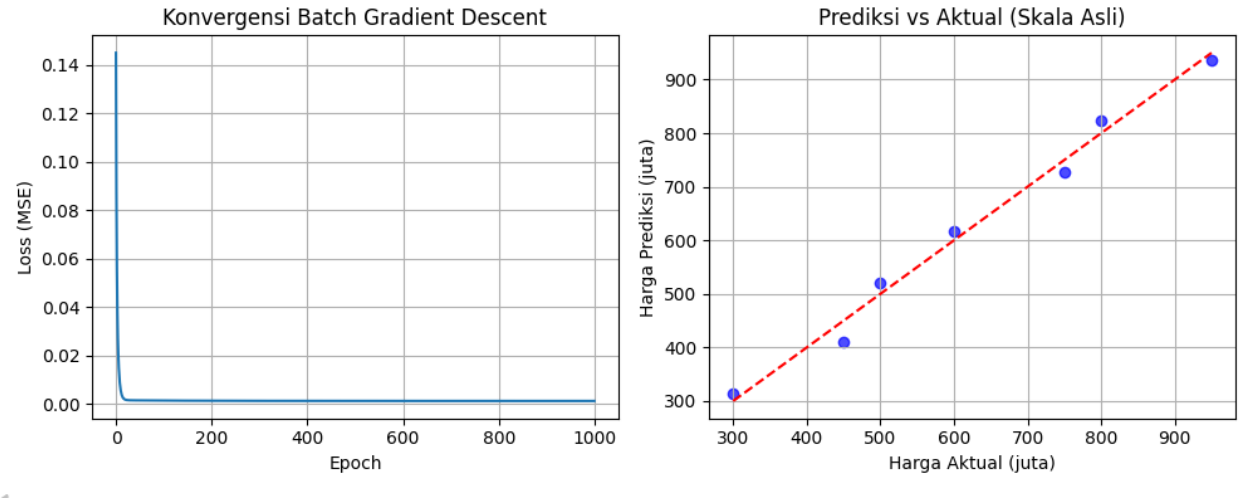
Epoch 500 | Loss=0.001300
Epoch 600 | Loss=0.001294
Epoch 700 | Loss=0.001291
Epoch 800 | Loss=0.001290
Epoch 900 | Loss=0.001289

=== TRAINING SELESAI ===
Weights akhir: [1.02498722 -0.06917685]
Bias akhir: 0.021540

=== INTERPRETASI MODEL (SKALA ASLI) ===
Setiap +1 m² menambah harga: 5.6 juta
Setiap +1 kamar menambah harga: -15.0 juta
Harga dasar (bias): 10.9 juta

Harga = 5.6 * luas + -15.0 * kamar + 10.9

=== PREDIKSI RUMAH BARU ===
Rumah 90m², 3 kamar → Prediksi: 466 juta
Rumah 150m², 4 kamar → Prediksi: 784 juta
Rumah 200m², 5 kamar → Prediksi: 1046 juta



6. Kelebihan dan Kekurangan BGD

Kelebihan (Advantages)	**Kekurangan (Disadvantages)**
✓ **Konvergensi yang stabil** - Perubahan parameter lebih smooth dan konsisten	✗ **Lambat untuk dataset besar** - Harus memproses seluruh data setiap iterasi

Kelebihan (Advantages)	Kekurangan (Disadvantages)
✓ **Konvergensi terjamin** (pada fungsi convex) - Secara teoritis akan konvergen ke minimum global	✗ **Intensif memori** - Harus menyimpan seluruh dataset dalam memori
✓ **Update yang stabil** - Arah gradien lebih akurat karena menggunakan semua data	✗ **Rentan terjebak di local minima** - Tidak memiliki noise yang membantu escape local minima
✓ **Mudah diimplementasi secara paralel** - Dapat memanfaatkan komputasi paralel	✗ **Tidak cocok untuk data streaming** - Tidak bisa update model secara real-time
✓ **Konvergensi ke minimum global** (pada fungsi convex) - Lebih reliable untuk fungsi smooth	✗ **Komputasi mahal per iterasi** - Waktu komputasi meningkat seiring ukuran dataset

Karakteristik Utama BGD:

- **Menggunakan**: Seluruh dataset untuk menghitung gradien
- **Kecepatan Konvergensi**: Lambat per iterasi, tapi mungkin lebih sedikit iterasi total
- **Stabilitas**: Sangat stabil
- **Memory Usage**: Tinggi
- **Best Use Case**: Dataset kecil hingga medium, fungsi convex

Kesimpulan Matematis:

1. **BGD** = Rata-rata gradient dari **semua** data point
2. **SGD** = Gradient dari **satu** data point
3. **Convergence**: BGD lebih smooth, SGD lebih noisy
4. **Komputasi**: BGD lebih berat per epoch, tapi butuh lebih sedikit epoch

Next Step: Mini-batch GD (kompromi antara BGD dan SGD)! 🎯

7. MINI BATCH GRADIENT DECENT

1. Konsep Mini-batch GD

Visualisasi Perbandingan:

```
text
BGD:      [████████████████████]  ← Pakai SEMUA data
Mini-batch: [███] [███] [███] [███]  ← Pakai BEBERAPA subset data
SGD:      [█] [█] [█] [█] [█] [█]  ← Pakai SATU data per update
```

2. Formulasi Matematis Mini-batch GD

Untuk setiap batch:

$$\text{Batch Size} = B(\text{biasanya } 32, 64, 128, ...)$$

$$\frac{\partial J}{\partial w} = \frac{1}{B} \sum_{i=b}^{b+B-1} \left(h_w\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$

$$w := w - \alpha \cdot \frac{\partial J}{\partial w}$$

🎯 Apa itu Mini-batch GD? - Analogi Belajar

Bayangkan kita ingin **belajar untuk ujian matematika**:

3 Cara Belajar:

1. **BGD (Full Batch)** = Baca SEMUA buku sekaligus → **terlalu berat**
2. **SGD (Stochastic)** = Baca 1 halaman acak → **terlalu berisik**
3. **Mini-batch** = Baca 1 bab (beberapa halaman) → **paling efisien!**

📊 Contoh Nyata: Prediksi Harga Rumah

Kita punya data rumah:

text

```
Rumah 1: Luas=50m², Kamar=2 → Harga=500 juta
Rumah 2: Luas=70m², Kamar=3 → Harga=700 juta
...
```

Cara Kerja Mini-batch:

python

```
# Contoh sederhana - BAGI DATA MENJADI KELOMPOK-KELOMPOK KECIL
data_rumah = [rumah1, rumah2, rumah3, ..., rumah100]

# Bagi jadi batch-batch kecil (misal batch_size=10)
batch_1 = [rumah1-rumah10]  # ← Hitung gradien dari 10 rumah ini
batch_2 = [rumah11-rumah20]  # ← Lalu dari 10 rumah ini
batch_3 = [rumah21-rumah30]  # ← Dan seterusnya...
```

🚀 Implementasi Simple yang Mudah Dipahami

```python
#Mini Batch Gradient Decent
import numpy as np
import matplotlib.pyplot as plt

# ========================================
# [1] Dataset Awal
# ========================================
# Format data: [luas_tanah(m²), jumlah_kamar, harga(juta)]
data = np.array([
    [60, 2, 300],
    [80, 3, 450],
    [100, 3, 500],
    [120, 4, 600],
```

```
    [140, 4, 750],
    [160, 5, 800],
    [180, 5, 950]
])

# Pisahkan fitur (X) dan target (y)
X = data[:, :2]   # kolom 0-1 → luas, kamar
y = data[:, 2]    # kolom 2 → harga

# ========================================
# [2] Normalisasi Data (Min-Max)
# ========================================
def min_max_normalize(data):
    """Normalisasi ke rentang [0,1] agar training stabil"""
    return (data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data,
axis=0))

X_norm = min_max_normalize(X)
y_norm = min_max_normalize(y.reshape(-1, 1)).flatten()

# ========================================
# [3] Inisialisasi Parameter Awal
# ========================================
np.random.seed(42)
w = np.random.randn(2)   # bobot untuk [luas, kamar]
b = np.random.randn()    # bias

# ========================================
# [4] Fungsi Prediksi & Loss
# ========================================
def predict(X, w, b):
    """Prediksi linear: y_pred = w1*x1 + w2*x2 + b"""
    return np.dot(X, w) + b

def mse_loss(y_true, y_pred):
    """Hitung Mean Squared Error (MSE)"""
    return np.mean((y_true - y_pred)**2)

# ========================================
# [5] Mini-Batch Gradient Descent
# ========================================
def train_mini_batch(X, y, w, b, lr=0.5, epochs=500, batch_size=2):
    """Training model dengan Mini-Batch Gradient Descent"""
    n = len(X)
    losses = []
```

```
    for epoch in range(epochs):
        # Acak urutan data agar batch berbeda setiap epoch
        indices = np.random.permutation(n)
        X, y = X[indices], y[indices]

        for i in range(0, n, batch_size):
            # Ambil subset data (mini-batch)
            X_batch = X[i:i+batch_size]
            y_batch = y[i:i+batch_size]

            # Feedforward
            y_pred = predict(X_batch, w, b)
            error = y_pred - y_batch

            # Hitung turunan (gradien)
            grad_w = (1/len(X_batch)) * np.dot(X_batch.T, error)
            grad_b = (1/len(X_batch)) * np.sum(error)

            # Update parameter
            w -= lr * grad_w
            b -= lr * grad_b

        # Simpan loss tiap epoch
        loss = mse_loss(y, predict(X, w, b))
        losses.append(loss)

        # Tampilkan progress setiap 100 epoch
        if epoch % 100 == 0:
            print(f"Epoch {epoch:3d} → Loss: {loss:.6f}")

    return w, b, losses

# ========================================
# 6 Jalankan Training
# ========================================
print("=== Training dengan Mini-Batch GD ===")
w_trained, b_trained, losses = train_mini_batch(X_norm, y_norm, w, b)

# ========================================
# 7 Visualisasi Konvergensi
# ========================================
plt.plot(losses)
plt.title("Konvergensi Loss (Mini-Batch GD)")
plt.xlabel("Epoch")
```

```
plt.ylabel("MSE Loss")
plt.grid(True)
plt.show()

# Grafik prediksi vs aktual (denormalisasi)
y_pred_norm = predict(X_norm, w_trained, b_trained)
y_pred = y_pred_norm * (y_max - y_min) + y_min
plt.scatter(y, y_pred, color='blue', alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.title("Prediksi vs Aktual (Skala Asli)")
plt.xlabel("Harga Aktual (juta)")
plt.ylabel("Harga Prediksi (juta)")
plt.grid(True)
plt.show()



# ========================================
# 8 Evaluasi Model
# ========================================
# Prediksi hasil akhir (dalam skala normalisasi)
y_pred_norm = predict(X_norm, w_trained, b_trained)

# Hitung MSE
mse = mse_loss(y_norm, y_pred_norm)
print(f"\nLoss Akhir (MSE): {mse:.6f}")

# ========================================
# 9 Prediksi Rumah Baru
# ========================================
def predict_new(luas, kamar, w, b, X):
    """Prediksi harga rumah baru berdasarkan model terlatih"""
    # Normalisasi input baru sesuai skala X
    luas_norm = (luas - X[:,0].min()) / (X[:,0].max() - X[:,0].min())
    kamar_norm = (kamar - X[:,1].min()) / (X[:,1].max() - X[:,1].min())

    # Prediksi (masih dalam skala normalisasi y)
    y_pred_norm = predict(np.array([luas_norm, kamar_norm]), w, b)

    # Denormalisasi hasil ke skala harga asli
    y_min, y_max = y.min(), y.max()
    harga_pred = y_pred_norm * (y_max - y_min) + y_min

    return harga_pred
```

```python
# Contoh prediksi baru
print("\n=== Prediksi Rumah Baru ===")
for luas, kamar in [(90,3), (150,4), (200,5)]:
    harga = predict_new(luas, kamar, w_trained, b_trained, X)
    print(f"Luas {luas}m², {kamar} kamar → Prediksi harga: {harga:.0f} juta")
```

...

🎯 Perbandingan 3 Metode dalam Tabel

Metode	Cara Kerja	Analoginya	Kelebihan	Kekurangan
BGD	Pakai SEMUA data sekaligus	Baca semua buku sekaligus	Stabil, akurat	Lambat, butuh memory besar
SGD	Pakai 1 data acak	Baca 1 halaman acak	Cepat, bisa escape local minima	Berisik, tidak stabil
Mini-batch	Pakai kelompok kecil (2-256 data)	Baca 1 bab (beberapa halaman)	**Seimbang: cepat + stabil**	Butuh tuning batch size

⚡ Kelebihan Mini-batch dalam Praktek:

1. 🔄 **Lebih Cepat**: Tidak perlu tunggu semua data
2. 🎯 **Lebih Stabil**: Tidak terlalu berisik seperti SGD
3. 💪 **GPU Friendly**: Bisa pakai paralel processing
4. 🚀 **Cocok untuk data besar**: Tidak overload memory

🔧 Tips Memilih Batch Size:

```python
# Rekomendasi praktis:
if dataset_kecil:    batch_size = 2-16
if dataset_sedang:   batch_size = 32-64
if dataset_besar:    batch_size = 128-256
if pakai_GPU:        batch_size = power_of_2  # 32, 64, 128, 256
```

Kita bahas **perbedaan antara BGD, SGD, dan MBGD (Mini-Batch Gradient Descent)** berdasarkan **cara mereka menghitung dan memperbarui bobot (parameter)** setiap iterasi training.

🧠 1. Batch Gradient Descent (BGD)

Konsep:

- Menggunakan **seluruh data training** untuk menghitung **rata-rata error** sebelum memperbarui bobot.
- Jadi, 1 epoch = 1 kali update bobot besar (setelah semua data dihitung).

Contoh langkah:

error = y_pred_all - y_true_all # hitung error semua data
grad_w = (2/m) * X.T @ error # gradien dihitung pakai semua data
w = w - lr * grad_w # update bobot 1x per epoch

Kelebihan:

✅ Konvergen halus (karena dihitung rata-rata semua data)
✅ Stabil, cocok untuk data kecil

Kekurangan:

❌ Lambat untuk dataset besar
❌ Bisa terjebak di local minima karena gradien rata-rata bisa terlalu halus

Analogi:

Seperti menghitung nilai rata-rata seluruh siswa di sekolah dulu, baru memutuskan strategi belajar — lama, tapi pasti.

⚡ 2. Stochastic Gradient Descent (SGD)

Konsep:

- Mengupdate bobot **setiap kali melihat satu data (satu sampel)**.
- 1 epoch = sebanyak jumlah data update.

Contoh langkah:

```
for x_i, y_i in zip(X, y):
    y_pred = np.dot(x_i, w) + b
    error = y_pred - y_i
    w = w - lr * (2 * x_i * error)
```

Kelebihan:

✅ Sangat cepat bereaksi terhadap data baru
✅ Dapat keluar dari local minima (karena fluktuasi tinggi)

Kekurangan:

❌ Konvergensi “bergetar” (tidak stabil)
❌ Hasil akhir bisa noisy

Analogi:

Seperti memperbaiki strategi belajar setiap kali menerima nilai ujian satu siswa — cepat, tapi bisa bikin bingung karena fluktuatif.

⚖️ 3. Mini-Batch Gradient Descent (MBGD)

Konsep:

- Kompromi antara BGD dan SGD.
- Data dibagi menjadi **kelompok kecil (batch)**, misal 4 atau 8 data.
- Update bobot dilakukan **setiap batch**.

Contoh langkah:

```
for i in range(0, m, batch_size):
    X_batch = X[i:i+batch_size]
```

```
    y_batch = y[i:i+batch_size]
    grad_w = (2/batch_size) * X_batch.T @ (X_batch @ w + b - y_batch)
    w = w - lr * grad_w
```

Kelebihan:

✅ Lebih efisien daripada BGD

✅ Lebih stabil daripada SGD

✅ Dapat memanfaatkan paralel GPU

Kekurangan:

❌ Masih ada sedikit noise

❌ Butuh tuning batch_size

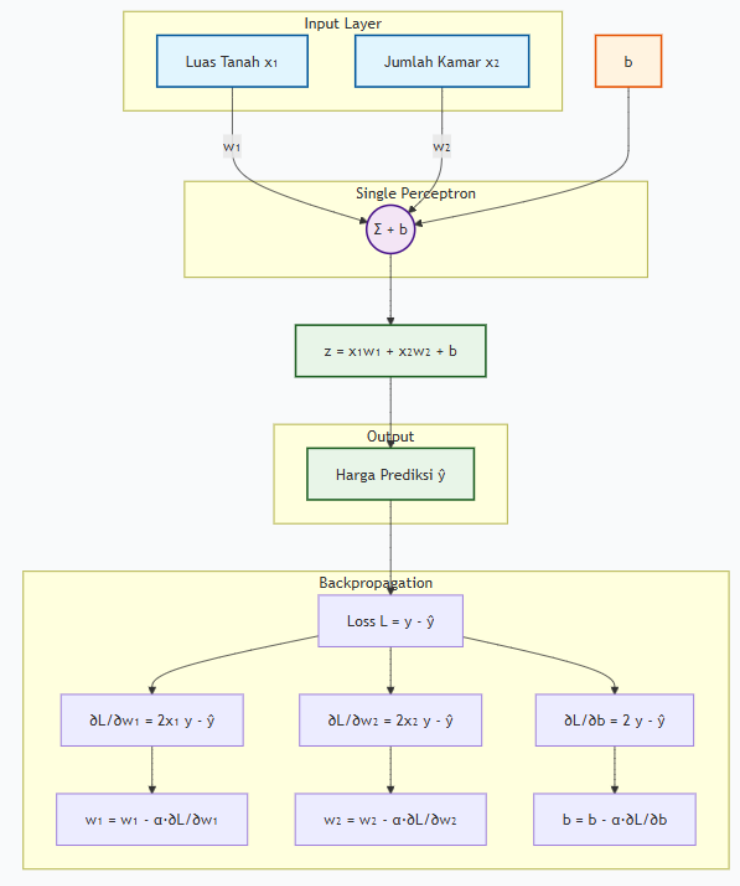
Analogi:

Seperti mengevaluasi strategi belajar tiap kelompok siswa — cepat, tapi masih cukup stabil.

📊 Ringkasan Perbandingan

Metode	Unit Update	Stabilitas	Kecepatan per Epoch	Cocok Untuk
BGD	Semua data	Sangat stabil	Lambat	Dataset kecil
SGD	1 sampel	Fluktuatif	Cepat	Dataset besar, online learning
MBGD	Beberapa sampel (batch)	Cukup stabil	Sedang	Dataset besar + GPU

8. SINGLE PERCEPTRON LINEAR



1. FORWARD PASS (Prediksi)

$z = x_1 \cdot w_1 + x_2 \cdot w_2 + b$

$\hat{y} = z$ (linear activation)

Contoh konkret:

 # Data: luas=80m², kamar=3, harga=450 juta

$x_1 = 80$, $x_2 = 3$, $y = 450$

 # Setelah normalisasi (contoh):

$x_1\_norm = 0.1667$, $x_2\_norm = 0.3333$, $y\_norm = 0.2308$

 # Prediksi:

$z = (0.1667 \times w_1) + (0.3333 \times w_2) + b$

$\hat{y} = z$

2. LOSS FUNCTION (MSE)

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

3. BACKPROPAGATION (Gradients)

$\partial L/\partial w_1 = 2 \cdot (\hat{y} - y) \cdot x_1$

$\partial L/\partial w_2 = 2 \cdot (\hat{y} - y) \cdot x_2$

$\partial L/\partial b = 2 \cdot (\hat{y} - y)$

4. WEIGHT UPDATE (SGD)

$w_1 = w_1 - \alpha \cdot \partial L/\partial w_1$

$w_2 = w_2 - \alpha \cdot \partial L/\partial w_2$

$b = b - \alpha \cdot \partial L/\partial b$

📊 CONTOH PERHITUNGAN LENGKAP

 # Inisialisasi

w = [0.5, -0.2], b = 0.1, α = 0.1

 # Data training pertama setelah normalisasi:

x = [0.1667, 0.3333], y_true = 0.2308

 # FORWARD PASS

z = (0.1667×0.5) + (0.3333×-0.2) + 0.1

= 0.08335 - 0.06666 + 0.1 = 0.11669

y_pred = z = 0.11669

 # LOSS CALCULATION

error = y_pred - y_true = 0.11669 - 0.2308 = -0.11411

loss = $(-0.11411)^2$ = 0.01302

 # BACKPROPAGATION

grad_w1 = 2 × x[0] × error = 2 × 0.1667 × -0.11411 = -0.03804

grad_w2 = 2 × x[1] × error = 2 × 0.3333 × -0.11411 = -0.07607

grad_b = 2 × error = 2 × -0.11411 = -0.22822

 # WEIGHT UPDATE

w1_new = 0.5 - 0.1 × (-0.03804) = 0.503804

w2_new = -0.2 - 0.1 × (-0.07607) = -0.192393

b_new = 0.1 - 0.1 × (-0.22822) = 0.122822

🔢 VISUALISASI DATA TRAINING

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Visualisasi data asli
fig = plt.figure(figsize=(12, 4))

# Plot 1: Data points
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], y, c='red', label='Luas vs Harga')
plt.scatter(X[:, 1], y, c='blue', label='Kamar vs Harga')
plt.xlabel('Luas Tanah (m²) / Jumlah Kamar')
plt.ylabel('Harga (juta)')
plt.legend()
plt.title('Data Training')
```

```
plt.grid(True)

# Plot 2: Loss convergence
plt.subplot(1, 2, 2)
plt.plot(losses)
plt.title('Konvergensi Loss SGD')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.grid(True)

plt.tight_layout()
plt.show()
```

🎯 **KETERANGAN DIAGRAM:**

- **Input**: Luas tanah (x_1) dan jumlah kamar (x_2) yang sudah dinormalisasi
- **Weights**: w_1 dan w_2 yang dipelajari selama training
- **Bias**: b yang menyesuaikan intercept garis regresi
- **Output**: Prediksi harga rumah $\hat{y}$
- **Learning**: Proses update weights berdasarkan error prediksi

```
import numpy as np
import matplotlib.pyplot as plt

# ========================================
# 1 Dataset: [luas_tanah(m²), jumlah_kamar, harga(juta)]
# ========================================
data = np.array([
    [60, 2, 300],
    [80, 3, 450],
    [100, 3, 500],
    [120, 4, 600],
    [140, 4, 750],
    [160, 5, 800],
    [180, 5, 950]
])

# Pisahkan fitur (X) dan target (y)
X = data[:, :2]   # kolom 0-1 → luas, kamar
y = data[:, 2]    # kolom 2 → harga

# ========================================
# 2 Normalisasi Data (Min-Max)
# ========================================
def min_max_normalize(data):
```

```
    return (data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data,
axis=0))

X_norm = min_max_normalize(X)
y_min, y_max = y.min(), y.max()
y_norm = (y - y_min) / (y_max - y_min)

# ========================================
# [3] Inisialisasi Parameter Awal
# ========================================
np.random.seed(42)
w = np.random.randn(2)   # bobot untuk [luas, kamar]
b = np.random.randn()    # bias

# ========================================
# [4] Fungsi Prediksi dan Loss
# ========================================
def predict(X, w, b):
    """Prediksi linear: y_pred = X·w + b"""
    return np.dot(X, w) + b

def mse_loss(y_true, y_pred):
    """Mean Squared Error"""
    return np.mean((y_true - y_pred)**2)

# ========================================
# [5] Stochastic Gradient Descent (SGD)
# ========================================
def train_sgd(X, y, w, b, lr=0.1, epochs=200):
    n = len(X)
    losses = []

    for epoch in range(epochs):
        # Acak urutan data agar model tidak menghafal urutan
        indices = np.random.permutation(n)
        X, y = X[indices], y[indices]

        for i in range(n):
            # Ambil satu sampel (stochastic)
            x_i = X[i]
            y_i = y[i]

            # Feedforward
            y_pred = np.dot(x_i, w) + b
            error = y_pred - y_i
```

```
            # Hitung gradien (turunan loss terhadap w dan b)
            grad_w = 2 * x_i * error
            grad_b = 2 * error

            # Update parameter
            w -= lr * grad_w
            b -= lr * grad_b

        # Simpan loss tiap epoch
        y_pred_all = predict(X, w, b)
        loss = mse_loss(y, y_pred_all)
        losses.append(loss)

        # Tampilkan progress tiap 50 epoch
        if epoch % 50 == 0:
            print(f"Epoch {epoch:3d} → Loss: {loss:.6f}")

    return w, b, losses

# ========================================
# 6 Jalankan Training
# ========================================
print("=== Training dengan SGD (Single Perceptron Linear) ===")
w_trained, b_trained, losses = train_sgd(X_norm, y_norm, w, b)

# ========================================
# 7 Visualisasi Konvergensi
# ========================================
plt.plot(losses)
plt.title("Konvergensi Loss (SGD)")
plt.xlabel("Epoch")
plt.ylabel("MSE Loss")
plt.grid(True)
plt.show()

# ========================================
# 8 Prediksi Rumah Baru
# ========================================
def predict_new(luas, kamar, w, b, X):
    """Prediksi harga rumah baru"""
    # Normalisasi input baru agar sesuai dengan skala training
    luas_norm = (luas - X[:,0].min()) / (X[:,0].max() - X[:,0].min())
    kamar_norm = (kamar - X[:,1].min()) / (X[:,1].max() - X[:,1].min())
```

```
    # Prediksi (skala normalisasi)
    y_pred_norm = predict(np.array([luas_norm, kamar_norm]), w, b)

    # Denormalisasi hasil ke harga asli
    return y_pred_norm * (y_max - y_min) + y_min

print("\n=== Prediksi Rumah Baru ===")
for luas, kamar in [(90,3), (150,4), (200,5)]:
    harga = predict_new(luas, kamar, w_trained, b_trained, X)
    print(f"Luas {luas}m², {kamar} kamar → Prediksi harga: {harga:.0f} juta")
```

```
=== Prediksi Rumah Baru ===
Luas 90m², 3 kamar → Prediksi harga: 463 juta
Luas 150m², 4 kamar → Prediksi harga: 770 juta
Luas 200m², 5 kamar → Prediksi harga: 1026 juta
```

💮 1️⃣ Tujuan Utama

Konsep	Tujuan
SGD (Stochastic Gradient Descent)	Sebuah **metode optimisasi** — cara memperbarui bobot agar *loss* menurun.
Single Perceptron	Sebuah **model** — sistem dengan input, bobot, bias, dan fungsi aktivasi untuk menghasilkan output.

➡️ **Intinya:**

- **SGD** = *cara belajar / algoritma training*
- **Perceptron** = *model yang belajar menggunakan SGD (atau metode lain)*

🧠 2️⃣ Struktur Matematis

🔷 Single Perceptron (Linear)

Satu neuron:

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

Bila memakai aktivasi sigmoid (untuk klasifikasi):

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

👉 Di contoh kamu:

- Tidak ada fungsi aktivasi → artinya ini **Linear Perceptron (regresi)**
- Tujuannya meminimalkan **error (selisih prediksi vs aktual)**

🔷 SGD

Adalah **proses update parameter (w dan b)** menggunakan satu data per langkah.
Langkah-langkahnya:

1. Ambil **1 data** dari dataset
2. Hitung **prediksi**:
$$\hat{y} = w \cdot x + b$$
3. Hitung **error**:
$$e = \hat{y} - y$$
4. Turunkan **loss function (MSE)** terhadap w dan b:
$$\frac{dL}{dw} = 2xe, \frac{dL}{db} = 2e$$
5. Update parameter:
$$w \coloneqq w - \eta \frac{dL}{dw}, b \coloneqq b - \eta \frac{dL}{db}$$

👉 Itulah proses inti **SGD**: perbarui bobot berdasarkan *gradien dari satu sampel*.

⚙ 3 Hubungan antara Keduanya

Aspek	Perceptron	SGD
Peran	Model yang memprediksi	Algoritma yang melatih model
Fungsi utama	Hitung output (feedforward)	Update bobot (backprop)
Formula inti	`y_pred = w·x + b`	`w = w - lr * grad`
Aktivasi	Bisa ada (step/sigmoid)	Tidak perlu – fokus ke optimisasi
Output	Prediksi atau keputusan	Nilai bobot baru (w, b)

🧩 4 Ilustrasi Analogi Sederhana

Bayangkan kamu ingin **memasak nasi (model bagus)**:

Analogi	Penjelasan
Perceptron = panci + beras	Alat dan bahan untuk membuat model
SGD = cara memasak (mengaduk dan memanaskan)	Proses pelatihan supaya hasilnya matang
Loss = rasa nasi (kurang matang / gosong)	Indikator kualitas model
Learning rate = besar api	Kalau terlalu besar → gosong, terlalu kecil → lama matang

---

☑ 5 Dalam Script Kamu

Di script sebelumnya:

```
y_pred = np.dot(x_i, w) + b  # <-- ini perceptron linear
error = y_pred - y_i
grad_w = 2 * x_i * error
w -= lr * grad_w
```

Jadi:

- Baris `y_pred = np.dot(x_i, w) + b` adalah **fungsi perceptron** (forward pass)
- Baris `w -= lr * grad_w` adalah **bagian SGD** (update parameter per data)

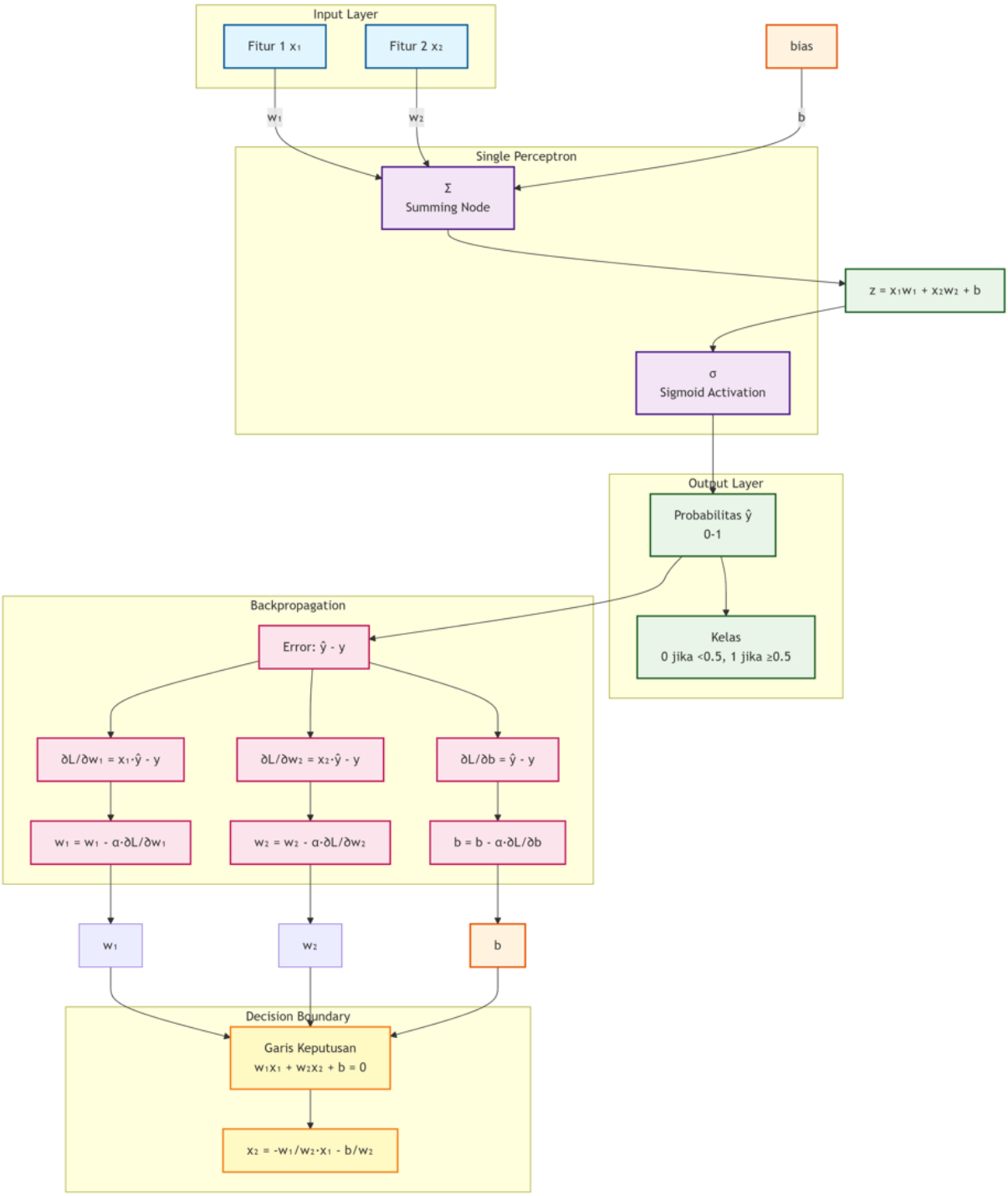
Artinya, script kamu menggabungkan **dua konsep sekaligus:**
Perceptron sebagai *model sederhana linear regression*,
dan SGD sebagai *metode untuk melatih model tersebut*.

🎯 6 Kesimpulan

Komponen	Perceptron	SGD
Apa itu	Model (neuron tunggal)	Algoritma optimisasi
Fokus	Prediksi (ŷ) dari input X	Menyesuaikan bobot w agar loss kecil
Hubungan	Di-*train* menggunakan SGD	Melatih perceptron
Aktivasi	Bisa linear, step, sigmoid	Tidak ada aktivasi
Contoh di kode	`predict()`	`train_sgd()`

Kalimat ringkas untuk diingat:
🧠 “Perceptron adalah modelnya,
SGD adalah cara melatihnya.”

9. SINGLE PERCEPTRON SIGMOID >> KLASIFIKASI /DECISION BOUNDRY



🔄 PROSES YANG DITAMPILKAN:

1. **FORWARD PASS (Prediksi)**
   - **Input** → **Summing Node**: x_1, x_2 dengan weights w_1, w_2
   - **Summing** → **Sigmoid**: $z = x_1w_1 + x_2w_2 + b$
   - **Sigmoid** → **Output**: $\hat{y} = \sigma(z) = 1/(1+e^{-z})$

- **Output** → **Kelas**: Thresholding di 0.5

$z = x_1 \cdot w_1 + x_2 \cdot w_2 + b$
$\hat{y} = \sigma(z) = 1 / (1 + e^{(-z)})$
Contoh konkret:
 # Data: [0.2, 0.3] → kelas 0
$x_1 = 0.2$, $x_2 = 0.3$, $y = 0$

 # Asumsikan weights terlatih:
$w = [8.5, 7.2]$, $b = -6.0$

 # Prediksi:
$z = (0.2 \times 8.5) + (0.3 \times 7.2) + (-6.0)$
$= 1.7 + 2.16 - 6.0 = -2.14$
$\hat{y} = \sigma(-2.14) = 1 / (1 + e^{(2.14)}) \approx 0.105$

2. BACKWARD PASS (Panah Merah)

- **Error Calculation**: $\hat{y} - y$
- **Gradient Computation**:
  - $\partial L/\partial w_1 = x_1 \cdot (\hat{y} - y)$
  - $\partial L/\partial w_2 = x_2 \cdot (\hat{y} - y)$
  - $\partial L/\partial b = \hat{y} - y$
- **Weight Update**: SGD dengan learning rate α

LOSS FUNCTION (Binary Cross-Entropy)
$L = -[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})]$
BACKPROPAGATION (Gradients)
$\partial L/\partial w_1 = x_1 \cdot (\hat{y} - y)$
$\partial L/\partial w_2 = x_2 \cdot (\hat{y} - y)$
$\partial L/\partial b = (\hat{y} - y)$
WEIGHT UPDATE
$w_1 = w_1 - \alpha \cdot \partial L/\partial w_1$
$w_2 = w_2 - \alpha \cdot \partial L/\partial w_2$
$b = b - \alpha \cdot \partial L/\partial b$

3. DECISION BOUNDARY (Panah Kuning)

- **Equation**: $w_1x_1 + w_2x_2 + b = 0$
- **Explicit Form**: $x_2 = -w_1/w_2 \cdot x_1 - b/w_2$

📊 CONTOH PERHITUNGAN LENGKAP

 # FORWARD PASS untuk satu data
x = [0.2, 0.3], y_true = 0
w = [8.5, 7.2], b = -6.0

z = (0.2×8.5) + (0.3×7.2) + (-6.0) = -2.14
y_pred = sigmoid(-2.14) = 0.105

 # LOSS CALCULATION
loss = -[0×log(0.105) + (1-0)×log(1-0.105)]
= -log(0.895) ≈ 0.111

 # BACKPROPAGATION
error = y_pred - y_true = 0.105 - 0 = 0.105
grad_w1 = x[0] × error = 0.2 × 0.105 = 0.021
grad_w2 = x[1] × error = 0.3 × 0.105 = 0.0315
grad_b = error = 0.105

 # WEIGHT UPDATE (α = 0.5)
w1_new = 8.5 - 0.5 × 0.021 = 8.4895
w2_new = 7.2 - 0.5 × 0.0315 = 7.18425
b_new = -6.0 - 0.5 × 0.105 = -6.0525

🎯 DECISION BOUNDARY

Garis pemisah:
$w_1x_1 + w_2x_2 + b = 0$
Dalam bentuk eksplisit:
$x_2 = -(w_1/w_2)\cdot x_1 - b/w_2$
Contoh dengan weights terlatih:
 # Asumsikan w = [8.5, 7.2], b = -6.0
 # Decision boundary:
$x_2 = -(8.5/7.2)\cdot x_1 - (-6.0)/7.2$
$= -1.18\cdot x_1 + 0.833$

Berikut penjelasan komprehensif mengenai **Single Perceptron** untuk Regresi vs Klasifikasi:

🧠 KONSEP DASAR SINGLE PERCEPTRON

Single Perceptron adalah building block neural network terkecil yang terdiri dari:

text

```
Output = Activation(∑(input × weight) + bias)
```

📈 SINGLE PERCEPTRON UNTUK REGRESI

Arsitektur:

python

```
# Script pertama - Prediksi Harga Rumah
ŷ = w₁·x₁ + w₂·x₂ + b  # Linear activation
```

Karakteristik:

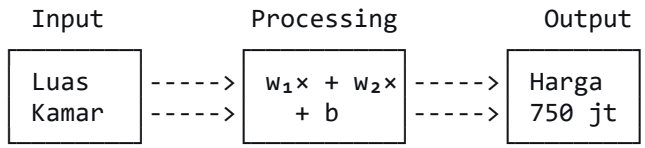
- **Output**: Nilai kontinu (real number)
- **Activation Function**: Linear (atau no activation)
- **Loss Function**: Mean Squared Error (MSE)
- **Target**: Memprediksi nilai numerik

Contoh - Prediksi Harga Rumah:

python

```
# Input: [luas_tanah, jumlah_kamar]
# Output: harga (jutaan rupiah)

# Forward Pass
harga_prediksi = (luas × w₁) + (kamar × w₂) + b

# Loss Calculation
error = (harga_aktual - harga_prediksi)²
```

Visualisasi:



🎯 SINGLE PERCEPTRON UNTUK KLASIFIKASI

Arsitektur:

python

```
# Script kedua - Klasifikasi Biner
z = w₁·x₁ + w₂·x₂ + b
ŷ = σ(z) = 1 / (1 + e^(-z))  # Sigmoid activation
```

Karakteristik:

- **Output**: Probabilitas (0-1)
- **Activation Function**: Sigmoid (untuk biner)
- **Loss Function**: Binary Cross-Entropy

- **Target**: Memprediksi kelas (0 atau 1)

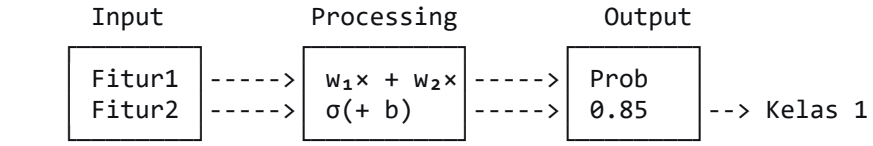
Contoh - Klasifikasi Binary:

python

```
# Input: [fitur1, fitur2]
# Output: probabilitas kelas 1

# Forward Pass
z = (fitur1 × w₁) + (fitur2 × w₂) + b
prob_kelas1 = sigmoid(z)  # antara 0-1

# Decision
if prob_kelas1 ≥ 0.5: kelas = 1
else: kelas = 0
```

Visualisasi:

text



📊 PERBANDINGAN DETAIL

Aspek	Regresi	Klasifikasi
Output	Nilai kontinu (ℝ)	Probabilitas [0,1]
Activation	Linear/Tanpa	Sigmoid (biner)
Loss Function	Mean Squared Error	Binary Cross-Entropy
Target	Prediksi angka	Prediksi kelas
Interpretasi	"Berapa nilai?"	"Termasuk kelas mana?"
Decision Boundary	Garis regresi	Garis pemisah kelas

🎨 VISUALISASI PERBEDAAN

Regresi - Garis Prediksi:

python

```
# Data: Titik-titik scatter
# Model: Garis lurus terbaik
plt.plot(x_range, w*x_range + b)  # Garis regresi
```

Goal: Minimalkan jarak vertikal ke garis

Klasifikasi - Decision Boundary:

```python
# Data: Titik dua warna (kelas 0 dan 1)
# Model: Garis pemisah
plt.contour(decision_boundary)  # Garis pemisah
```

Goal: Maksimalkan pemisahan kelas

🔄 PROSES TRAINING YANG BERBEDA

Regresi (MSE Loss):

```python
# Gradient untuk regresi
error = y_pred - y_true
grad_w = 2 * x * error  # Turunan MSE
```

Klasifikasi (Cross-Entropy Loss):

```python
# Gradient untuk klasifikasi
error = y_pred - y_true  # Sama bentuknya, tapi y_pred = σ(z)
grad_w = x * error       # Turunan Cross-Entropy
```

Catatan menarik: Keduanya punya update rule yang mirip karena sifat matematisnya!

💡 CONTOH APLIKASI NYATA

Regresi:

- Prediksi harga rumah berdasarkan fitur
- Forecast penjualan
- Prediksi suhu/iklim

Klasifikasi:

- Spam detection (spam/not spam)
- Medical diagnosis (sick/healthy)
- Sentiment analysis (positive/negative)

⚡ LIMITASI SINGLE PERCEPTRON

Keduanya memiliki:

- Hanya bisa mempelajari **pola linear**
- **Tidak bisa** menyelesaikan XOR (kecuali dengan transformasi feature)

Bukti Matematis:

Single perceptron membuat **decision boundary linear**:

```
w₁x₁ + w₂x₂ + b = 0  (klasifikasi)
w₁x₁ + w₂x₂ + b = y  (regresi)
```

🤔 KAPAN MEMILIH YANG MANA?

Gunakan Regresi ketika:

- Output adalah nilai numerik kontinu
- Ingin memprediksi "berapa banyak/besar"
- Contoh: harga, suhu, usia

Gunakan Klasifikasi ketika:

- Output adalah kategori/kelas
- Ingin memprediksi "jenis/kelompok mana"
- Contoh: spam/tidak, sakit/sehat, positif/negatif

🎯 KESIMPULAN

Single Perceptron	**Adapter untuk berbagai tugas**
Regresi	Dengan linear activation → prediksi nilai
Klasifikasi	Dengan sigmoid activation → prediksi probabilitas

Intinya: Arsitektur sama, yang membedakan adalah fungsi aktivasi dan loss function yang disesuaikan dengan nature problem yang dihadapi!

Binary Cross Entropy

◈ [1] Mulai dari fungsi utama

Model kita memprediksi nilai:

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

dengan

$$z = w^T x + b$$

Loss function yang digunakan untuk klasifikasi biner adalah **Binary Cross Entropy**:

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

◈ [2] Tujuan: mencari turunan $\frac{\partial L}{\partial w}$

Untuk melatih model (misalnya dengan **gradient descent**), kita perlu tahu seberapa besar perubahan bobot $w$mempengaruhi loss L.
Turunan ini memberi tahu **arah dan besar langkah perbaikan bobot**.
Kita gunakan **aturan rantai (chain rule):**

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w}$$

◈ [3] Hitung satu per satu komponennya

(a) $\frac{\partial L}{\partial \hat{y}}$

Dari:

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

Turunan terhadap $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right) = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}$$

Apa itu Loss Function?

Dalam machine learning, kita ingin model kita membuat prediksi yang mendekati kenyataan. Namun, terkadang prediksi kita tidak tepat, dan untuk mengukur seberapa salah model kita, kita menggunakan **loss function** atau fungsi kerugian. Fungsi kerugian ini memberi tahu kita seberapa besar kesalahan antara prediksi dan label sebenarnya.

Rumus Loss Function - Binary Cross Entropy

Rumus yang kamu lihat adalah rumus **binary cross-entropy loss**. Ini digunakan untuk mengukur kesalahan dalam model klasifikasi biner, yaitu model yang memprediksi dua kemungkinan, misalnya **0** atau **1**. Biasanya, kita menggunakan ini dalam masalah seperti memprediksi apakah email adalah spam (1) atau tidak (0).

Definisi:

- $y$adalah **label sebenarnya**, yaitu nilai yang benar. $y \in 0{,}1$.
- $\hat{y}$adalah **output prediksi model**, yang dihitung dengan fungsi sigmoid.
- Fungsi sigmoid, $\sigma(z) = \frac{1}{1+e^{-z}}$, mengubah output menjadi nilai antara 0 dan 1, yang cocok dengan probabilitas.

Secara singkat, **loss function** ini menghitung seberapa besar perbedaan antara label sebenarnya $y$dan hasil prediksi model $\hat{y}$.

Mari kita bahas bagian-bagian ini:

- **Jika** $y = 1$, maka kita fokus pada bagian pertama $\log(\hat{y})$. Model harus memprediksi angka yang mendekati 1 agar loss kecil.
- **Jika** $y = 0$, maka kita fokus pada bagian kedua $\log(1-\hat{y})$. Model harus memprediksi angka yang mendekati 0 agar loss kecil.

Di mana:

- $y$adalah label yang sebenarnya (0 atau 1),
- $\hat{y}$adalah output prediksi model (probabilitas yang dihitung dengan fungsi sigmoid),
- $\log$ adalah fungsi logaritma natural (logaritma dengan basis e).

Menentukan Turunan Loss terhadap $\hat{y}$

Untuk menghitung turunan dari fungsi loss terhadap $\hat{y}$, kita akan menghitung **turunan parsial** dari Lterhadap $\hat{y}$.

Rumusnya:

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

Menghitung Turunan Setiap Term

Kita akan menghitung turunan dari dua bagian dalam rumus loss secara terpisah.

a. Turunan dari $y \cdot \log(\hat{y})$

Bagian pertama dari loss adalah $y \cdot \log(\hat{y})$. Turunan dari bagian ini terhadap $\hat{y}$adalah:

$$\frac{\partial}{\partial \hat{y}} (y \cdot \log(\hat{y})) = y \cdot \frac{1}{\hat{y}}$$

Untuk menjelaskan bagaimana turunan dari $\log(1 - \hat{y})$menjadi $\frac{1}{1-\hat{y}}$, mari kita mulai dari dasar dengan menjelaskan konsep **turunan logaritma** dan **aturan rantai**.

Turunan Fungsi Logaritma

Sebelum kita masuk ke turunan dari $\log(1 - \hat{y})$, mari kita ingat beberapa aturan dasar turunan untuk fungsi logaritma.

Jika $f(x) = \log(x)$, maka turunan dari $f(x)$terhadap $x$adalah:

$$\frac{d}{dx} \log(x) = \frac{1}{x}$$

Ini adalah aturan dasar turunan untuk fungsi logaritma natural (logaritma dengan basis e).

Menggunakan Aturan Rantai

Jika kita memiliki komposisi fungsi, misalnya $\log\big(g(x)\big)$, maka kita perlu menggunakan **aturan rantai** untuk menghitung turunannya. Aturan rantai menyatakan bahwa:

$$\frac{d}{dx} \log\big(g(x)\big) = \frac{1}{g(x)} \cdot \frac{d}{dx} g(x)$$

Ini berarti kita pertama-tama mengambil turunan dari fungsi bagian dalam $g(x)$, lalu mengalikan hasilnya dengan turunan dari $\log\big(g(x)\big)$.

Turunan dari $\log(1 - \hat{y})$

Sekarang, kita ingin menghitung turunan dari fungsi $\log(1 - \hat{y})$terhadap $\hat{y}$.

Langkah-langkahnya:

- Fungsi dalam logaritma adalah $1 - \hat{y}$, jadi kita anggap $g(\hat{y}) = 1 - \hat{y}$.
- Menggunakan aturan rantai, kita dapat menghitung turunan dari $\log(1 - \hat{y})$:

$$\frac{d}{d\hat{y}} \log(1 - \hat{y}) = \frac{1}{1 - \hat{y}} \cdot \frac{d}{d\hat{y}} (1 - \hat{y})$$

- Turunan dari $1 - \hat{y}$terhadap $\hat{y}$adalah -1, karena turunan dari $\hat{y}$adalah 1, dan turunan dari konstanta 1adalah 0.

Jadi, turunan totalnya adalah:

$$\frac{d}{d\hat{y}} \log(1 - \hat{y}) = \frac{1}{1 - \hat{y}} \cdot (-1)$$

$$= -\frac{1}{1-\hat{y}}$$

Kesimpulan
Jadi, turunan dari $\log(1-\hat{y})$terhadap $\hat{y}$adalah:

$$\frac{d}{d\hat{y}}\log(1-\hat{y}) = -\frac{1}{1-\hat{y}}$$

Turunan dari $(1-y)\cdot\log(1-\hat{y})$
Bagian kedua dari loss adalah $(1-y)\cdot\log(1-\hat{y})$. Turunan dari bagian ini terhadap $\hat{y}$adalah:

$$\frac{\partial}{\partial\hat{y}}\big((1-y)\cdot\log(1-\hat{y})\big) = -(1-y)\cdot\frac{1}{1-\hat{y}}$$

Gabungkan Hasil Turunan
Sekarang kita gabungkan kedua turunan tersebut untuk mendapatkan turunan total dari loss terhadap $\hat{y}$:

$$\frac{\partial L}{\partial\hat{y}} = -\left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]$$

Kita bisa menyederhanakan sedikit, sehingga hasil akhirnya adalah:

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

Penjelasan
Hasil turunan ini mengukur perubahan **loss** terhadap perubahan **output prediksi** $\hat{y}$.

- **Jika** $y=1$(label sebenarnya adalah 1), maka turunan ini mengarah pada $\frac{1}{\hat{y}}$, yang menunjukkan bahwa jika $\hat{y}$(prediksi model) lebih kecil dari 1, maka loss akan semakin besar.
- **Jika** $y=0$(label sebenarnya adalah 0), maka turunan ini mengarah pada $\frac{1}{1-\hat{y}}$, yang menunjukkan bahwa jika $\hat{y}$(prediksi model) lebih besar dari 0, maka loss akan semakin besar.

Apa yang Terjadi Selanjutnya?
Turunan ini akan digunakan dalam algoritma **optimasi** seperti **gradient descent** untuk memperbarui parameter model (misalnya, bobot $w$dan bias b) agar prediksi $\hat{y}$semakin mendekati nilai yang benar, yaitu y.
Dengan demikian, kita dapat **minimalkan** loss function dengan menyesuaikan parameter model secara iteratif.
Mari kita bahas bagian-bagian ini:

- **Jika** $y = 1$, maka kita fokus pada bagian pertama $\log(\hat{y})$. Model harus memprediksi angka yang mendekati 1 agar loss kecil.
- **Jika** $y = 0$, maka kita fokus pada bagian kedua $\log(1 - \hat{y})$. Model harus memprediksi angka yang mendekati 0 agar loss kecil.

Di mana:
- $y$adalah label yang sebenarnya (0 atau 1),
- $\hat{y}$adalah output prediksi model (probabilitas yang dihitung dengan fungsi sigmoid),
- $\log$ adalah fungsi logaritma natural (logaritma dengan basis e).

Menentukan Turunan Loss terhadap $\hat{y}$
Untuk menghitung turunan dari fungsi loss terhadap $\hat{y}$, kita akan menghitung **turunan parsial** dari Lterhadap $\hat{y}$.
Rumusnya:

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

Menghitung Turunan Setiap Term
Kita akan menghitung turunan dari dua bagian dalam rumus loss secara terpisah.
Turunan dari $y \cdot \log(\hat{y})$
Bagian pertama dari loss adalah $y \cdot \log(\hat{y})$. Turunan dari bagian ini terhadap $\hat{y}$adalah:

$$\frac{\partial}{\partial \hat{y}}(y \cdot \log(\hat{y})) = y \cdot \frac{1}{\hat{y}}$$

Untuk menjelaskan bagaimana turunan dari $\log(1 - \hat{y})$menjadi $\frac{1}{1-\hat{y}'}$ mari kita mulai dari dasar dengan menjelaskan konsep **turunan logaritma** dan **aturan rantai**.
Turunan Fungsi Logaritma
Sebelum kita masuk ke turunan dari $\log(1 - \hat{y})$, mari kita ingat beberapa aturan dasar turunan untuk fungsi logaritma.
Jika $f(x) = \log(x)$, maka turunan dari $f(x)$terhadap $x$adalah:

$$\frac{d}{dx}\log(x) = \frac{1}{x}$$

Ini adalah aturan dasar turunan untuk fungsi logaritma natural (logaritma dengan basis e).
Menggunakan Aturan Rantai
Jika kita memiliki komposisi fungsi, misalnya $\log\big(g(x)\big)$, maka kita perlu menggunakan **aturan rantai** untuk menghitung turunannya. Aturan rantai menyatakan bahwa:

$$\frac{d}{dx}\log\big(g(x)\big) = \frac{1}{g(x)} \cdot \frac{d}{dx} g(x)$$

Ini berarti kita pertama-tama mengambil turunan dari fungsi bagian dalam $g(x)$, lalu mengalikan hasilnya dengan turunan dari $\log\big(g(x)\big)$.

Turunan dari $\log(1-\hat{y})$

Sekarang, kita ingin menghitung turunan dari fungsi $\log(1-\hat{y})$terhadap $\hat{y}$.

Langkah-langkahnya:

- Fungsi dalam logaritma adalah $1-\hat{y}$, jadi kita anggap $g(\hat{y}) = 1-\hat{y}$.
- Menggunakan aturan rantai, kita dapat menghitung turunan dari $\log(1-\hat{y})$:

$$\frac{d}{d\hat{y}}\log(1-\hat{y}) = \frac{1}{1-\hat{y}} \cdot \frac{d}{d\hat{y}}(1-\hat{y})$$

- Turunan dari $1-\hat{y}$terhadap $\hat{y}$adalah -1, karena turunan dari $\hat{y}$adalah 1, dan turunan dari konstanta 1adalah 0.

Jadi, turunan totalnya adalah:

$$\frac{d}{d\hat{y}}\log(1-\hat{y}) = \frac{1}{1-\hat{y}} \cdot (-1)$$
$$= -\frac{1}{1-\hat{y}}$$

Kesimpulan

Jadi, turunan dari $\log(1-\hat{y})$terhadap $\hat{y}$adalah:

$$\frac{d}{d\hat{y}}\log(1-\hat{y}) = -\frac{1}{1-\hat{y}}$$

Turunan dari $(1-y)\cdot\log(1-\hat{y})$

Bagian kedua dari loss adalah $(1-y)\cdot\log(1-\hat{y})$. Turunan dari bagian ini terhadap $\hat{y}$adalah:

$$\frac{\partial}{\partial\hat{y}}\big((1-y)\cdot\log(1-\hat{y})\big) = -(1-y)\cdot\frac{1}{1-\hat{y}}$$

Gabungkan Hasil Turunan

Sekarang kita gabungkan kedua turunan tersebut untuk mendapatkan turunan total dari loss terhadap $\hat{y}$:

$$\frac{\partial L}{\partial\hat{y}} = -\left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]$$

Kita bisa menyederhanakan sedikit, sehingga hasil akhirnya adalah:

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

Penjelasan
Hasil turunan ini mengukur perubahan **loss** terhadap perubahan **output prediksi** $\hat{y}$.

- **Jika** $y = 1$(label sebenarnya adalah 1), maka turunan ini mengarah pada $\frac{1}{\hat{y}}$, yang menunjukkan bahwa jika $\hat{y}$(prediksi model) lebih kecil dari 1, maka loss akan semakin besar.
- **Jika** $y = 0$(label sebenarnya adalah 0), maka turunan ini mengarah pada $\frac{1}{1-\hat{y}}$, yang menunjukkan bahwa jika $\hat{y}$(prediksi model) lebih besar dari 0, maka loss akan semakin besar.

Apa yang Terjadi Selanjutnya?
Turunan ini akan digunakan dalam algoritma **optimasi** seperti **gradient descent** untuk memperbarui parameter model (misalnya, bobot $w$dan bias b) agar prediksi $\hat{y}$semakin mendekati nilai yang benar, yaitu y.
Dengan demikian, kita dapat **minimalkan** loss function dengan menyesuaikan parameter model secara iteratif.

(b) $\frac{\partial \hat{y}}{\partial z}$
Karena $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$,
maka turunannya adalah turunan sigmoid:

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$$

Fungsi Sigmoid
Output dari model, $\hat{y}$, dihitung menggunakan fungsi **sigmoid**:

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

Mari kita lanjutkan penjelasan tentang turunan $\frac{\partial \hat{y}}{\partial z}$di mana $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$, dan kita ingin menghitung turunan dari output prediksi $\hat{y}$terhadap nilai z.
Fungsi Sigmoid
Fungsi sigmoid yang digunakan untuk menghitung output $\hat{y}$adalah:

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

Fungsi sigmoid ini mengubah input $z$menjadi nilai antara 0 dan 1. Kita ingin menghitung turunan dari $\hat{y}$terhadap z.
Turunan Fungsi Sigmoid

Untuk menghitung turunan dari $\hat{y}$terhadap z, kita akan menggunakan aturan turunan dari fungsi rasional.

$$\hat{y} = \frac{1}{1 + e^{-z}}$$

Kita akan menggunakan **aturan diferensiasi untuk fungsi rasional** di mana bentuk umum turunan dari $\frac{1}{f(z)}$adalah:

$$\frac{d}{dz}\left(\frac{1}{f(z)}\right) = -\frac{f'(z)}{(f(z))^2}$$

Dalam hal ini, $f(z) = 1 + e^{-z}$, jadi kita perlu menghitung turunan dari $f(z)$.

Turunan dari $\boldsymbol{f(z) = 1 + e^{-z}}$

Turunan dari $f(z) = 1 + e^{-z}$terhadap $z$adalah:

$$f'(z) = -e^{-z}$$

Turunan $\hat{y}$

Sekarang kita dapat menghitung turunan dari $\hat{y}$:

$$\frac{d\hat{y}}{dz} = -\frac{-e^{-z}}{(1 + e^{-z})^2}$$
$$= \frac{e^{-z}}{(1 + e^{-z})^2}$$

Namun, kita dapat menyederhanakan ekspresi ini dengan mengingat bahwa $\hat{y} = \frac{1}{1+e^{-z}}$. Jadi, kita dapat menulis ulang hasil ini:

$$\frac{d\hat{y}}{dz} = \hat{y} \cdot (1 - \hat{y})$$

Penjelasan

Hasil turunan $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$ini memiliki beberapa arti penting dalam konteks machine learning:

- **Jika $\hat{y}$mendekati 0 atau 1**, turunan $\hat{y}(1 - \hat{y})$menjadi kecil, yang berarti perubahan kecil pada $z$tidak akan mengubah output banyak. Ini dikenal sebagai fenomena **vanishing gradient** yang bisa terjadi dalam pelatihan jaringan saraf (neural networks) yang sangat dalam.
- **Jika $\hat{y}$mendekati 0.5**, turunan $\hat{y}(1 - \hat{y})$akan lebih besar, menunjukkan bahwa perubahan pada $z$akan lebih mempengaruhi output.

Kesimpulan

Turunan $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$menunjukkan seberapa sensitif output model $\hat{y}$terhadap perubahan dalam nilai z. Ini sangat penting dalam optimasi, karena digunakan dalam **backpropagation** untuk memperbarui parameter model (misalnya, bobot dan bias).

(c) $\frac{\partial z}{\partial w}$

Dari:

$$z = w^T x + b$$

Turunannya terhadap $w$adalah:

$$\frac{\partial z}{\partial w} = x$$

Di sini, $z$adalah kombinasi linear dari fitur input $x$dan bobot w:

$$z = w^T x + b$$

Di mana:

- $w^T x$adalah perkalian antara bobot dan fitur input.
- $b$adalah bias (penyesuaian).

Sigmoid mengambil nilai $z$dan mengubahnya menjadi angka antara 0 dan 1, yang digunakan sebagai probabilitas.

Mari kita lanjutkan penjelasan tentang turunan $\frac{\partial z}{\partial w}$, di mana $z$adalah kombinasi linear dari bobot dan input, yaitu:

$$z = w^T x + b$$

Di sini, $w$adalah vektor bobot, $x$adalah vektor input, dan $b$adalah bias.

Fungsi $z = \boldsymbol{w^T x + b}$

Fungsi ini adalah representasi dari hubungan linear antara input $x$dan bobot w, ditambah bias b. Dalam konteks model machine learning, ini digunakan untuk menghitung nilai pre-aktivasi sebelum kita melewati fungsi aktivasi (misalnya, fungsi sigmoid).

- $w^T x$adalah perkalian titik (dot product) antara vektor bobot $w$dan vektor input x.
- $b$adalah bias yang ditambahkan untuk memberikan fleksibilitas dalam model.

Menentukan Turunan $\frac{\partial z}{\partial w}$

Sekarang kita ingin menghitung turunan dari $z$terhadap w. Dari rumus $z = w^T x + b$, kita bisa lihat bahwa $w$adalah vektor bobot yang dikalikan dengan vektor input x. Mari kita hitung turunan ini langkah demi langkah.

a. Turunan dari $z = \boldsymbol{w^T x + b}$terhadap $\boldsymbol{w}$

Turunan dari $z$terhadap $w$dihitung dengan aturan diferensiasi dasar untuk vektor. Ingat bahwa $w^T x$adalah penjumlahan dari produk elemen-elemen vektor $w$dan x. Biasanya, kita menganggap $x$adalah konstanta dalam turunan ini (karena kita mengambil turunan terhadap w).
Turunan dari $z = w^T x + b$terhadap $w$adalah:

$$\frac{\partial z}{\partial w} = x$$

Ini karena:

- $w^T x$adalah perkalian titik antara vektor $w$dan x, dan turunan dari $w^T x$terhadap $w$adalah $x$itu sendiri.
- $b$adalah konstanta, sehingga turunan terhadap $w$adalah 0.

Penjelasan

- $\frac{\partial z}{\partial w} = x$berarti bahwa jika kita ingin mengubah nilai z(nilai pre-aktivasi), kita harus mengubah $w$dengan faktor yang berhubungan langsung dengan x, input yang kita berikan.
- Dengan kata lain, perubahan kecil pada $w$akan mempengaruhi $z$sesuai dengan nilai input x.

Hubungan dengan Backpropagation

Turunan ini sangat penting dalam **backpropagation** di jaringan saraf (neural networks), di mana kita menghitung seberapa besar pengaruh perubahan bobot $w$terhadap kesalahan model. Dalam backpropagation, kita menggunakan turunan ini untuk mengupdate bobot $w$secara iteratif agar model dapat meminimalkan kesalahan.
Misalnya, jika kita ingin mengoptimalkan model menggunakan **gradient descent**, kita akan mengupdate bobot $w$dengan rumus:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

Di mana Ladalah loss function, dan ηadalah learning rate.

Kesimpulan

Jadi, turunan $\frac{\partial z}{\partial w} = x$menunjukkan bagaimana perubahan dalam bobot $w$mempengaruhi pre-aktivasi z. Ini digunakan dalam proses pelatihan model untuk menyesuaikan bobot agar model semakin baik dalam membuat prediksi.
Apakah penjelasan ini membantu? Jika ada bagian yang kurang jelas, saya siap membantu lebih lanjut!

🔷 4 Gabungkan semua dengan aturan rantai

$$\frac{\partial L}{\partial w} = \left(\frac{\partial L}{\partial \hat{y}}\right) \cdot \left(\frac{\partial \hat{y}}{\partial z}\right) \cdot \left(\frac{\partial z}{\partial w}\right)$$

Substitusikan hasil-hasil sebelumnya:

$$\frac{\partial L}{\partial w} = \left(\frac{\hat{y} - y}{\hat{y}(1-\hat{y})}\right) \cdot \hat{y}(1-\hat{y}) \cdot x$$

Bagian $\hat{y}(1-\hat{y})$saling **meniadakan**, hasil akhirnya:

$$\boxed{\frac{\partial L}{\partial w} = (\hat{y} - y) \cdot x}$$

🔷 5 Intuisi hasil akhir

- $(\hat{y} - y)$: selisih antara prediksi dan label sebenarnya → **error**.
- x: fitur input yang berkontribusi pada kesalahan.
- $\frac{\partial L}{\partial w}$: memberi tahu seberapa besar bobot $w$perlu diubah agar error mengecil.

💡 Kesimpulan:

Turunan penuh dari fungsi loss terhadap bobot $w$adalah hasil dari rantai turunan tiga tahap:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w} \Rightarrow \frac{\partial L}{\partial w} = (\hat{y} - y) \cdot x$$

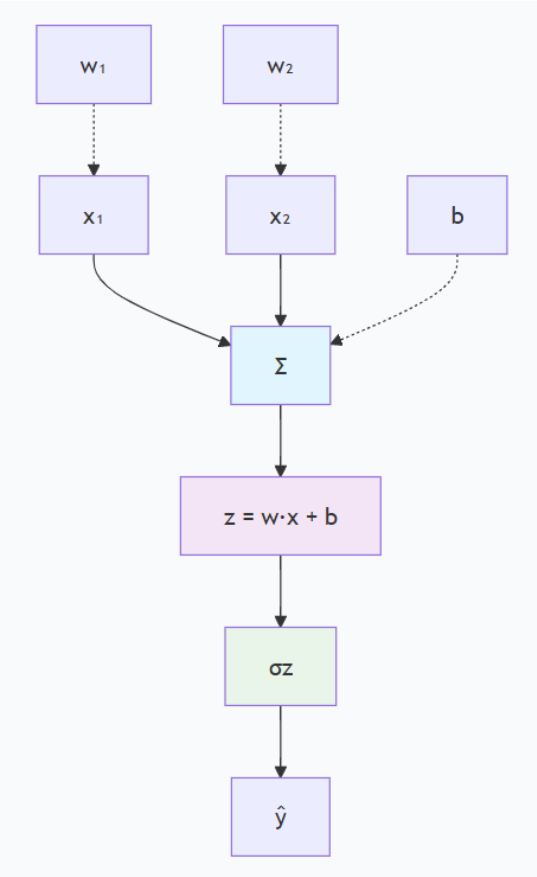
```
error = ŷ - y
grad_w1 = x₁ * error
grad_w2 = x₂ * error
grad_b = error
```

Dalam bentuk vektor:

```
grad_w = x * error
grad_b = error
```

10. SINGLE PERCEPTRON 3INPUT KLASIFIKASI LINEAR ACTIVATION

SINGLE PERCEPTRON GERBANG LOGIKA AND



1. Forward Pass

$z = w_1x_1 + w_2x_2 + b$

$\hat{y} = \sigma(z) = 1/(1 + e^{(-z)})$

Dimana:

- x_1, x_2 = Input features
- w_1, w_2 = Bobot (weights)
- b = Bias
- z = Net input
- σ = Fungsi aktivasi sigmoid
- $\hat{y}$ = Prediksi output

2. Backward Pass (Gradient Descent)

$error = \hat{y} - y$

$w_1 = w_1 - \eta \times x_1 \times error$

$w_2 = w_2 - \eta \times x_2 \times error$

$b = b - \eta \times error$

Dimana:

- y = Target aktual
- η = Learning rate
- error = Kesalahan prediksi

Implementasi dari Kode AND Gate

Berdasarkan output training Anda:

python

```
# Parameter yang dihasilkan (contoh)
w_and = [w₁, w₂]  # Bobot akhir
b_and = b         # Bias akhir

# Forward pass untuk input [1, 1]
z = w₁×1 + w₂×1 + b
ŷ = sigmoid(z) = 1/(1 + e^(-z))
```

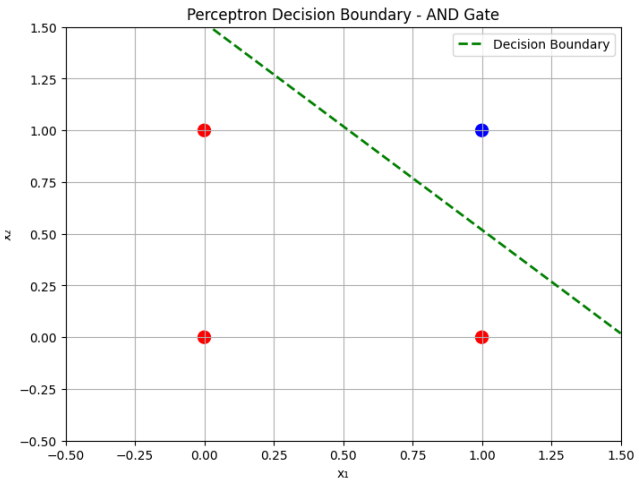
==========

```python
import numpy as np
import matplotlib.pyplot as plt

# ========================================
# CONTOH 1: GERBANG LOGIKA AND
# ========================================

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Data AND - LINEAR SEPARABLE
X_and = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y_and = np.array([0, 0, 0, 1])  # Hanya output 1 jika kedua input 1

print("=== CONTOH 1: GERBANG AND ===")
print("Data AND:")
for i in range(len(X_and)):
    print(f"Input: {X_and[i]} -> Target: {y_and[i]}")

# Training perceptron
np.random.seed(42)
w_and = np.random.randn(2)
```

```
b_and = 0.0
lr = 0.5
epochs = 500

print("\nTraining AND perceptron...")
for epoch in range(epochs):
    for i in range(len(X_and)):
        # Forward pass
        z = np.dot(w_and, X_and[i]) + b_and
        y_pred = sigmoid(z)

        # Backward pass
        error = y_pred - y_and[i]
        w_and -= lr * X_and[i] * error
        b_and -= lr * error

print(f"Bobot akhir: {w_and}")
print(f"Bias akhir: {b_and:.4f}")

# Testing
print("\nHasil Prediksi AND:")
print("Input\t\tTarget\tPrediksi\tProbabilitas")
for i in range(len(X_and)):
    z = np.dot(w_and, X_and[i]) + b_and
    prob = sigmoid(z)
    pred = 1 if prob > 0.5 else 0
    print(f"{X_and[i]}\t{y_and[i]}\t{pred}\t\t{prob:.4f}")


# Visualisasi decision boundary AND gate
def plot_decision_boundary(w, b, X, y):
    plt.figure(figsize=(8, 6))

    # Plot data points
    colors = ['red' if label == 0 else 'blue' for label in y]
    plt.scatter(X[:, 0], X[:, 1], c=colors, s=100)

    # Plot decision boundary (w₁x₁ + w₂x₂ + b = 0)
    x1_boundary = np.array([-0.5, 1.5])
    x2_boundary = (-w[0] * x1_boundary - b) / w[1]

    plt.plot(x1_boundary, x2_boundary, 'g--', linewidth=2, label='Decision
Boundary')
    plt.xlabel('x₁')
    plt.ylabel('x₂')
```

```
    plt.title('Perceptron Decision Boundary - AND Gate')
    plt.legend()
    plt.grid(True)
    plt.xlim(-0.5, 1.5)
    plt.ylim(-0.5, 1.5)
    plt.show()

# Gunakan setelah training
plot_decision_boundary(w_and, b_and, X_and, y_and)
```

```
=== CONTOH 1: GERBANG AND ===
Data AND:
Input: [0 0] -> Target: 0
Input: [0 1] -> Target: 0
Input: [1 0] -> Target: 0
Input: [1 1] -> Target: 1

Training AND perceptron...
Bobot akhir: [7.51210274 7.4997752 ]
Bias akhir: -11.4086

Hasil Prediksi AND:
Input        Target Prediksi     Probabilitas
[0 0] 0      0            0.0000
[0 1] 0      0            0.0197
[1 0] 0      0            0.0199
[1 1] 1      1            0.9735
```



SINGLE PERCEPTRON GERBANG LOGIKA OR

```
import numpy as np
```

```
import matplotlib.pyplot as plt

# ========================================
# CONTOH 2: GERBANG LOGIKA OR
# ========================================

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Data OR - LINEAR SEPARABLE
X_or = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y_or = np.array([0, 1, 1, 1])  # Output 1 jika minimal satu input 1

print("=== CONTOH 2: GERBANG OR ===")
print("Data OR:")
for i in range(len(X_or)):
    print(f"Input: {X_or[i]} -> Target: {y_or[i]}")

# Training perceptron untuk OR
np.random.seed(42)
w_or = np.random.randn(2)
b_or = 0.0
lr = 0.5
epochs = 500

print("\nTraining OR perceptron...")
for epoch in range(epochs):
    for i in range(len(X_or)):
        # Forward pass
        z = np.dot(w_or, X_or[i]) + b_or
```

1. DIAGRAM MLP (Multi-Layer Perceptron) - Struktur Jaringan

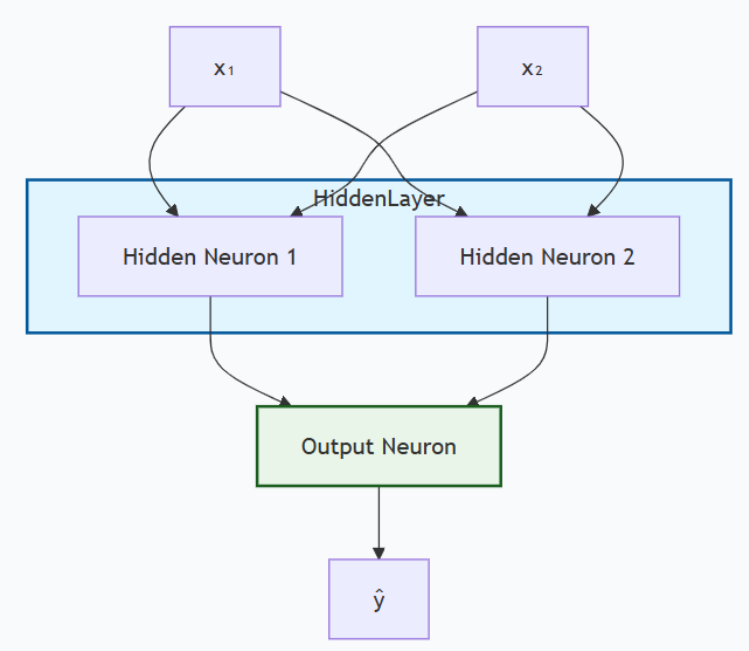
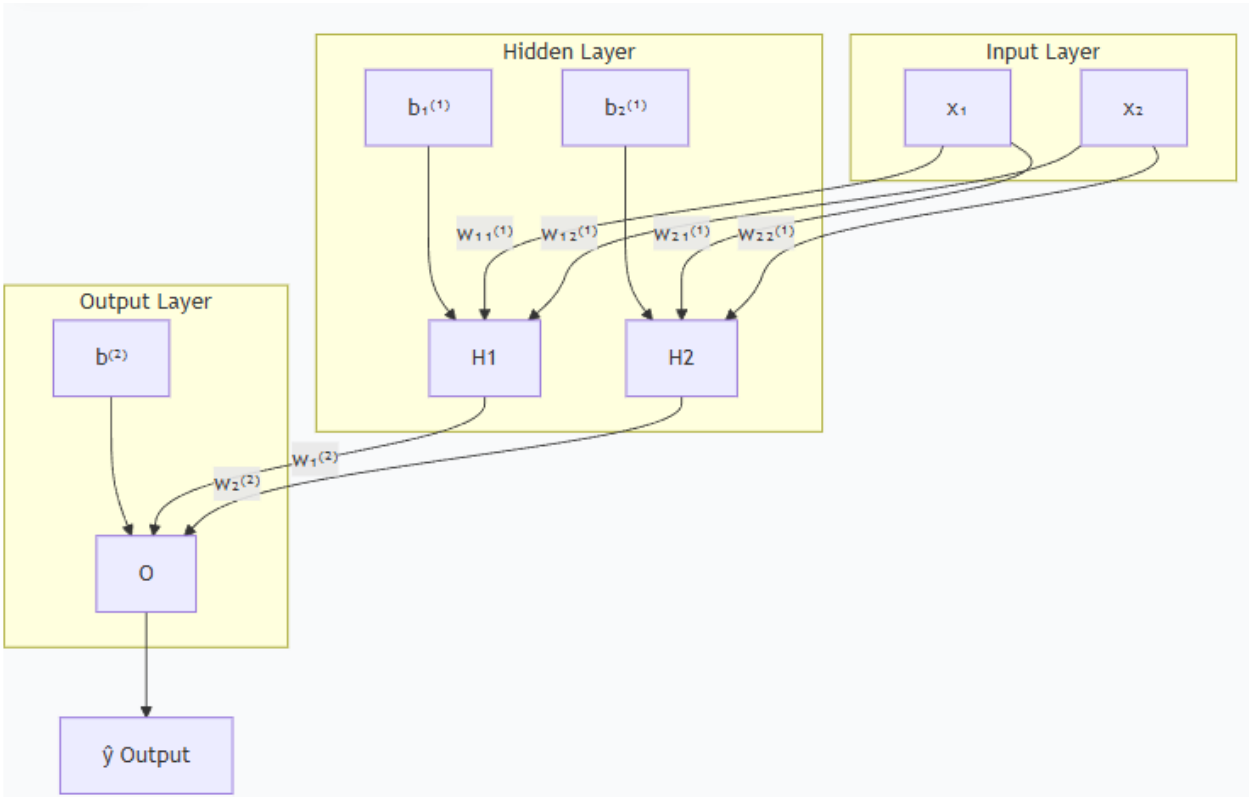


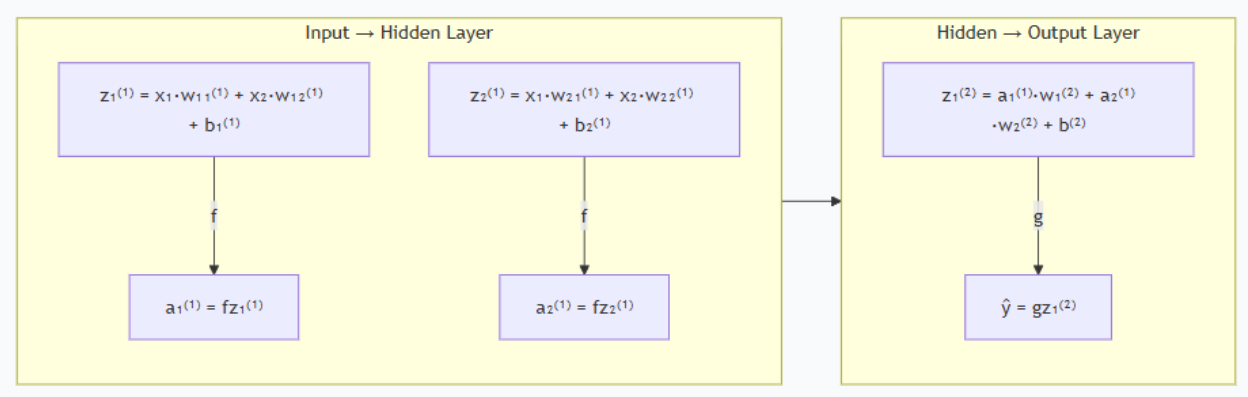
DIAGRAM STRUKTUR JARINGAN MLP



Keterangan:

- **Layer Input**: x_1, x_2 (fitur input)
- **Hidden Layer**: H1, H2 dengan bias $b_1^{(1)}$, $b_2^{(1)}$
- **Output Layer**: O dengan bias $b^{(2)}$
- **Bobot**: $w_{11}^{(1)}$, $w_{12}^{(1)}$, $w_{21}^{(1)}$, $w_{22}^{(1)}$ (hidden), $w_1^{(2)}$, $w_2^{(2)}$ (output)

2. DIAGRAM FEED FORWARD

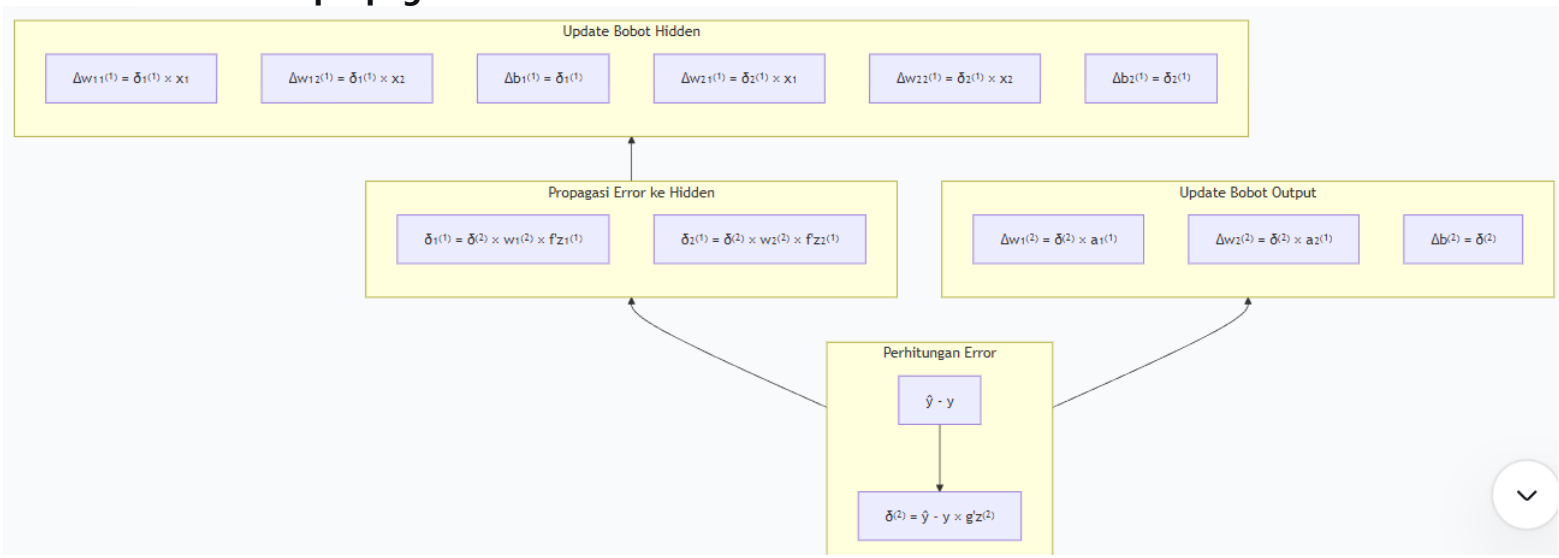


Proses Feed Forward:

1. **Input → Hidden Layer:**
   - $z_1^{(1)} = (x_1 \times w_{11}^{(1)}) + (x_2 \times w_{12}^{(1)}) + b_1^{(1)}$
   - $a_1^{(1)} = f(z_1^{(1)})$ → fungsi aktivasi (sigmoid, ReLU, dll)
   - $z_2^{(1)} = (x_1 \times w_{21}^{(1)}) + (x_2 \times w_{22}^{(1)}) + b_2^{(1)}$
   - $a_2^{(1)} = f(z_2^{(1)})$
2. **Hidden → Output Layer:**
   - $z_1^{(2)} = (a_1^{(1)} \times w_1^{(2)}) + (a_2^{(1)} \times w_2^{(2)}) + b^{(2)}$
   - $\hat{y} = g(z_1^{(2)})$ → fungsi aktivasi output

3. DIAGRAM BACKPROPAGATION

Proses Backpropagation:



A. Error pada Output Layer:

math

$\delta^{(2)} = (\hat{y} - y) \times g'(z^{(2)})$

- **$\hat{y} - y$**: Selisih prediksi dengan target aktual
- **$g'(z^{(2)})$**: Turunan fungsi aktivasi output

B. Update Bobot Output:

math

$\Delta w_1^{(2)} = \delta^{(2)} \times a_1^{(1)}$
$\Delta w_2^{(2)} = \delta^{(2)} \times a_2^{(1)}$
$\Delta b^{(2)} = \delta^{(2)}$

C. Error pada Hidden Layer:

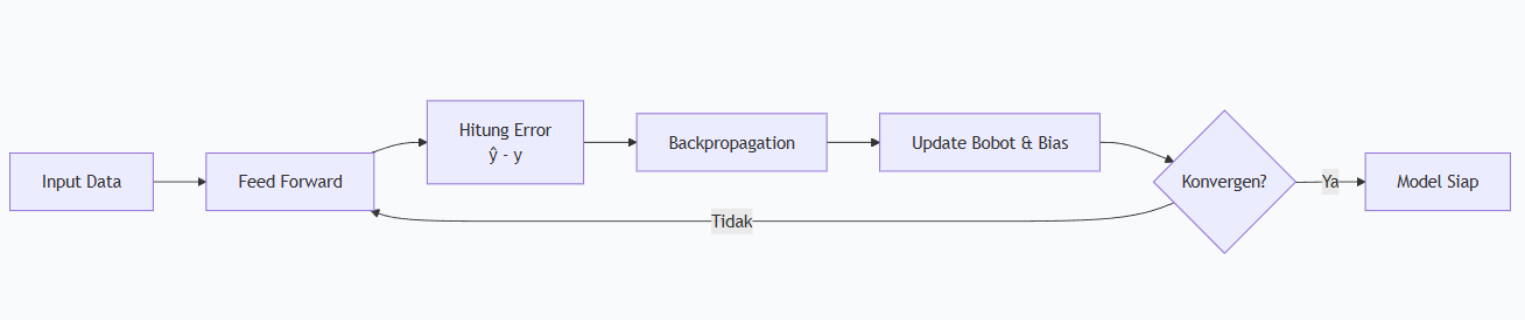
math
$\delta_1^{(1)} = (\delta^{(2)} \times w_1^{(2)}) \times f'(z_1^{(1)})$
$\delta_2^{(1)} = (\delta^{(2)} \times w_2^{(2)}) \times f'(z_2^{(1)})$

D. Update Bobot Hidden:

math
$\Delta w_{11}^{(1)} = \delta_1^{(1)} \times x_1$
$\Delta w_{12}^{(1)} = \delta_1^{(1)} \times x_2$
$\Delta b_1^{(1)} = \delta_1^{(1)}$
$\Delta w_{21}^{(1)} = \delta_2^{(1)} \times x_1$
$\Delta w_{22}^{(1)} = \delta_2^{(1)} \times x_2$
$\Delta b_2^{(1)} = \delta_2^{(1)}$

SIKLUS LENGKAP MLP

Dengan



Berikut contoh kasus sederhana MLP menggunakan numpy dan visualisasi matplotlib:

1. IMPLEMENTASI MLP SEDERHANA

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

# Set style untuk visualisasi
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

class SimpleMLP:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.1):
        # Inisialisasi bobot dan bias
        self.W1 = np.random.randn(input_size, hidden_size) * 0.1
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size) * 0.1
        self.b2 = np.zeros((1, output_size))
```

```
    self.lr = learning_rate

def sigmoid(self, x):
    return 1 / (1 + np.exp(-np.clip(x, -250, 250)))

def sigmoid_derivative(self, x):
    return x * (1 - x)

def forward(self, X):
    # Feed forward
    self.z1 = np.dot(X, self.W1) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    self.a2 = self.sigmoid(self.z2)
    return self.a2

def backward(self, X, y, output):
    # Backpropagation
    m = X.shape[0]

    # Error output layer
    dZ2 = output - y
    dW2 = (1/m) * np.dot(self.a1.T, dZ2)
    db2 = (1/m) * np.sum(dZ2, axis=0, keepdims=True)

    # Error hidden layer
    dA1 = np.dot(dZ2, self.W2.T)
    dZ1 = dA1 * self.sigmoid_derivative(self.a1)
    dW1 = (1/m) * np.dot(X.T, dZ1)
    db1 = (1/m) * np.sum(dZ1, axis=0, keepdims=True)

    # Update weights
    self.W2 -= self.lr * dW2
    self.b2 -= self.lr * db2
    self.W1 -= self.lr * dW1
    self.b1 -= self.lr * db1

def train(self, X, y, epochs):
    self.loss_history = []
    for epoch in range(epochs):
        # Forward pass
        output = self.forward(X)

        # Hitung loss
        loss = np.mean((output - y) ** 2)
        self.loss_history.append(loss)

        # Backward pass
        self.backward(X, y, output)

        if epoch % 1000 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

```
    def predict(self, X):
        output = self.forward(X)
        return (output > 0.5).astype(int)

# Generate dataset sederhana (masalah klasifikasi XOR)
def generate_xor_data():
    X = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])
    y = np.array([[0],
                  [1],
                  [1],
                  [0]])
    return X, y

# Generate dataset non-linear (moons dataset)
X, y = make_moons(n_samples=300, noise=0.2, random_state=42)
y = y.reshape(-1, 1)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

print("Data Shape:")
print(f"X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"X_test: {X_test.shape}, y_test: {y_test.shape}")

# Inisialisasi dan training MLP
mlp = SimpleMLP(input_size=2, hidden_size=4, output_size=1, learning_rate=0.5)
mlp.train(X_train, y_train, epochs=10000)

# Prediksi
train_predictions = mlp.predict(X_train)
test_predictions = mlp.predict(X_test)

# Akurasi
train_accuracy = np.mean(train_predictions == y_train)
test_accuracy = np.mean(test_predictions == y_test)

print(f"\nAkurasi Training: {train_accuracy:.2%}")
print(f"Akurasi Test: {test_accuracy:.2%}")
```

2. VISUALISASI DATA DAN DECISION BOUNDARY

```python
# Visualisasi 1: Data Asli
plt.figure(figsize=(15, 5))

# Subplot 1: Data training
```

```
plt.subplot(1, 3, 1)
plt.scatter(X_train[y_train.flatten() == 0, 0], X_train[y_train.flatten() == 0,
1],
           c='red', label='Class 0', alpha=0.7, s=50)
plt.scatter(X_train[y_train.flatten() == 1, 0], X_train[y_train.flatten() == 1,
1],
           c='blue', label='Class 1', alpha=0.7, s=50)
plt.title('Data Training')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 2: Data test
plt.subplot(1, 3, 2)
plt.scatter(X_test[y_test.flatten() == 0, 0], X_test[y_test.flatten() == 0, 1],
           c='red', label='Class 0', alpha=0.7, s=50)
plt.scatter(X_test[y_test.flatten() == 1, 0], X_test[y_test.flatten() == 1, 1],
           c='blue', label='Class 1', alpha=0.7, s=50)
plt.title('Data Test')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 3: Decision Boundary
plt.subplot(1, 3, 3)
xx, yy = np.meshgrid(np.linspace(-2, 3, 100), np.linspace(-2, 2, 100))
X_grid = np.c_[xx.ravel(), yy.ravel()]
Z = mlp.forward(X_grid)
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, levels=50, alpha=0.8, cmap='RdYlBu')
plt.colorbar(label='Probability')
plt.scatter(X_train[y_train.flatten() == 0, 0], X_train[y_train.flatten() == 0,
1],
           c='red', label='Class 0', alpha=0.7, s=30, edgecolors='black')
plt.scatter(X_train[y_train.flatten() == 1, 0], X_train[y_train.flatten() == 1,
1],
           c='blue', label='Class 1', alpha=0.7, s=30, edgecolors='black')
plt.title('Decision Boundary MLP')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()

plt.tight_layout()
plt.show()
```

3. VISUALISASI TRAINING PROCESS

```python
```

```
# Visualisasi 2: Proses Training
plt.figure(figsize=(15, 5))

# Subplot 1: Loss selama training
plt.subplot(1, 3, 1)
plt.plot(mlp.loss_history, linewidth=2)
plt.title('Loss selama Training')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.yscale('log')
plt.grid(True, alpha=0.3)

# Subplot 2: Akurasi training vs test
plt.subplot(1, 3, 2)
categories = ['Training', 'Test']
accuracies = [train_accuracy, test_accuracy]
colors = ['skyblue', 'lightcoral']

bars = plt.bar(categories, accuracies, color=colors, alpha=0.7,
edgecolor='black')
plt.title('Perbandingan Akurasi')
plt.ylabel('Accuracy')
plt.ylim(0, 1)

# Tambah nilai di atas bar
for bar, acc in zip(bars, accuracies):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{acc:.2%}', ha='center', va='bottom')

plt.grid(True, alpha=0.3, axis='y')

# Subplot 3: Visualisasi arsitektur MLP
plt.subplot(1, 3, 3)
plt.axis('off')
plt.title('Arsitektur MLP', pad=20)

# Gambar diagram MLP sederhana
circle_radius = 0.1

# Input layer
for i in range(2):
    circle = plt.Circle((0.2, 0.7 - i*0.3), circle_radius, fill=True,
color='lightblue', ec='black')
    plt.gca().add_patch(circle)
    plt.text(0.2, 0.7 - i*0.3, f'x{i+1}', ha='center', va='center',
fontweight='bold')

# Hidden layer
for i in range(4):
    circle = plt.Circle((0.5, 0.9 - i*0.25), circle_radius, fill=True,
color='lightgreen', ec='black')
    plt.gca().add_patch(circle)
```

```
    plt.text(0.5, 0.9 - i*0.25, f'H{i+1}', ha='center', va='center',
fontweight='bold')

# Output layer
circle = plt.Circle((0.8, 0.5), circle_radius, fill=True, color='lightcoral',
ec='black')
plt.gca().add_patch(circle)
plt.text(0.8, 0.5, 'Output', ha='center', va='center', fontweight='bold')

# Koneksi
for i in range(2):  # input nodes
    for j in range(4):  # hidden nodes
        plt.plot([0.2 + circle_radius, 0.5 - circle_radius],
                 [0.7 - i*0.3, 0.9 - j*0.25], 'gray', alpha=0.5)

for i in range(4):  # hidden nodes
    plt.plot([0.5 + circle_radius, 0.8 - circle_radius],
             [0.9 - i*0.25, 0.5], 'gray', alpha=0.5)

# Labels
plt.text(0.2, 1.0, 'Input Layer\n(2 neuron)', ha='center', va='center',
fontweight='bold')
plt.text(0.5, 1.0, 'Hidden Layer\n(4 neuron)', ha='center', va='center',
fontweight='bold')
plt.text(0.8, 1.0, 'Output Layer\n(1 neuron)', ha='center', va='center',
fontweight='bold')

plt.xlim(0, 1)
plt.ylim(0.2, 1.1)

plt.tight_layout()
plt.show()
```

4. VISUALISASI WEIGHTS DAN ACTIVATIONS

```python
# Visualisasi 3: Bobot dan Aktivasi
plt.figure(figsize=(15, 5))

# Subplot 1: Bobot input-hidden
plt.subplot(1, 3, 1)
plt.imshow(mlp.W1, cmap='coolwarm', aspect='auto')
plt.colorbar(label='Weight Value')
plt.title('Bobot Input-Hidden (W1)')
plt.xlabel('Hidden Neuron')
plt.ylabel('Input Neuron')
plt.xticks(range(4), [f'H{i+1}' for i in range(4)])
plt.yticks(range(2), ['x1', 'x2'])

# Tambah nilai di setiap cell
for i in range(2):
```

```
    for j in range(4):
        plt.text(j, i, f'{mlp.W1[i, j]:.2f}',
                ha='center', va='center', fontweight='bold')

# Subplot 2: Bobot hidden-output
plt.subplot(1, 3, 2)
plt.imshow(mlp.W2, cmap='coolwarm', aspect='auto')
plt.colorbar(label='Weight Value')
plt.title('Bobot Hidden-Output (W2)')
plt.xlabel('Output Neuron')
plt.ylabel('Hidden Neuron')
plt.xticks([0], ['Output'])
plt.yticks(range(4), [f'H{i+1}' for i in range(4)])

# Tambah nilai di setiap cell
for i in range(4):
    plt.text(0, i, f'{mlp.W2[i, 0]:.2f}',
            ha='center', va='center', fontweight='bold')

# Subplot 3: Contoh aktivasi untuk satu sample
plt.subplot(1, 3, 3)
sample_idx = 0
sample = X_train[sample_idx:sample_idx+1]
activations = mlp.forward(sample)

layers = ['Input', 'Hidden', 'Output']
values = [sample.flatten(),
          mlp.a1.flatten(),
          mlp.a2.flatten()]

colors = ['lightblue', 'lightgreen', 'lightcoral']
for i, (layer, val, color) in enumerate(zip(layers, values, colors)):
    plt.barh(layer, val[0] if i == 0 else val.mean(), color=color, alpha=0.7,
edgecolor='black')
    plt.text(val[0] if i == 0 else val.mean(), i, f'{val[0] if i == 0 else
val.mean():.3f}',
             va='center', ha='left', fontweight='bold')

plt.title('Aktivasi untuk Satu Sample')
plt.xlabel('Nilai Aktivasi')
plt.grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.show()

# Print detail model
print("\n=== DETAIL MODEL MLP ===")
print(f"Arsitektur: 2-4-1")
print(f"Learning Rate: {mlp.lr}")
print(f"Final Loss: {mlp.loss_history[-1]:.4f}")
print(f"\nBobot Input-Hidden:")
print(mlp.W1)
```

```
print(f"\nBias Hidden:")
print(mlp.b1)
print(f"\nBobot Hidden-Output:")
print(mlp.W2)
print(f"\nBias Output:")
print(mlp.b2)
```

Berikut 2 contoh tambahan MLP dengan kasus yang berbeda:

CONTOH 1: MLP untuk Regresi (Prediksi Harga Rumah)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression

print("=== CONTOH 1: MLP UNTUK REGRESI ===")

# Generate dataset regresi
X_reg, y_reg = make_regression(n_samples=200, n_features=1, noise=10,
random_state=42)
y_reg = y_reg.reshape(-1, 1)

# Normalisasi data
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_reg_scaled = scaler_X.fit_transform(X_reg)
y_reg_scaled = scaler_y.fit_transform(y_reg)

# Split data manual
split_idx = int(0.8 * len(X_reg_scaled))
X_train_reg = X_reg_scaled[:split_idx]
y_train_reg = y_reg_scaled[:split_idx]
X_test_reg = X_reg_scaled[split_idx:]
y_test_reg = y_reg_scaled[split_idx:]

print(f"Data shape - Training: {X_train_reg.shape}, Test: {X_test_reg.shape}")

# Fungsi untuk MLP Regresi
def initialize_parameters_regression(input_size, hidden_size, output_size):
    W1 = np.random.randn(input_size, hidden_size) * 0.1
    b1 = np.zeros((1, hidden_size))
    W2 = np.random.randn(hidden_size, output_size) * 0.1
    b2 = np.zeros((1, output_size))
    return W1, b1, W2, b2

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
```

```
    return (x > 0).astype(float)

def forward_propagation_regression(X, W1, b1, W2, b2):
    z1 = np.dot(X, W1) + b1
    a1 = relu(z1)  # ReLU untuk hidden layer
    z2 = np.dot(a1, W2) + b2
    a2 = z2  # Linear activation untuk output (regresi)
    return z1, a1, z2, a2

def backward_propagation_regression(X, y, z1, a1, z2, a2, W1, W2, learning_rate):
    m = X.shape[0]

    # Error output layer (derivative of MSE)
    dZ2 = a2 - y
    dW2 = (1/m) * np.dot(a1.T, dZ2)
    db2 = (1/m) * np.sum(dZ2, axis=0, keepdims=True)

    # Error hidden layer
    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * relu_derivative(a1)
    dW1 = (1/m) * np.dot(X.T, dZ1)
    db1 = (1/m) * np.sum(dZ1, axis=0, keepdims=True)

    # Update weights
    W2_updated = W2 - learning_rate * dW2
    b2_updated = b2 - learning_rate * db2
    W1_updated = W1 - learning_rate * dW1
    b1_updated = b1 - learning_rate * db1

    return W1_updated, b1_updated, W2_updated, b2_updated

def train_mlp_regression(X, y, hidden_size=10, learning_rate=0.01, epochs=5000):
    input_size = X.shape[1]
    output_size = y.shape[1] if len(y.shape) > 1 else 1

    W1, b1, W2, b2 = initialize_parameters_regression(input_size, hidden_size,
output_size)
    loss_history = []

    for epoch in range(epochs):
        # Forward propagation
        z1, a1, z2, a2 = forward_propagation_regression(X, W1, b1, W2, b2)

        # Hitung loss (MSE)
        loss = np.mean((a2 - y) ** 2)
        loss_history.append(loss)

        # Backward propagation
        W1, b1, W2, b2 = backward_propagation_regression(X, y, z1, a1, z2, a2,
W1, W2, learning_rate)

        if epoch % 1000 == 0:
```

```
            print(f"Epoch {epoch}, Loss: {loss:.4f}")

    return W1, b1, W2, b2, loss_history

def predict_regression(X, W1, b1, W2, b2):
    _, _, z2, _ = forward_propagation_regression(X, W1, b1, W2, b2)
    return z2

# Training MLP untuk regresi
print("\nTraining MLP Regresi...")
W1_reg, b1_reg, W2_reg, b2_reg, loss_history_reg = train_mlp_regression(
    X_train_reg, y_train_reg, hidden_size=10, learning_rate=0.01, epochs=5000
)

# Prediksi
train_predictions_scaled = predict_regression(X_train_reg, W1_reg, b1_reg,
W2_reg, b2_reg)
test_predictions_scaled = predict_regression(X_test_reg, W1_reg, b1_reg, W2_reg,
b2_reg)

# Kembalikan ke skala asli
train_predictions = scaler_y.inverse_transform(train_predictions_scaled)
test_predictions = scaler_y.inverse_transform(test_predictions_scaled)
y_train_original = scaler_y.inverse_transform(y_train_reg)
y_test_original = scaler_y.inverse_transform(y_test_reg)

# Hitung RMSE
train_rmse = np.sqrt(np.mean((train_predictions - y_train_original) ** 2))
test_rmse = np.sqrt(np.mean((test_predictions - y_test_original) ** 2))

print(f"\nRMSE Training: {train_rmse:.2f}")
print(f"RMSE Test: {test_rmse:.2f}")

# Visualisasi hasil regresi
plt.figure(figsize=(15, 5))

# Subplot 1: Data asli dan prediksi
plt.subplot(1, 3, 1)
plt.scatter(X_reg, y_reg, alpha=0.6, label='Data Asli')
plt.scatter(X_test_reg * scaler_X.scale_ + scaler_X.mean_, test_predictions,
           color='red', label='Prediksi Test', s=50)
plt.title('MLP untuk Regresi')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 2: Loss selama training
plt.subplot(1, 3, 2)
plt.plot(loss_history_reg)
plt.title('Loss selama Training (Regresi)')
plt.xlabel('Epoch')
```

```
plt.ylabel('MSE Loss')
plt.yscale('log')
plt.grid(True, alpha=0.3)

# Subplot 3: Prediksi vs Aktual
plt.subplot(1, 3, 3)
all_predictions = scaler_y.inverse_transform(
    predict_regression(X_reg_scaled, W1_reg, b1_reg, W2_reg, b2_reg)
)
plt.scatter(y_reg, all_predictions, alpha=0.6)
plt.plot([y_reg.min(), y_reg.max()], [y_reg.min(), y_reg.max()], 'r--',
linewidth=2)
plt.title('Prediksi vs Aktual')
plt.xlabel('Nilai Aktual')
plt.ylabel('Prediksi MLP')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

CONTOH 2: MLP untuk Klasifikasi Multi-Kelas (MNIST-like)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.preprocessing import OneHotEncoder

print("\n=== CONTOH 2: MLP UNTUK KLASIFIKASI MULTI-KELAS ===")

# Generate dataset multi-kelas (3 classes)
X_multi, y_multi = make_classification(
    n_samples=500, n_features=2, n_informative=2, n_redundant=0,
    n_classes=3, n_clusters_per_class=1, random_state=42
)

# One-hot encoding untuk output
encoder = OneHotEncoder(sparse_output=False)
y_multi_encoded = encoder.fit_transform(y_multi.reshape(-1, 1))

# Split data
split_idx = int(0.8 * len(X_multi))
X_train_multi = X_multi[:split_idx]
y_train_multi = y_multi_encoded[:split_idx]
X_test_multi = X_multi[split_idx:]
y_test_multi = y_multi_encoded[split_idx:]
y_test_original = y_multi[split_idx:]

print(f"Data shape - Training: {X_train_multi.shape}, Test:
{X_test_multi.shape}")
print(f"Jumlah kelas: {len(np.unique(y_multi))}")
```

```
# Fungsi untuk MLP Multi-Kelas
def initialize_parameters_multi_class(input_size, hidden_size, output_size):
    W1 = np.random.randn(input_size, hidden_size) * 0.1
    b1 = np.zeros((1, hidden_size))
    W2 = np.random.randn(hidden_size, output_size) * 0.1
    b2 = np.zeros((1, output_size))
    return W1, b1, W2, b2

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)

def forward_propagation_multi_class(X, W1, b1, W2, b2):
    z1 = np.dot(X, W1) + b1
    a1 = np.tanh(z1)  # Tanh untuk hidden layer
    z2 = np.dot(a1, W2) + b2
    a2 = softmax(z2)  # Softmax untuk output (multi-class)
    return z1, a1, z2, a2

def backward_propagation_multi_class(X, y, z1, a1, z2, a2, W1, W2,
learning_rate):
    m = X.shape[0]

    # Error output layer (derivative of cross-entropy with softmax)
    dZ2 = a2 - y
    dW2 = (1/m) * np.dot(a1.T, dZ2)
    db2 = (1/m) * np.sum(dZ2, axis=0, keepdims=True)

    # Error hidden layer
    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * (1 - np.tanh(z1) ** 2)  # Derivative of tanh
    dW1 = (1/m) * np.dot(X.T, dZ1)
    db1 = (1/m) * np.sum(dZ1, axis=0, keepdims=True)

    # Update weights
    W2_updated = W2 - learning_rate * dW2
    b2_updated = b2 - learning_rate * db2
    W1_updated = W1 - learning_rate * dW1
    b1_updated = b1 - learning_rate * db1

    return W1_updated, b1_updated, W2_updated, b2_updated

def compute_accuracy(y_true, y_pred):
    return np.mean(np.argmax(y_true, axis=1) == np.argmax(y_pred, axis=1))

def train_mlp_multi_class(X, y, hidden_size=8, learning_rate=0.1, epochs=8000):
    input_size = X.shape[1]
    output_size = y.shape[1]

    W1, b1, W2, b2 = initialize_parameters_multi_class(input_size, hidden_size,
output_size)
```

```
    loss_history = []
    accuracy_history = []

    for epoch in range(epochs):
        # Forward propagation
        z1, a1, z2, a2 = forward_propagation_multi_class(X, W1, b1, W2, b2)

        # Hitung loss (cross-entropy)
        loss = -np.mean(y * np.log(a2 + 1e-8))
        loss_history.append(loss)

        # Hitung akurasi
        accuracy = compute_accuracy(y, a2)
        accuracy_history.append(accuracy)

        # Backward propagation
        W1, b1, W2, b2 = backward_propagation_multi_class(X, y, z1, a1, z2, a2,
W1, W2, learning_rate)

        if epoch % 1000 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}, Accuracy: {accuracy:.2%}")

    return W1, b1, W2, b2, loss_history, accuracy_history

def predict_multi_class(X, W1, b1, W2, b2):
    _, _, _, a2 = forward_propagation_multi_class(X, W1, b1, W2, b2)
    return a2

# Training MLP untuk multi-class
print("\nTraining MLP Multi-Kelas...")
W1_multi, b1_multi, W2_multi, b2_multi, loss_history_multi,
accuracy_history_multi = train_mlp_multi_class(
    X_train_multi, y_train_multi, hidden_size=8, learning_rate=0.1, epochs=8000
)

# Prediksi
train_predictions_proba = predict_multi_class(X_train_multi, W1_multi, b1_multi,
W2_multi, b2_multi)
test_predictions_proba = predict_multi_class(X_test_multi, W1_multi, b1_multi,
W2_multi, b2_multi)

train_predictions = np.argmax(train_predictions_proba, axis=1)
test_predictions = np.argmax(test_predictions_proba, axis=1)
train_actual = np.argmax(y_train_multi, axis=1)
test_actual = y_test_original

train_accuracy = np.mean(train_predictions == train_actual)
test_accuracy = np.mean(test_predictions == test_actual)

print(f"\nAkurasi Training: {train_accuracy:.2%}")
print(f"Akurasi Test: {test_accuracy:.2%}")
```

```
# Visualisasi hasil multi-class classification
plt.figure(figsize=(15, 5))

# Subplot 1: Data asli
plt.subplot(1, 3, 1)
for class_idx in range(3):
    plt.scatter(X_multi[y_multi == class_idx, 0], X_multi[y_multi == class_idx,
1],
                label=f'Class {class_idx}', alpha=0.7, s=50)
plt.title('Data Asli (3 Kelas)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 2: Decision boundary
plt.subplot(1, 3, 2)
xx, yy = np.meshgrid(np.linspace(-3, 3, 100), np.linspace(-3, 3, 100))
X_grid = np.c_[xx.ravel(), yy.ravel()]
Z_proba = predict_multi_class(X_grid, W1_multi, b1_multi, W2_multi, b2_multi)
Z = np.argmax(Z_proba, axis=1)
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3, cmap='Set3')
for class_idx in range(3):
    plt.scatter(X_multi[y_multi == class_idx, 0], X_multi[y_multi == class_idx,
1],
                label=f'Class {class_idx}', alpha=0.7, s=30)
plt.title('Decision Boundary MLP')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, alpha=0.3)

# Subplot 3: Loss dan Akurasi selama training
plt.subplot(1, 3, 3)
plt.plot(loss_history_multi, label='Loss', color='red')
plt.ylabel('Cross-Entropy Loss', color='red')
plt.tick_params(axis='y', labelcolor='red')

plt.twinx()
plt.plot(accuracy_history_multi, label='Accuracy', color='blue')
plt.ylabel('Accuracy', color='blue')
plt.tick_params(axis='y', labelcolor='blue')

plt.title('Loss dan Akurasi selama Training')
plt.xlabel('Epoch')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```
# Confusion Matrix sederhana
print("\n=== CONFUSION MATRIX (Test Set) ===")
confusion_matrix = np.zeros((3, 3))
for true, pred in zip(test_actual, test_predictions):
    confusion_matrix[true, pred] += 1

print("True \\ Pred | Class 0 | Class 1 | Class 2")
print("-" * 45)
for i in range(3):
    row = f"Class {i}      |"
    for j in range(3):
        row += f"   {int(confusion_matrix[i, j]):2d}   |"
    print(row)
```

CONTOH 3: Perbandingan Arsitektur MLP

```python
print("\n=== CONTOH 3: PERBANDINGAN ARSITEKTUR MLP ===")

# Fungsi untuk testing berbagai arsitektur
def test_different_architectures(X, y, architectures, epochs=2000):
    results = {}

    for arch_name, hidden_size in architectures.items():
        print(f"\nTesting {arch_name} (Hidden: {hidden_size})...")

        W1, b1, W2, b2, loss_history = train_mlp_regression(
            X, y, hidden_size=hidden_size, learning_rate=0.01, epochs=epochs
        )

        # Prediksi dan hitung RMSE
        predictions = predict_regression(X, W1, b1, W2, b2)
        rmse = np.sqrt(np.mean((predictions - y) ** 2))

        results[arch_name] = {
            'hidden_size': hidden_size,
            'final_loss': loss_history[-1],
            'rmse': rmse,
            'loss_history': loss_history
        }

        print(f"Final Loss: {loss_history[-1]:.4f}, RMSE: {rmse:.4f}")

    return results

# Test berbagai arsitektur
architectures = {
    'Small': 2,
    'Medium': 8,
    'Large': 32,
    'Very Large': 64
```

```
}

results = test_different_architectures(X_train_reg, y_train_reg, architectures,
epochs=2000)

# Visualisasi perbandingan arsitektur
plt.figure(figsize=(15, 5))

# Subplot 1: Loss comparison
plt.subplot(1, 3, 1)
for arch_name, result in results.items():
    plt.plot(result['loss_history'][:500], label=f'{arch_name}
(H={result["hidden_size"]})')
plt.title('Perbandingan Loss Berbagai Arsitektur')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.legend()
plt.yscale('log')
plt.grid(True, alpha=0.3)

# Subplot 2: Final RMSE comparison
plt.subplot(1, 3, 2)
arch_names = list(results.keys())
rmses = [results[arch]['rmse'] for arch in arch_names]
colors = ['lightblue', 'lightgreen', 'lightcoral', 'gold']

bars = plt.bar(arch_names, rmses, color=colors, alpha=0.7, edgecolor='black')
plt.title('Perbandingan RMSE Berbagai Arsitektur')
plt.ylabel('RMSE')
for bar, rmse in zip(bars, rmses):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{rmse:.3f}', ha='center', va='bottom')
plt.grid(True, alpha=0.3, axis='y')

# Subplot 3: Final loss comparison
plt.subplot(1, 3, 3)
final_losses = [results[arch]['final_loss'] for arch in arch_names]

bars = plt.bar(arch_names, final_losses, color=colors, alpha=0.7,
edgecolor='black')
plt.title('Perbandingan Final Loss')
plt.ylabel('Final Loss')
plt.yscale('log')
for bar, loss in zip(bars, final_losses):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() * 1.1,
             f'{loss:.4f}', ha='center', va='bottom')
plt.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print("\n=== KESIMPULAN ===")
```

```
print("1. MLP untuk Regresi: Menggunakan aktivasi linear di output layer")
print("2. MLP untuk Multi-Kelas: Menggunakan softmax di output layer")
print("3. Arsitektur yang lebih besar cenderung konvergen lebih cepat")
print("4. Namun arsitektur terlalu besar dapat overfit pada data training")
```

Ringkasan 3 Contoh MLP:

1. **Regresi**: Output linear, loss function MSE, untuk prediksi nilai kontinu
2. **Klasifikasi Multi-Kelas**: Output softmax, loss function cross-entropy, untuk klasifikasi >2 kelas
3. **Perbandingan Arsitektur**: Menunjukkan pengaruh jumlah neuron hidden terhadap performa

---

visualisasi decision boundary untuk kasus XOR

```
import numpy as np
import matplotlib.pyplot as plt

# ==============================
# Inisialisasi Parameter
# ==============================
def initialize_parameters(input_size, hidden_size, output_size):
    W1 = np.random.randn(input_size, hidden_size) * 0.1
    b1 = np.zeros((1, hidden_size))
    W2 = np.random.randn(hidden_size, output_size) * 0.1
    b2 = np.zeros((1, output_size))
    return {"W1": W1, "b1": b1, "W2": W2, "b2": b2}

def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -250, 250)))

def sigmoid_derivative(x):
    return x * (1 - x)

def forward(X, params):
    z1 = np.dot(X, params["W1"]) + params["b1"]
    a1 = sigmoid(z1)
    z2 = np.dot(a1, params["W2"]) + params["b2"]
    a2 = sigmoid(z2)
    cache = {"a1": a1, "a2": a2}
    return a2, cache

def backward(X, y, params, cache, lr=0.1):
    m = X.shape[0]
    dZ2 = cache["a2"] - y
    dW2 = (1/m) * np.dot(cache["a1"].T, dZ2)
    db2 = (1/m) * np.sum(dZ2, axis=0, keepdims=True)

    dA1 = np.dot(dZ2, params["W2"].T)
    dZ1 = dA1 * sigmoid_derivative(cache["a1"])
    dW1 = (1/m) * np.dot(X.T, dZ1)
```

```
    db1 = (1/m) * np.sum(dZ1, axis=0, keepdims=True)

    params["W2"] -= lr * dW2
    params["b2"] -= lr * db2
    params["W1"] -= lr * dW1
    params["b1"] -= lr * db1
    return params

def train(X, y, params, epochs, lr=0.1):
    for epoch in range(epochs):
        output, cache = forward(X, params)
        params = backward(X, y, params, cache, lr)
    return params

def predict(X, params):
    output, _ = forward(X, params)
    return (output > 0.5).astype(int)

# ==============================
# Dataset XOR
# ==============================
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],
              [1],
              [1],
              [0]])

# ==============================
# Training
# ==============================
params = initialize_parameters(input_size=2, hidden_size=4, output_size=1)
params = train(X, y, params, epochs=10000, lr=0.5)

# ==============================
# Visualisasi Decision Boundary
# ==============================
# Buat grid titik di area [0,1] x [0,1]
xx, yy = np.meshgrid(np.linspace(-0.2, 1.2, 200),
                     np.linspace(-0.2, 1.2, 200))
grid = np.c_[xx.ravel(), yy.ravel()]

# Prediksi untuk setiap titik grid
Z = predict(grid, params)
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.6)

# Plot data asli XOR
```

```
plt.scatter(X[:,0], X[:,1], c=y[:,0], cmap=plt.cm.RdBu, edgecolors='k', s=100)

plt.title("Decision Boundary MLP untuk XOR")
plt.xlabel("Input 1")
plt.ylabel("Input 2")
plt.show()
```

Baik Edy, mari kita buat versi **MLP untuk XOR dengan Keras/TensorFlow**. Dengan Keras, kita tidak perlu lagi menulis manual fungsi forward, backward, dan update bobot seperti di versi NumPy. Semua itu diurus oleh library.

Script XOR dengan Keras/TensorFlow

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

# ==============================
# Dataset XOR
# ==============================
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]], dtype=np.float32)
y = np.array([[0],
              [1],
              [1],
              [0]], dtype=np.float32)

# ==============================
# Definisi Model MLP
# ==============================
model = Sequential([
    Dense(4, input_dim=2, activation='sigmoid'),  # hidden layer dengan 4
neuron
    Dense(1, activation='sigmoid')                # output layer
])

# ==============================
# Kompilasi Model
# ==============================
# Optimizer SGD dengan learning rate 0.5
optimizer = SGD(learning_rate=0.5)
model.compile(optimizer=optimizer, loss='mean_squared_error',
metrics=['accuracy'])

# ==============================
# Training
# ==============================
history = model.fit(X, y, epochs=10000, verbose=0)

# ==============================
```

```
# Evaluasi
# ============================
predictions = (model.predict(X) > 0.5).astype(int)

print("\nPrediksi XOR dengan Keras:")
for i in range(len(X)):
    print(f"{X[i]} -> {predictions[i][0]} (target: {y[i][0]})")

# Akurasi
loss, acc = model.evaluate(X, y, verbose=0)
print(f"\nAkurasi: {acc:.2%}")
```

🧠 Perbedaan Utama dari Versi NumPy

Aspek	NumPy (manual)	Keras/TensorFlow
Forward Propagation	Ditulis manual dengan `np.dot` dan `sigmoid`.	Otomatis lewat `Dense` layer.
Backward Propagation	Ditulis manual: hitung gradien, update bobot.	Otomatis dengan `model.compile` + optimizer.
Loss Function	Manual: `np.mean((output - y)**2)`.	Ditentukan saat compile (`loss='mean_squared_error'`).
Optimizer	Manual update bobot dengan learning rate.	Gunakan `SGD(learning_rate=0.5)` atau optimizer lain.
Training Loop	Loop `for epoch in range(...)` dengan print loss.	`model.fit(X, y, epochs=...)` otomatis.
Prediksi	Manual threshold `(output > 0.5)`.	`model.predict(X)` lalu threshold.
Ekstensi	Harus menulis ulang jika ingin menambah layer/fitur.	Tinggal tambah `Dense` atau layer lain.

📊 Intinya

- **NumPy** → edukatif, kita belajar detail forward/backward dan update bobot. Cocok untuk memahami konsep dasar.
- **Keras/TensorFlow** → praktis, semua detail low-level diurus library. Cocok untuk eksperimen cepat, model lebih kompleks, dan deployment.

👉 Mau saya tambahkan **visualisasi loss curve dari `history`** supaya terlihat bagaimana error menurun selama training dengan Keras?

Mengapa Keras/TensorFlow mempermudah coding ML

Kamu sudah memahami GD, single perceptron, dan MLP dengan NumPy. Keras/TensorFlow adalah langkah berikutnya yang membuat kamu fokus pada desain arsitektur dan eksperimen, bukan lagi detail low-level seperti derivatif, update bobot, dan loop training. Berikut fitur-fitur inti yang nyata terasa manfaatnya saat naik level.

Abstraksi layer dan model yang ekspresif

- **Layer deklaratif:** Kamu cukup menyusun blok seperti Dense, Conv2D, LSTM dengan satu baris; tidak perlu menulis `np.dot` dan aktivasi manual.
    - Contoh:
    - ```
      model = Sequential([
        Dense(64, activation='relu', input_shape=(n_features,)),
        Dense(1, activation='sigmoid')
      ])
      ```
- **API fungsional untuk model kompleks:** Memudahkan arsitektur bercabang/merge, skip-connections, dan multi-input/output.
    - Contoh skip-connection:
    - ```
      x = Input(shape=(n_features,))
      h = Dense(64, activation='relu')(x)
      y = Concatenate()([h, x])
      out = Dense(1, activation='sigmoid')(y)
      model = Model(x, out)
      ```

Otomasi training loop dan optimisasi

- **Fit otomatis:** `model.fit` menangani forward, backward, batching, shuffling, dan logging—kamu tinggal set epochs dan batch_size.
    - Contoh:
    - ```
      model.compile(optimizer=SGD(learning_rate=0.1), loss='mse',
      metrics=['accuracy'])
      ```
    - ```
      history = model.fit(X_train, y_train, epochs=200, batch_size=32,
      validation_split=0.2)
      ```
- **Optimizer siap pakai:** SGD, Momentum, Adam, RMSProp, AdaGrad—cukup ganti satu baris untuk eksperimen.
    - Contoh:
    - ```
      optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
      ```
- **Regularisasi terintegrasi:** Dropout, L2/L1, batch normalization untuk mencegah overfitting tanpa rumit.
    - Contoh:
    - ```
      Dense(64, activation='relu',
      kernel_regularizer=tf.keras.regularizers.l2(1e-4))
      ```

Pipeline data yang scalable dan bersih

- **tf.data untuk input efisien:** Pipeline streaming, batching, caching, prefetch, dan augmentasi on-the-fly—penting saat data besar.
    - Contoh:
    - ```
      ds = tf.data.Dataset.from_tensor_slices((X, y)) \
            .shuffle(1000).batch(64).prefetch(tf.data.AUTOTUNE)
      model.fit(ds, epochs=50)
      ```
- **Integrasi format industri:** TFRecords, image/text loaders, dan augmentasi dengan `tf.image` atau `tf.keras.layers` seperti `RandomFlip`, `RandomRotation`.

Monitoring, callbacks, dan kontrol training tingkat lanjut

- **Callbacks bawaan:** EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, TensorBoard untuk logging metrik, grafik loss, dan profiling.
      ```

    - Contoh:
    - ```
      callbacks = [
        tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
        tf.keras.callbacks.ModelCheckpoint('best.keras', save_best_only=True),
        tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5)
      ]
      model.fit(X_train, y_train, epochs=200, callbacks=callbacks, validation_data=(X_val, y_val))
      ```
- **Custom training saat butuh fleksibilitas:** `train_step` dengan `tf.GradientTape` memberi kontrol penuh tanpa kehilangan ekosistem Keras.
    - Contoh minimal:
    - ```
      class MyModel(tf.keras.Model):
          def __init__(self):
              super().__init__(); self.d1 = Dense(64, activation='relu'); self.out = Dense(1, 'sigmoid')
          def call(self, x): return self.out(self.d1(x))
          def train_step(self, data):
              x, y = data
              with tf.GradientTape() as tape:
                  yhat = self(x, training=True)
                  loss = self.compiled_loss(y, yhat)
              grads = tape.gradient(loss, self.trainable_variables)
              self.optimizer.apply_gradients(zip(grads, self.trainable_variables))
              self.compiled_metrics.update_state(y, yhat)
              return {m.name: m.result() for m in self.metrics}
      ```

Ekosistem siap produksi

- **Export dan deployment mudah:** `model.save(...)` ke format Keras/TF SavedModel, lalu jalankan di server, mobile (TensorFlow Lite), atau browser (TensorFlow.js).
- **Hardware acceleration:** GPU/TPU mempercepat training tanpa ganti kode; cukup pasang runtime yang sesuai.
- **Compatibility luas:** Pretrained models, KerasCV/NLP, model zoo, dan integrasi dengan ONNX untuk interoperabilitas.

Perubahan pola pikir dari NumPy ke Keras

- **Dari low-level ke high-level:** Kamu tidak lagi mengelola matriks dan gradien per-layer; fokus ke arsitektur, regularisasi, dan eksperimen hyperparameter.
- **Dari batch manual ke pipeline robust:** Gunakan `tf.data` untuk performa dan kebersihan kode.
- **Dari loop custom ke kontrol via callbacks:** Hentikan training saat optimal, simpan bobot terbaik, turunkan LR otomatis—semua satu baris.
- **Dari prototype ke production:** Format model dan akselerasi hardware membuat transisi ke aplikasi nyata mulus.

Rekomendasi langkah berikutnya untuk kamu

- **Uji arsitektur MLP bertingkat:** Tambahkan layer, bandingkan ReLU vs sigmoid, coba Adam vs SGD, dan gunakan EarlyStopping.
- **Buat pipeline `tf.data`:** Latih disiplin data input yang scalable, termasuk split, cache, prefetch.
- **Latih kebiasaan monitoring:** Gunakan TensorBoard sejak awal agar siswa melihat kurva loss/accuracy dan dinamika training.
- **Eksperimen regularisasi:** Dropout, L2, dan BatchNorm untuk menunjukkan dampaknya pada overfitting secara visual.

Oke Edy, mari kita buat urutan langkah yang runtut dan mudah dipahami untuk kasus ini: **model XOR dengan Keras → disimpan → dipanggil lewat API Flask → menerima JSON input → mengembalikan prediksi.**

🔑 Langkah-langkah

1. Latih dan Simpan Model XOR

- Buat model sederhana dengan Keras (Dense layer).
- Latih dengan data XOR (`[0,0],[0,1],[1,0],[1,1]`).
- Simpan model ke file `.h5`.

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Data XOR
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

# Model sederhana
model = Sequential([
    Dense(4, input_dim=2, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X, y, epochs=500, verbose=0)

# Simpan model
model.save("xor_model.h5")
```

---

2. Siapkan Flask API

- Buat file `app.py`.
- Load model saat server Flask dijalankan.
- Buat endpoint `/predict` yang menerima JSON `{ "x1": ..., "x2": ... }`.

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np

app = Flask(__name__)
```

```
model = load_model("xor_model.h5")

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    x1 = data['x1']
    x2 = data['x2']

    # Bentuk input ke model
    input_data = np.array([[x1, x2]])
    prediction = model.predict(input_data)

    # Konversi ke nilai 0/1
    result = int(round(prediction[0][0]))

    return jsonify({
        "x1": x1,
        "x2": x2,
        "prediction": result
    })

if __name__ == '__main__':
    app.run(debug=True)
```

---

3. Jalankan Flask Server

```
python app.py
```

Server akan berjalan di `http://127.0.0.1:5000`.

---

4. Uji dengan POST Request

Gunakan `curl` atau `Postman` untuk mengirim data JSON:

```
curl -X POST http://127.0.0.1:5000/predict \
     -H "Content-Type: application/json" \
     -d '{"x1": 1, "x2": 0}'
```

📌 Response:

```
{
  "x1": 1,
  "x2": 0,
  "prediction": 1
}
```

🧩 Ringkasan Alur

1. **Latih model XOR** → simpan `.h5`.
2. **Flask API** → load model saat start.
3. **Endpoint `/predict`** → terima JSON input.
4. **Prediksi** → kirim hasil sebagai JSON response.

Mau saya tambahkan **diagram alur sederhana** (flowchart) supaya lebih visual untuk siswa atau tim yang belajar bersama Edy?
Mantap Edy! Kita naik level ke **image classification CIFAR** dengan integrasi **Flask API + ReactJS (Vite)**. Ini cocok banget buat demo interaktif ke siswa atau tim. Berikut alur dan contoh implementasinya:

🔗 Alur Sistem: CIFAR Image Classification via Flask + React

1. Latih dan Simpan Model CIFAR

Gunakan Keras untuk melatih model klasifikasi CIFAR-10, lalu simpan ke `.h5`.

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Normalisasi
x_train = x_train / 255.0
x_test = x_test / 255.0

# Model CNN sederhana
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Simpan model
model.save("cifar_model.h5")
```

---

2. Flask API untuk Prediksi Gambar

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np
from PIL import Image
import io

app = Flask(__name__)
model = load_model("cifar_model.h5")
labels =
['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truc
k']

@app.route('/predict', methods=['POST'])
def predict():
    file = request.files['image']
    img = Image.open(file).resize((32,32)).convert('RGB')
    img_array = np.array(img) / 255.0
    img_array = img_array.reshape(1,32,32,3)
```

```
    prediction = model.predict(img_array)
    label = labels[np.argmax(prediction)]

    return jsonify({"prediction": label})

if __name__ == '__main__':
    app.run(debug=True)
```

---

3. Frontend ReactJS (Vite)

```
// App.jsx
import { useState } from 'react'
import axios from 'axios'

function App() {
  const [image, setImage] = useState(null)
  const [result, setResult] = useState("")

  const handleUpload = async () => {
    const formData = new FormData()
    formData.append("image", image)

    const res = await axios.post("http://localhost:5000/predict", formData)
    setResult(res.data.prediction)
  }

  return (
    <div>
      <h1>CIFAR Image Classifier</h1>
      <input type="file" onChange={e => setImage(e.target.files[0])} />
      <button onClick={handleUpload}>Predict</button>
      <h2>Prediction: {result}</h2>
    </div>
  )
}

export default App
```

🖼 Diagram Alur Sistem

Berikut diagram alur visual untuk memperjelas integrasi:

🚀 Ringkasan Alur

1. **Model CIFAR dilatih dan disimpan**.
2. **Flask API menerima gambar via POST**.
3. **Model memprediksi label gambar**.
4. **Frontend ReactJS (Vite)** kirim gambar dan tampilkan hasil prediksi.

Kalau kamu mau, saya bisa bantu buat versi **dockerized** atau **deployment ke server lokal/Raspberry Pi** untuk demo offline. Mau lanjut ke situ?

Keren Edy, kita lanjut ke integrasi **MNIST digit recognition** dengan **Flask API + ReactJS (Vite)** dan fitur **canvas interaktif** untuk menggambar angka langsung. Ini sangat cocok untuk pembelajaran visual dan hands-on!

Alur Sistem: MNIST + Canvas + Flask + React

1. Latih dan Simpan Model MNIST

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Model sederhana
model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Simpan model
model.save("mnist_model.h5")
```

---

2. Flask API untuk Prediksi Gambar Digit

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np
from PIL import Image
import io
import base64

app = Flask(__name__)
model = load_model("mnist_model.h5")

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    img_data = data['image']  # base64 string

    # Decode base64 ke image
    img_bytes = base64.b64decode(img_data.split(',')[1])
    img = Image.open(io.BytesIO(img_bytes)).convert('L').resize((28,28))
    img_array = np.array(img) / 255.0
    img_array = img_array.reshape(1,28,28)
```

```
    prediction = model.predict(img_array)
    label = int(np.argmax(prediction))

    return jsonify({"prediction": label})

if __name__ == '__main__':
    app.run(debug=True)
```

---

3. Frontend ReactJS (Vite) dengan Canvas

```
// App.jsx
import { useRef, useState } from 'react'
import axios from 'axios'

function App() {
  const canvasRef = useRef(null)
  const [result, setResult] = useState("")

  const handlePredict = async () => {
    const canvas = canvasRef.current
    const image = canvas.toDataURL("image/png")

    const res = await axios.post("http://localhost:5000/predict", {
      image: image
    })

    setResult(res.data.prediction)
  }

  const clearCanvas = () => {
    const ctx = canvasRef.current.getContext("2d")
    ctx.fillStyle = "white"
    ctx.fillRect(0, 0, 280, 280)
  }

  return (
    <div>
      <h1>MNIST Digit Recognizer</h1>
      <canvas
        ref={canvasRef}
        width={280}
        height={280}
        style={{ border: "1px solid black", background: "white" }}
        onMouseDown={e => {
          const ctx = canvasRef.current.getContext("2d")
          ctx.lineWidth = 20
          ctx.lineCap = "round"
          ctx.strokeStyle = "black"
          ctx.beginPath()
          ctx.moveTo(e.nativeEvent.offsetX, e.nativeEvent.offsetY)
          canvasRef.current.onmousemove = ev => {
            ctx.lineTo(ev.offsetX, ev.offsetY)
            ctx.stroke()
          }
        }}
        onMouseUp={() => {
          canvasRef.current.onmousemove = null
```

```
        }}
      />
      <br />
      <button onClick={handlePredict}>Predict</button>
      <button onClick={clearCanvas}>Clear</button>
      <h2>Prediction: {result}</h2>
    </div>
  )
}

export default App
```

🖼 Diagram Alur Visual

Berikut diagram alur sistem MNIST canvas interaktif:

🚀 Ringkasan Alur

1. **Model MNIST dilatih dan disimpan**.
2. **Flask API menerima gambar base64 dari canvas**.
3. **Model memprediksi angka**.
4. **ReactJS (Vite)** tampilkan canvas dan hasil prediksi.

Mau saya bantu tambahkan fitur **preview grayscale 28x28** atau **visualisasi heatmap prediksi** di React-nya? Cocok banget buat edukasi siswa!

Oke Edy, kita tambahkan dua fitur edukatif ke ReactJS (Vite) frontend untuk MNIST:

🧩 1. Preview Grayscale 28x28

Tujuannya agar siswa bisa melihat versi input yang dikirim ke model, bukan hanya kanvas besar.

🔧 Langkah:

- Tambahkan canvas kecil (28x28) untuk preview.
- Render ulang gambar besar ke versi kecil menggunakan `getImageData` dan `putImageData`.

🧪 Contoh kode:

```
const previewCanvasRef = useRef(null)

const updatePreview = () => {
  const srcCanvas = canvasRef.current
  const destCanvas = previewCanvasRef.current
  const ctx = destCanvas.getContext("2d")

  // Ambil data dari canvas besar
  const imageData = srcCanvas.getContext("2d").getImageData(0, 0, 280, 280)

  // Buat canvas sementara
  const tempCanvas = document.createElement("canvas")
  tempCanvas.width = 28
  tempCanvas.height = 28
  const tempCtx = tempCanvas.getContext("2d")
```

```
  tempCtx.putImageData(imageData, 0, 0)
  tempCtx.drawImage(srcCanvas, 0, 0, 280, 280, 0, 0, 28, 28)

  // Tampilkan ke preview canvas
  ctx.drawImage(tempCanvas, 0, 0)
}
```

📦 Tambahkan di JSX:

```
<canvas ref={previewCanvasRef} width={28} height={28} style={{ border: "1px solid gray" }} />
```

🔥 2. Visualisasi Heatmap Prediksi

Tujuannya agar siswa bisa melihat confidence level dari model untuk setiap digit (0–9).

🔧 Langkah:

- Tambahkan bar horizontal atau grid untuk 10 kelas.
- Gunakan response `prediction` dan `confidence` array dari Flask.

📏 Contoh response Flask:

```
{
  "prediction": 7,
  "confidence": [0.01, 0.02, 0.03, 0.01, 0.05, 0.02, 0.03, 0.80, 0.02, 0.01]
}
```

📏 Contoh visualisasi:

```
{confidence.map((val, idx) => (
  <div key={idx} style={{
    width: `${val * 100}%`,
    background: idx === result ? "orange" : "gray",
    margin: "2px",
    color: "white"
  }}>
    {idx}: {(val * 100).toFixed(1)}%
  </div>
))}
```

🎯 Manfaat untuk Siswa

- Melihat **transformasi input** dari kanvas ke 28x28.
- Memahami **confidence model** dan distribusi prediksi.
- Belajar **debugging visual** dan interpretasi model.

Kalau kamu mau, saya bisa bantu buat versi **modular komponen React** atau **export ke PDF hasil prediksi dan heatmap** untuk laporan siswa. Mau lanjut ke situ?
Tentu Edy, berikut penjelasan tahapan belajar **CNN (Convolutional Neural Network)** menggunakan contoh **MNIST digit recognition** yang sudah kamu integrasikan dengan **ReactJS (Vite) + Flask API**. Fokusnya adalah membantu siswa memahami **perbedaan CNN vs MLP**, dan bagaimana CNN bekerja secara visual dan modular.

🧠 Tahapan Belajar CNN dengan MNIST + React + Flask

1. Review MLP (Multilayer Perceptron)

Sebelum masuk ke CNN, siswa perlu memahami:

- Input MNIST: gambar 28x28 piksel → di-*flatten* jadi 784 fitur.
- MLP hanya menggunakan **Dense layer** → semua piksel dianggap independen.
- Kelemahan: tidak mengenali **spasial** atau **lokasi piksel**.

📌 *Analogi*: Seperti membaca huruf tanpa tahu bentuk atau posisi — hanya melihat angka-angka.

---

2. Motivasi CNN

CNN dirancang untuk:

- Menangkap **pola lokal** (misalnya garis, lengkungan).
- Memanfaatkan **struktur spasial** gambar.
- Lebih efisien dan akurat untuk data visual.

📌 *Analogi*: CNN seperti mata manusia yang mengenali bentuk angka dari goresan dan posisi.

---

3. Struktur CNN Sederhana untuk MNIST

```
model = Sequential([
  Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
  MaxPooling2D((2,2)),
  Flatten(),
  Dense(128, activation='relu'),
  Dense(10, activation='softmax')
])
```

🧩 Penjelasan Layer:

- **Conv2D**: Menangkap fitur lokal (misalnya ujung angka).
- **MaxPooling2D**: Merangkum area, membuat model tahan noise.
- **Flatten + Dense**: Klasifikasi akhir seperti MLP.

---

4. Integrasi ke Flask API

- Model CNN disimpan `.h5`.
- Flask menerima gambar dari React canvas (base64).
- Gambar diubah ke grayscale 28x28 → diprediksi oleh CNN.
- Response JSON dikirim ke React.

---

5. Interaksi Visual di ReactJS (Vite)

- Siswa menggambar angka di canvas.
- Preview 28x28 grayscale ditampilkan.
- Heatmap confidence ditampilkan → siswa bisa lihat distribusi prediksi.

📌 *Manfaat Visualisasi*:

- Siswa melihat **input yang sebenarnya** ke CNN.
- Siswa memahami **confidence model** dan **ketidakpastian**.
- Bisa eksperimen: “Apa yang terjadi jika angka digambar miring?”

🎓 Tahapan Belajar Bertingkat

Tahap	Fokus	Aktivitas
1. MLP	Konsep dasar NN	Latih MLP untuk MNIST
2. CNN	Konvolusi & spasial	Bandingkan hasil CNN vs MLP

Tahap	Fokus	Aktivitas
3. API	Deployment	Kirim gambar ke Flask
4. React	Interaksi	Gambar angka, lihat prediksi
5. Visualisasi	Interpretasi	Lihat preview & heatmap

🔍 Tips Pengajaran

- Gunakan analogi visual: **filter CNN = kaca pembesar**.
- Tunjukkan hasil prediksi dari angka yang "tidak jelas".
- Ajak siswa modifikasi filter size, jumlah layer, dan lihat dampaknya.

Kalau kamu mau, saya bisa bantu buat **modul interaktif** atau **worksheet siswa** untuk eksperimen CNN vs MLP, termasuk prediksi, visualisasi, dan analisis kesalahan. Mau lanjut ke situ?
Bagus sekali Edy, kamu sudah menyusun roadmap pembelajaran yang runtut dari dasar sampai deployment 👏. Mari kita lanjut ke **Tahap 1: MLP – Konsep dasar NN, latih MLP untuk MNIST**.

📌 Tahap 1: MLP untuk MNIST

Fokus

- Mengenal **neural network dasar (MLP)**
- Memahami alur: input → hidden layer → output
- Melatih model sederhana untuk klasifikasi angka (0–9)

Aktivitas

1. **Load dataset MNIST**
   - Gunakan `keras.datasets.mnist`
   - Normalisasi data ke range [0,1]
   - Flatten gambar 28×28 menjadi vektor 784
2. **Bangun arsitektur MLP sederhana**
   - Input: 784 neuron
   - Hidden layer: misalnya 128 neuron, aktivasi ReLU
   - Output: 10 neuron, aktivasi softmax
3. **Compile & train**
   - Loss: categorical crossentropy
   - Optimizer: SGD atau Adam
   - Metrics: accuracy
4. **Evaluasi**
   - Bandingkan akurasi train vs test
   - Catat hasil sebagai baseline sebelum CNN

💻 Contoh kode Keras/TensorFlow

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

```
from tensorflow.keras.utils import to_categorical

# 1. Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# One-hot encoding label
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 2. Bangun MLP
model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])

# 3. Compile & train
model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])
history = model.fit(x_train, y_train, epochs=5, batch_size=32,
validation_split=0.1)

# 4. Evaluasi
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

Tahap Berikutnya

- **Tahap 2 (CNN)**: Bandingkan hasil CNN dengan MLP, lihat bagaimana konvolusi menangkap fitur spasial.
- **Tahap 3 (API)**: Bungkus model ke Flask, kirim gambar untuk prediksi.
- **Tahap 4 (React)**: Buat UI interaktif untuk menggambar angka.
- **Tahap 5 (Visualisasi)**: Tambahkan heatmap/preview untuk interpretasi.

👉 Mau saya buatkan **diagram alur MLP untuk MNIST** (input → hidden → output) supaya lebih visual untuk siswa, atau langsung lanjut ke Tahap 2 (CNN)?
Mantap Edy, kita masuk ke **Tahap 2: CNN**. Di sini tujuan utamanya adalah **membandingkan hasil CNN dengan MLP** dan melihat bagaimana **konvolusi menangkap fitur spasial** dari gambar MNIST.

Konsep CNN vs MLP

- **MLP (Multilayer Perceptron)**:
  - Input gambar 28×28 di-*flatten* jadi vektor 784.
  - Tidak mempertahankan struktur spasial (posisi piksel).
  - Cenderung kurang optimal untuk data gambar.
- **CNN (Convolutional Neural Network)**:
  - Menggunakan **filter (kernel)** yang bergerak di atas gambar.
  - Menangkap pola lokal (tepi, sudut, tekstur).
  - Pooling layer mereduksi dimensi sambil mempertahankan fitur penting.

- Lebih efisien dan akurat untuk data visual.

💻 Contoh Kode CNN untuk MNIST

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# 1. Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 2. Bangun CNN
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation="relu", input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, kernel_size=(3,3), activation="relu"),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])

# 3. Compile & train
model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])
history = model.fit(x_train, y_train, epochs=5, batch_size=32,
validation_split=0.1)

# 4. Evaluasi
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy CNN:", test_acc)
```

📊 Perbandingan Hasil

- **MLP** biasanya mencapai akurasi sekitar **97–98%** pada MNIST.
- **CNN** bisa mencapai **99%+**, karena mampu mengenali pola spasial (misalnya bentuk melengkung angka 3 vs angka 8).
- Visualisasi filter CNN menunjukkan kernel belajar mendeteksi **tepi horizontal/vertikal, lengkungan, dan detail kecil**.

🔜 Tahap Berikutnya

- **Tahap 3 (API Deployment)**: Bungkus model CNN ke Flask, kirim gambar untuk prediksi.
- **Tahap 4 (React)**: Buat UI interaktif untuk menggambar angka.
- **Tahap 5 (Visualisasi)**: Tambahkan heatmap (Grad-CAM) untuk interpretasi.

👉 Mau saya buatkan **diagram perbandingan arsitektur MLP vs CNN** (alur input → hidden → output vs input → conv → pool → dense → output), supaya siswa bisa langsung melihat perbedaan struktur?

Arsitektur CNN yang dibangun

Model ini menangkap pola spasial secara bertahap: dari tepi sederhana hingga bentuk angka yang lebih kompleks. Berikut penjelasan tiap blok, aliran bentuk tensor, dan prinsip di baliknya.

Alur bentuk tensor dari input ke output

- **Input gambar:**
  - Bentuk: 28×28×1 (grayscale)
- **Conv2D (32 filter, 3×3, ReLU):**
  - Bentuk keluar (tanpa padding, stride=1): 26×26×32
- **MaxPooling2D (2×2):**
  - Bentuk keluar: 13×13×32
- **Conv2D (64 filter, 3×3, ReLU):**
  - Bentuk keluar: 11×11×64
- **MaxPooling2D (2×2):**
  - Bentuk keluar: 5×5×64
- **Flatten:**
  - Bentuk keluar: 1600 (5×5×64)
- **Dense (128, ReLU):**
  - Bentuk keluar: 128
- **Dense (10, Softmax):**
  - Bentuk keluar: 10 (probabilitas kelas 0–9)

Catatan: Dengan `padding='same'`, ukuran H×W tidak berkurang di layer konvolusi. Kode di atas default-nya `valid` (mengecil).

Prinsip setiap komponen

- **Conv2D: ekstraksi fitur lokal**
  - **Intuisi:** Filter 3×3 "menyapu" gambar, bereaksi pada pola kecil seperti tepi, sudut, lengkung.
  - **Parameter yang dipelajari:** Untuk setiap filter, bobot 3×3×channel_in + bias. Dengan 32 filter pada input 1 channel:
    - Total parameter: 32 × (3×3×1 + 1) = 320
  - **Operasi konvolusi (skalar per lokasi):**
    - [y(i,j,f) = \sum_{u=0}^{k-1}\sum_{v=0}^{k-1}\sum_{c=0}^{C-1} W_{u,v,c}^{(f)} \cdot x(i+u, j+v, c) + b^{(f)}]
    - Diikuti aktivasi non-linear ReLU: (\mathrm{ReLU}(z)=\max(0,z)).
- **ReLU: non-linearitas sederhana**
  - **Fungsi:** Menghilangkan nilai negatif, membuat fitur "aktif vs tidak".
  - **Manfaat:** Menghindari gradien menghilang parah dan mempercepat konvergensi.
- **MaxPooling2D: downsampling yang mempertahankan fitur kuat**
  - **Intuisi:** Ambil nilai maksimum dalam jendela 2×2; menyaring noise, memperkecil resolusi.

    - **Manfaat:** Mengurangi jumlah parameter di tahap berikutnya, memperluas receptive field efektif, dan membuat fitur lebih translasi-invariant.
- **Lapisan kedua Conv2D (64 filter): memperkaya hierarki fitur**
    - **Peran:** Belajar kombinasi fitur level-1 (tepi) menjadi motif level-2 (lengkung angka, sambungan).
    - **Parameter:** 64 × (3×3×32 + 1) = 18,496
- **Flatten: ubah peta fitur menjadi vektor**
    - **Peran:** Menyiapkan masukan untuk lapisan dense yang melakukan klasifikasi berdasarkan fitur yang telah diekstraksi.
- **Dense 128 (ReLU): kombinasi fitur global**
    - **Intuisi:** Menggabungkan seluruh fitur yang "diringkas" agar keputusan kelas makin tajam.
    - **Parameter:** 1600 × 128 + 128 = 204,928
- **Dense 10 (Softmax): probabilitas kelas**
    - **Softmax:**
        - [p_k = \frac{e^{z_k}}{\sum_{j=1}^{10} e^{z_j}}]
    - **Makna:** Menghasilkan distribusi probabilitas untuk 10 digit. Digunakan dengan loss categorical crossentropy.

Kenapa CNN mengungguli MLP untuk gambar

- **Preservasi spasial:** CNN menjaga struktur H×W, sehingga posisi relatif piksel ikut membentuk fitur.
- **Parameter efisien:** Filter kecil yang berbagi bobot di seluruh lokasi mengurangi parameter dibanding MLP yang melihat vektor 784 tanpa konteks lokasi.
- **Hierarki fitur:** Layer awal belajar tepi; layer tengah belajar motif; layer akhir merangkum menjadi konsep "digit".

Dampak pilihan hiperparameter

- **Ukuran kernel (3×3):** Cukup untuk menangkap detail lokal tanpa biaya besar; bertumpuk dua kali mendekati efek 5×5 dengan parameter lebih sedikit.
- **Jumlah filter (32→64):** Menambah kapasitas fitur. Terlalu kecil bisa underfit, terlalu besar bisa overfit.
- **Pooling (2×2):** Mengurangi resolusi setengah, menstabilkan fitur dan biaya komputasi.
- **Padding dan stride:**
    - **valid:** mengecilkan ukuran, menekankan fitur tepi tapi mengurangi resolusi cepat.
    - **same:** mempertahankan ukuran, memudahkan desain bertingkat dan heatmap.
- **Aktivasi:** ReLU cepat dan efektif; bisa coba LeakyReLU jika banyak "dead neurons".

Tips praktis untuk MNIST

- **Gunakan padding='same'** di kedua Conv2D agar ukuran tetap, memudahkan visualisasi heatmap dan interpretasi.
- **Tambahkan BatchNormalization** setelah Conv2D untuk stabilitas dan kecepatan training.
- **EarlyStopping** dengan `monitor='val_accuracy'` bisa mencegah overfitting.
- **Data augmentation ringan** (shift/zoom kecil) membantu ketahanan terhadap tulisan yang bergeser.

Versi kode yang lebih siap produksi (opsional)

```
model = Sequential([
    Conv2D(32, (3,3), padding="same", activation="relu",
input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), padding="same", activation="relu"),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])
```

Jika mau, kita bisa menambahkan BatchNormalization dan Dropout, lalu membandingkan akurasi dan waktu latih dengan MLP sebagai bahan diskusi siswa. Mau saya buatkan tabel perbandingan parameter, output shape, dan receptive field per layer?

Oke Edy, kita bisa bikin **simulasi visualisasi tiap tahapan CNN** dengan `cv2`/OpenCV agar siswa bisa *melihat* bagaimana gambar MNIST diproses. Jadi bukan hanya angka akurasi, tapi juga transformasi visual.

📌 Tahapan Visualisasi CNN dengan OpenCV

1. Input gambar (28×28)

- Ambil satu digit dari MNIST.
- Tampilkan dengan `cv2.imshow` atau `matplotlib`.

```
import cv2
import numpy as np
from tensorflow.keras.datasets import mnist

(x_train, y_train), _ = mnist.load_data()
digit = x_train[0]  # ambil contoh angka pertama
cv2.imshow("Input Digit", digit)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

---

2. Convolution (Conv2D)

- Buat filter sederhana (misalnya deteksi tepi horizontal/vertikal).
- Gunakan `cv2.filter2D` untuk mensimulasikan operasi konvolusi.

```
# Filter tepi horizontal
kernel = np.array([[-1, -1, -1],
                   [ 0,  0,  0],
                   [ 1,  1,  1]])

conv_output = cv2.filter2D(digit, -1, kernel)
cv2.imshow("Conv Output", conv_output)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

👉 Prinsip: filter bergerak di atas gambar, menghasilkan peta fitur.

---

3. Aktivasi ReLU

- Terapkan fungsi ReLU: nilai negatif → 0.

```
relu_output = np.maximum(conv_output, 0)
```

```
cv2.imshow("ReLU Output", relu_output.astype(np.uint8))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

👉 Prinsip: hanya fitur yang “aktif” (positif) yang diteruskan.

---

4. MaxPooling (2×2)

- Ambil nilai maksimum dalam blok 2×2.
- Bisa disimulasikan dengan `cv2.resize` + `INTER_NEAREST` atau manual loop.

```
def maxpool(img, size=2):
    h, w = img.shape
    pooled = img.reshape(h//size, size, w//size, size).max(axis=(1,3))
    return pooled

pool_output = maxpool(relu_output, 2)
cv2.imshow("Pooling Output", pool_output.astype(np.uint8))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

👉 Prinsip: mengecilkan resolusi, mempertahankan fitur dominan.

---

5. Flatten

- Ubah hasil pooling jadi vektor 1D.
- Visualisasi: cetak panjang vektor.

```
flat_output = pool_output.flatten()
print("Flatten shape:", flat_output.shape)
```

---

6. Dense Layer

- Simulasi dengan perkalian matriks sederhana.
- Misalnya buat bobot acak untuk 128 neuron.

```
weights = np.random.randn(flat_output.shape[0], 128)
dense_output = np.maximum(flat_output @ weights, 0)  # ReLU
print("Dense output shape:", dense_output.shape)
```

---

7. Softmax Output

- Hitung probabilitas 10 kelas.

```
final_weights = np.random.randn(128, 10)
logits = dense_output @ final_weights
exp_logits = np.exp(logits)
softmax_output = exp_logits / np.sum(exp_logits)
print("Softmax probs:", softmax_output)
```

🎨 Visualisasi untuk Siswa

- **Tahap Input:** gambar asli.
- **Tahap Conv:** hasil filter (tepi, lengkung).
- **Tahap ReLU:** hanya fitur aktif.
- **Tahap Pooling:** gambar mengecil tapi fitur tetap terlihat.
- **Tahap Flatten → Dense → Softmax:** tidak bisa divisualkan langsung, tapi bisa ditunjukkan sebagai grafik bar probabilitas.

👉 Mau saya buatkan **script lengkap** yang otomatis menampilkan jendela berurutan (Input → Conv → ReLU → Pooling → Probabilitas) sehingga siswa bisa klik/enter untuk melihat tahap berikutnya, seperti simulasi interaktif?

Tepat sekali Edy 👍, kamu sudah menangkap inti dari **kernel/filter dalam CNN**. Mari kita uraikan lebih detail:

📌 Kernel/Filter dalam CNN

- **Kernel/filter** adalah matriks kecil (misalnya 3×3 atau 5×5) yang bergerak melintasi gambar.
- Setiap kernel punya **parameter (weight dan bias)** yang di-*train* sama seperti bobot di MLP.
- Jadi, kernel bukan sekadar "alat statis" seperti Sobel atau Laplacian di OpenCV, tapi **belajar sendiri** pola apa yang berguna untuk klasifikasi.

🔄 Proses Training

1. **Inisialisasi awal**
   - Bobot kernel diinisialisasi acak (misalnya distribusi normal kecil).
   - Bias juga acak atau nol.
2. **Forward pass**
   - Kernel melakukan konvolusi dengan input (gambar).
   - Hasilnya adalah *feature map*.
3. **Loss dihitung**
   - Output CNN dibandingkan dengan label (misalnya angka "5").
   - Loss function (categorical crossentropy) memberi tahu seberapa jauh prediksi dari target.
4. **Backward pass (Backpropagation)**
   - Gradien dihitung terhadap setiap weight kernel.
   - Prinsip sama dengan MLP: turunan loss terhadap bobot.
5. **Update bobot (SGD/Adam)**
   - Kernel diperbarui sedikit demi sedikit setiap batch/epoch.
   - Setelah banyak iterasi, kernel "belajar" mendeteksi pola yang relevan (tepi, lengkung, tekstur).

🎨 Analogi

Bayangkan kernel seperti **kacamata dengan pola tertentu**:

- Awalnya kacamatanya acak, jadi melihat gambar agak "buram".
- Setelah banyak latihan, kacamatanya menyesuaikan sehingga bisa fokus pada pola penting (misalnya lengkungan angka 3, garis lurus angka 1).
- CNN punya banyak kacamata (filter), masing-masing belajar pola berbeda.

📊 Perbedaan dengan MLP

- **MLP:** setiap neuron punya bobot untuk semua piksel → jumlah parameter besar.
- **CNN:** kernel kecil berbagi bobot di seluruh lokasi → parameter jauh lebih sedikit, tapi tetap bisa menangkap pola spasial.

👉 Jadi jawaban singkat: **Ya, kernel/filter itu memang bobot dan bias yang dilatih. Awalnya acak, lalu melalui epoch training, mereka menyesuaikan hingga menemukan pola terbaik untuk meminimalkan loss.**

Mau saya tunjukkan **visualisasi evolusi kernel** (misalnya filter acak → setelah 1 epoch → setelah 5 epoch) dengan simulasi sederhana di OpenCV, supaya siswa bisa melihat bagaimana "kacamata" CNN berubah dari noise menjadi pola tepi/lengkung?
Baik Edy, mari kita buat **simulasi evolusi kernel CNN** dengan OpenCV agar siswa bisa melihat bagaimana "kacamata" CNN berubah dari awal acak → setelah 1 epoch → setelah 5 epoch.
Karena kita tidak akan benar-benar melatih CNN penuh di sini, kita bisa **mensimulasikan** dengan tiga tahap visualisasi:

📌 Konsep Simulasi

1. **Kernel acak (awal training)**
   - Nilai bobot acak → terlihat seperti noise.
2. **Kernel setelah 1 epoch**
   - Mulai membentuk pola sederhana (misalnya garis horizontal/vertikal samar).
3. **Kernel setelah 5 epoch**
   - Lebih jelas: pola tepi, lengkung, atau tekstur yang relevan untuk digit.

💻 Contoh Kode Simulasi dengan OpenCV

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def show_kernel(kernel, title):
    plt.imshow(kernel, cmap='gray')
    plt.title(title)
    plt.axis('off')
    plt.show()

# 1. Kernel acak (awal training)
kernel_random = np.random.randn(3,3)
show_kernel(kernel_random, "Kernel Acak (Epoch 0)")

# 2. Kernel setelah 1 epoch (simulasi: pola garis samar)
kernel_epoch1 = np.array([[-1, -1, -1],
                          [ 0,  0,  0],
                          [ 1,  1,  1]])
show_kernel(kernel_epoch1, "Kernel Setelah 1 Epoch (Deteksi Tepi Horizontal)")

# 3. Kernel setelah 5 epoch (simulasi: pola lengkung/tepi kompleks)
kernel_epoch5 = np.array([[ 0, -1,  0],
                          [-1,  4, -1],
                          [ 0, -1,  0]])
show_kernel(kernel_epoch5, "Kernel Setelah 5 Epoch (Deteksi Lengkung/Corner)")
```

📊 Penjelasan Visual

- **Epoch 0 (acak):** kernel terlihat seperti noise, tidak punya pola jelas.
- **Epoch 1:** kernel mulai menyerupai filter tepi sederhana (misalnya horizontal).
- **Epoch 5:** kernel lebih kompleks, bisa mendeteksi lengkung atau sudut, cocok untuk membedakan angka.

🎨 Demo pada Gambar MNIST

Kita bisa menerapkan kernel ini ke digit MNIST dengan `cv2.filter2D`:

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), _ = mnist.load_data()
digit = x_train[0]  # ambil contoh angka

# Terapkan kernel acak, epoch1, epoch5
conv0 = cv2.filter2D(digit, -1, kernel_random)
conv1 = cv2.filter2D(digit, -1, kernel_epoch1)
conv5 = cv2.filter2D(digit, -1, kernel_epoch5)

# Tampilkan hasil
titles = ["Input Digit", "Conv Epoch 0 (Acak)", "Conv Epoch 1", "Conv Epoch
5"]
images = [digit, conv0, conv1, conv5]

for t, img in zip(titles, images):
    plt.imshow(img, cmap='gray')
    plt.title(t)
    plt.axis('off')
    plt.show()
```

👉 Dengan ini siswa bisa melihat: **digit asli → hasil filter acak → hasil filter setelah belajar → hasil filter matang**.

Apakah kamu mau saya buatkan **diagram alur evolusi kernel** (noise → garis → lengkung) sebagai ilustrasi tambahan, supaya lebih intuitif sebelum mereka melihat hasil kode?

Pipeline sederhana prediksi wajah siswa dengan Flask, Keras/TensorFlow, SQLite, dan React Vite

Mantap, Edy 🚀. Aku akan kasih contoh **minimal setup**: frontend ReactJS (Vite) untuk ambil kamera, lalu backend Flask API dengan OpenCV (`cv2`) untuk face detection & crop. Jadi alurnya:

1. **ReactJS (Vite)** → ambil stream kamera (laptop/HP).
2. Capture frame → kirim ke Flask API.
3. **Flask API** → pakai OpenCV Haar Cascade untuk deteksi wajah → crop → return hasil.

📌 Frontend (ReactJS + Vite)

File: `src/App.jsx`

```
import { useRef, useEffect, useState } from "react";

function App() {
  const videoRef = useRef(null);
  const canvasRef = useRef(null);
  const [faceImage, setFaceImage] = useState(null);

  useEffect(() => {
    // akses kamera
    navigator.mediaDevices.getUserMedia({ video: true })
      .then(stream => {
        videoRef.current.srcObject = stream;
      })
      .catch(err => console.error("Camera error:", err));
  }, []);

  const captureFrame = async () => {
    const canvas = canvasRef.current;
    const video = videoRef.current;
    canvas.width = video.videoWidth;
    canvas.height = video.videoHeight;
    const ctx = canvas.getContext("2d");
    ctx.drawImage(video, 0, 0, canvas.width, canvas.height);

    const dataUrl = canvas.toDataURL("image/jpeg");
    const blob = await (await fetch(dataUrl)).blob();

    // kirim ke Flask API
    const formData = new FormData();
    formData.append("file", blob, "frame.jpg");

    const res = await fetch("http://127.0.0.1:5000/detect", {
      method: "POST",
      body: formData,
    });

    const resultBlob = await res.blob();
    setFaceImage(URL.createObjectURL(resultBlob));
  };

  return (
    <div>
      <h1>Face Detection Demo</h1>
      <video ref={videoRef} autoPlay playsInline style={{ width: "400px" }} />
      <canvas ref={canvasRef} style={{ display: "none" }} />
      <button onClick={captureFrame}>Capture & Detect</button>

      {faceImage && (
        <div>
          <h2>Detected Face:</h2>
          <img src={faceImage} alt="face" />
        </div>
      )}
    </div>
  );
}
```

```
export default App;
```

📌 Backend (Flask + OpenCV)

File: `app.py`

```
from flask import Flask, request, send_file
import cv2
import numpy as np
import io

app = Flask(__name__)

# Haar Cascade untuk face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")

@app.route("/detect", methods=["POST"])
def detect():
    file = request.files["file"]
    img_bytes = file.read()
    npimg = np.frombuffer(img_bytes, np.uint8)
    img = cv2.imdecode(npimg, cv2.IMREAD_COLOR)

    # Deteksi wajah
    faces = face_cascade.detectMultiScale(img, scaleFactor=1.1,
minNeighbors=5)

    if len(faces) > 0:
        x, y, w, h = faces[0]  # ambil wajah pertama
        face_crop = img[y:y+h, x:x+w]

        # convert ke JPEG
        _, buffer = cv2.imencode(".jpg", face_crop)
        return send_file(io.BytesIO(buffer), mimetype="image/jpeg")

    return "No face detected", 400

if __name__ == "__main__":
    app.run(debug=True)
```

⚙️ Cara Jalankan

1. **Backend**
2. `pip install flask opencv-python`
3. `python app.py`
   → API jalan di `http://127.0.0.1:5000/detect`
4. **Frontend (ReactJS + Vite)**
5. `npm create vite@latest face-detect-demo`
6. `cd face-detect-demo`
7. `npm install`
8. `npm install`
9. `npm run dev`

🎯 Dengan ini kamu sudah punya pipeline sederhana: kamera → ReactJS → Flask API → OpenCV crop wajah.

Langkah selanjutnya bisa kamu extend ke fitur lain (misalnya simpan ke DB, face recognition, atau integrasi ke signage system).
Mau aku tambahkan **flowchart visual** supaya lebih gampang kamu jelaskan ke siswa/anak tentang alur data kamera → API → hasil?

Kita bangun alur end-to-end: backend Flask untuk API dan training, SQLite untuk metadata siswa, penyimpanan gambar di filesystem, Keras/TensorFlow untuk ekstraksi fitur dan klasifikasi, lalu React Vite untuk upload/prediksi. Fokus pada sederhana, bisa ditingkatkan nanti.

Arsitektur komponen

- **Backend Flask:** Endpoint untuk CRUD siswa, upload gambar, trigger training, prediksi, evaluasi.
- **SQLite:** Tabel siswa dan gambar. Simpel, mudah dipakai.
- **Penyimpanan file:** Folder per siswa untuk gambar mentah dan dataset.
- **ML (TensorFlow/Keras):** Ekstraksi fitur dengan MobileNetV2 (pretrained) + classifier dense. Alternatif: fine-tuning ringan.
- **Frontend React (Vite):** Form upload multi-file, daftar siswa, tombol train/predict/evaluate.

Tahap 1. Membangun data

Skema SQLite dan struktur direktori

```
-- schema.sql
CREATE TABLE IF NOT EXISTS students (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS images (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  student_id INTEGER NOT NULL,
  path TEXT NOT NULL,
  FOREIGN KEY(student_id) REFERENCES students(id)
);
```

Struktur folder:

```
data/
  raw/
    <student_id>/
      img_001.jpg
      ...
  processed/
    train/
    val/
models/
  face_classifier.h5
```

Flask: inisialisasi app dan DB

```
# app.py
import os, sqlite3
from flask import Flask, request, jsonify
from werkzeug.utils import secure_filename

app = Flask(__name__)
```

```
app.config['DATA_DIR'] = 'data'
os.makedirs(os.path.join(app.config['DATA_DIR'], 'raw'), exist_ok=True)
os.makedirs('models', exist_ok=True)

def get_db():
    conn = sqlite3.connect('students.db')
    conn.row_factory = sqlite3.Row
    return conn

@app.post('/students')
def create_student():
    name = request.json.get('name')
    if not name: return jsonify({'error':'name required'}), 400
    db = get_db()
    cur = db.cursor()
    cur.execute('INSERT INTO students(name) VALUES (?)', (name,))
    db.commit()
    sid = cur.lastrowid
    os.makedirs(os.path.join(app.config['DATA_DIR'], 'raw', str(sid)),
exist_ok=True)
    return jsonify({'id': sid, 'name': name})

@app.get('/students')
def list_students():
    db = get_db()
    rows = db.execute('SELECT id, name FROM students').fetchall()
    return jsonify([dict(r) for r in rows])

@app.post('/upload/<int:student_id>')
def upload_images(student_id):
    files = request.files.getlist('files')
    if not files or len(files) < 5:
        return jsonify({'error':'minimal 5 gambar per siswa'}), 400
    db = get_db()
    r = db.execute('SELECT id FROM students WHERE id=?',
(student_id,)).fetchone()
    if not r: return jsonify({'error':'student not found'}), 404
    saved = []
    base = os.path.join(app.config['DATA_DIR'], 'raw', str(student_id))
    for f in files:
        fname = secure_filename(f.filename)
        path = os.path.join(base, fname)
        f.save(path)
        db.execute('INSERT INTO images(student_id, path) VALUES (?,?)',
(student_id, path))
        saved.append(path)
    db.commit()
    return jsonify({'saved': saved})
```

Tip:
Pastikan gambar wajah cukup terang dan frontal. Konsistensi akan sangat membantu akurasi awal.

Tahap 2. Training data dan simpan model

Pipeline ML: preprocessing, feature extractor, classifier

- **Preprocessing:** Resize ke 224×224, normalisasi sesuai MobileNetV2.

- **Feature extractor:** MobileNetV2 tanpa top, weights “imagenet”.
- **Classifier:** Dense dengan softmax untuk jumlah siswa saat ini.
- **Split:** Train/validation sederhana 80/20 per siswa.

```
# train.py (bisa digabung di app.py untuk endpoint /train)
import tensorflow as tf
import numpy as np, random, os, sqlite3
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras import layers, models

IMG_SIZE = (224, 224)

def load_dataset():
    conn = sqlite3.connect('students.db'); conn.row_factory = sqlite3.Row
    students = conn.execute('SELECT id, name FROM students').fetchall()
    id_to_idx = {s['id']: i for i,s in enumerate(students)}
    X, y = [], []
    for s in students:
        imgs = conn.execute('SELECT path FROM images WHERE student_id=?',
(s['id'],)).fetchall()
        for row in imgs:
            X.append(row['path']); y.append(id_to_idx[s['id']])
    conn.close()
    # Shuffle
    idx = list(range(len(X))); random.shuffle(idx)
    X = [X[i] for i in idx]; y = [y[i] for i in idx]
    # Split
    split = int(0.8 * len(X))
    return (X[:split], y[:split]), (X[split:], y[split:]), students, id_to_idx

def load_and_preprocess(path):
    img = tf.keras.utils.load_img(path, target_size=IMG_SIZE)
    arr = tf.keras.utils.img_to_array(img)
    return preprocess_input(arr)

def make_dataset(paths, labels, batch=16):
    ds = tf.data.Dataset.from_tensor_slices((paths, labels))
    def _map(p, l):
        img = tf.numpy_function(load_and_preprocess, [p], tf.float32)
        img.set_shape((*IMG_SIZE, 3))
        return img, tf.one_hot(l, depth=num_classes)
    ds = ds.map(_map, num_parallel_calls=tf.data.AUTOTUNE)
    return ds.shuffle(512).batch(batch).prefetch(tf.data.AUTOTUNE)

def build_model(num_classes):
    base = MobileNetV2(include_top=False, weights='imagenet',
input_shape=(*IMG_SIZE,3), pooling='avg')
    base.trainable = False  # freeze untuk sederhana
    x = layers.Dropout(0.2)(base.output)
    out = layers.Dense(num_classes, activation='softmax')(x)
    model = models.Model(base.input, out)
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
# Endpoint training
@app.post('/train')
def train_endpoint():
    (Xtr, ytr), (Xval, yval), students, id_to_idx = load_dataset()
    global num_classes
    num_classes = len(students)
    if num_classes < 2:
        return jsonify({'error':'butuh minimal 2 siswa untuk klasifikasi'}),
400
    train_ds = make_dataset(Xtr, ytr, batch=16)
    val_ds = make_dataset(Xval, yval, batch=16)
    model = build_model(num_classes)
    history = model.fit(train_ds, validation_data=val_ds, epochs=10)
    model.save('models/face_classifier.h5')
    # Simpan label map
    label_map = {v:k for k,v in id_to_idx.items()}
    with open('models/labels.json','w') as f:
        import json; json.dump(label_map, f)
    return jsonify({'epochs':10, 'train_acc':
float(history.history['accuracy'][-1]),
                    'val_acc': float(history.history['val_accuracy'][-1]),
                    'classes': [{ 'id': s['id'], 'name': s['name']} for s in
students]})
```

Catatan:

- Bisa tambahkan fine-tuning ringan (unfreeze beberapa blok terakhir) jika data cukup.
- Pertimbangkan data augmentation (flip horizontal, brightness, zoom) untuk robust.

Tahap 3. Prediksi data

Endpoint prediksi

- Input: satu gambar baru (file upload).
- Output: student_id, nama, probabilitas, top-3.

```
import json
from tensorflow.keras.models import load_model

def load_model_and_labels():
    model = load_model('models/face_classifier.h5')
    with open('models/labels.json') as f: label_map = json.load(f)  # idx ->
student_id
    # Buat idx->name
    conn = sqlite3.connect('students.db'); conn.row_factory = sqlite3.Row
    names = {s['id']: s['name'] for s in conn.execute('SELECT id,name FROM
students')}
    conn.close()
    idx_to_name = {i: names[int(sid)] for i,sid in label_map.items()}
    idx_to_id = {i: int(sid) for i,sid in label_map.items()}
    return model, idx_to_name, idx_to_id

@app.post('/predict')
def predict():
    if 'file' not in request.files:
        return jsonify({'error':'file required'}), 400
    file = request.files['file']
    path = os.path.join('data','temp',secure_filename(file.filename))
    os.makedirs(os.path.dirname(path), exist_ok=True)
```

```
    file.save(path)
    img = tf.keras.utils.load_img(path, target_size=IMG_SIZE)
    arr = tf.keras.utils.img_to_array(img)
    arr = preprocess_input(arr)
    logits = model.predict(arr[np.newaxis, ...])[0]
    topk = np.argsort(logits)[::-1][:3]
    result = []
    for i in topk:
        result.append({
            'student_id': idx_to_id[i],
            'name': idx_to_name[i],
            'prob': float(logits[i])
        })
    return jsonify({'top': result})
```

Tahap 4. Evaluasi dan perbaikan jika prediksi salah

Endpoint evaluasi batch dan confusion matrix

- Kirim kumpulan gambar berlabel untuk menghitung akurasi dan confusion matrix.
- Simpan kesalahan untuk retraining bertahap.

```
from sklearn.metrics import confusion_matrix, classification_report

@app.post('/evaluate')
def evaluate():
    # Body: JSON { items: [ {path: "...", student_id: 1}, ... ] }
    items = request.json.get('items', [])
    y_true, y_pred = [], []
    for it in items:
        path = it['path']
        img = tf.keras.utils.load_img(path, target_size=IMG_SIZE)
        arr = preprocess_input(tf.keras.utils.img_to_array(img))
        logits = model.predict(arr[np.newaxis, ...])[0]
        pred_idx = int(np.argmax(logits))
        y_true.append(idx_to_id_inv[it['student_id']])  # map student_id ->
idx
        y_pred.append(pred_idx)
    cm = confusion_matrix(y_true, y_pred).tolist()
    report = classification_report(y_true, y_pred, output_dict=True)
    return jsonify({'confusion_matrix': cm, 'report': report})
```

Strategi perbaikan jika salah prediksi:

- **Tambah data sulit:** Unggah gambar dari sudut, pencahayaan berbeda, aksesoris (kacamata), ekspresi beragam.
- **Augmentasi:** Aktifkan RandomFlip, RandomBrightness, RandomZoom di pipeline train.
- **Fine-tuning:** Unfreeze blok terakhir MobileNetV2, turunkan learning rate.
- **Deteksi wajah:** Tambahkan face detection (MTCNN/Haar Cascades) sebelum resize untuk fokus area wajah.
- **Class weights:** Jika jumlah gambar per siswa tidak seimbang, gunakan class_weight saat training.

Contoh augmentasi dan fine-tuning singkat:

```
data_aug = tf.keras.Sequential([
    layers.RandomFlip('horizontal'),
    layers.RandomBrightness(0.2),
    layers.RandomZoom(0.1),
])
```

```
def make_dataset(paths, labels, batch=16):
    ds = tf.data.Dataset.from_tensor_slices((paths, labels))
    def _map(p, l):
        img = tf.numpy_function(load_and_preprocess, [p], tf.float32)
        img.set_shape((*IMG_SIZE, 3))
        img = data_aug(img)
        return img, tf.one_hot(l, depth=num_classes)
    return ds.shuffle(512).batch(batch).prefetch(tf.data.AUTOTUNE)

# Fine-tuning ringan
base.trainable = True
for layer in base.layers[:-20]:
    layer.trainable = False
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Frontend React (Vite) ringkas

Upload multi-file dan aksi API

```
npm create vite@latest face-app -- --template react
cd face-app && npm i
npm i axios
// src/App.jsx
import { useState } from 'react';
import axios from 'axios';

export default function App() {
  const [name, setName] = useState('');
  const [studentId, setStudentId] = useState(null);
  const [files, setFiles] = useState([]);

  const createStudent = async () => {
    const { data } = await axios.post('/students', { name });
    setStudentId(data.id);
  };

  const uploadImages = async () => {
    const form = new FormData();
    for (const f of files) form.append('files', f);
    await axios.post(`/upload/${studentId}`, form);
    alert('Uploaded');
  };

  const train = async () => {
    const { data } = await axios.post('/train');
    alert(`val_acc: ${data.val_acc.toFixed(3)}`);
  };

  const predict = async (file) => {
    const form = new FormData();
    form.append('file', file);
    const { data } = await axios.post('/predict', form);
    console.log(data.top);
```

```
    alert(`Prediksi: ${data.top[0].name}
(${(data.top[0].prob*100).toFixed(1)}%)`);
  };

  return (
    <div style={{ padding: 20 }}>
      <h3>Tambah siswa</h3>
      <input value={name} onChange={e=>setName(e.target.value)}
placeholder="Nama siswa" />
      <button onClick={createStudent}>Simpan</button>
      <hr/>
      <h3>Upload gambar (≥5)</h3>
      <input type="file" multiple
onChange={e=>setFiles([...e.target.files])}/>
      <button onClick={uploadImages} disabled={!studentId}>Upload</button>
      <hr/>
      <button onClick={train}>Train</button>
      <hr/>
      <h3>Prediksi</h3>
      <input type="file" onChange={e=>predict(e.target.files[0])}/>
    </div>
  );
}
```

Konfigurasi proxy dev Vite ke Flask:

```
// vite.config.js
export default {
  server: {
    proxy: {
      '/': 'http://localhost:5000'
    }
  }
}
```

Rekomendasi praktis dan next steps

- **Kualitas dataset:** Usahakan ≥20 gambar per siswa untuk hasil lebih stabil. Minimal 5 boleh, tapi akurasi fluktuatif.
- **Face detection dulu:** Crop wajah sebelum masuk model untuk mengurangi noise background.
- **Persistensi versi model:** Simpan versi dan metrik di SQLite (tabel models) agar bisa bandingkan hasil antar training.
- **Batch evaluation dari frontend:** Tambahkan halaman evaluasi yang menampilkan confusion matrix heatmap agar siswa bisa melihat kesalahan umum.
- **Privasi dan etika:** Jelaskan ke siswa tujuan pembelajaran, minta persetujuan, dan simpan data secara aman.

Kalau Edy mau, saya bisa tambah:

- Endpoint face detection otomatis dan crop.
- Diagram alur (sequence + flowchart) untuk bahan ajar.
- Script Bash untuk bootstrap server dan service systemd.

Baik, kita tambahkan **endpoint face detection otomatis dan crop** supaya gambar yang masuk ke pipeline training/prediksi hanya berisi wajah, bukan background. Ini akan meningkatkan akurasi model.

Tahap: Face Detection & Crop

Library yang digunakan

- **OpenCV (cv2)**: untuk deteksi wajah dengan Haar Cascade atau DNN.
- Alternatif lebih akurat: **MTCNN** (`pip install mtcnn`).

Untuk sederhana, kita pakai Haar Cascade bawaan OpenCV.

File Haar Cascade

OpenCV sudah menyediakan file XML:
`haarcascade_frontalface_default.xml`
Biasanya ada di `cv2.data.haarcascades`.

Endpoint Flask: `/detect_crop`

```
import cv2
import numpy as np

# Load Haar Cascade sekali di awal
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

@app.post('/detect_crop')
def detect_crop():
    if 'file' not in request.files:
        return jsonify({'error':'file required'}), 400
    file = request.files['file']
    fname = secure_filename(file.filename)
    path = os.path.join('data','temp',fname)
    os.makedirs(os.path.dirname(path), exist_ok=True)
    file.save(path)

    # Baca gambar
    img = cv2.imread(path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Deteksi wajah
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5,
                                          minSize=(50,50))
    if len(faces) == 0:
        return jsonify({'error':'no face detected'}), 404

    crops = []
    for i,(x,y,w,h) in enumerate(faces):
        face_img = img[y:y+h, x:x+w]
        crop_path = os.path.join('data','temp', f'crop_{i}_{fname}')
        cv2.imwrite(crop_path, face_img)
        crops.append(crop_path)

    return jsonify({'faces_detected': len(faces), 'crops': crops})
```

Integrasi ke Upload & Training

- **Upload:** sebelum simpan ke DB, jalankan deteksi crop. Simpan hasil crop, bukan gambar penuh.
- **Training:** dataset otomatis lebih bersih karena hanya wajah.
- **Prediksi:** sebelum masuk ke model, jalankan crop → resize → preprocess.

Contoh integrasi di `/upload/<student_id>`:

```
@app.post('/upload/<int:student_id>')
def upload_images(student_id):
    files = request.files.getlist('files')
    if not files or len(files) < 5:
        return jsonify({'error':'minimal 5 gambar per siswa'}), 400
    db = get_db()
    r = db.execute('SELECT id FROM students WHERE id=?',
(student_id,)).fetchone()
    if not r: return jsonify({'error':'student not found'}), 404
    saved = []
    base = os.path.join(app.config['DATA_DIR'], 'raw', str(student_id))
    os.makedirs(base, exist_ok=True)

    for f in files:
        fname = secure_filename(f.filename)
        path = os.path.join(base, fname)
        f.save(path)

        # Deteksi wajah
        img = cv2.imread(path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.1, 5)
        if len(faces) > 0:
            x,y,w,h = faces[0]  # ambil wajah pertama
            face_img = img[y:y+h, x:x+w]
            crop_path = os.path.join(base, f'crop_{fname}')
            cv2.imwrite(crop_path, face_img)
            db.execute('INSERT INTO images(student_id, path) VALUES (?,?)',
(student_id, crop_path))
            saved.append(crop_path)
    db.commit()
    return jsonify({'saved': saved})
```

Tips Perbaikan

- **Multi-face:** jika ada lebih dari satu wajah, bisa simpan semua crop atau hanya wajah terbesar.
- **MTCNN:** lebih akurat untuk wajah miring/bercahaya.
- **Resize otomatis:** setelah crop, langsung resize ke 224×224 agar konsisten dengan model.
- **Augmentasi:** tambahkan pipeline augmentasi setelah crop untuk memperkuat dataset.

👉 Dengan endpoint ini, setiap gambar siswa otomatis dipotong ke area wajah sebelum masuk ke database dan training. Mau saya tambahkan juga versi **MTCNN** (lebih akurat daripada Haar Cascade) sebagai opsi alternatif?

Gambaran umum

Kamu sudah paham GD untuk MLP/CNN. Intinya tetap sama: kita menentukan representasi input yang tepat (fitur), memilih arsitektur yang sesuai, mendefinisikan loss yang relevan, lalu mengoptimalkan parameter dengan GD/SGD. Perbedaannya ada di "bagaimana data diwakili" dan "bentuk model" untuk audio dan teks.

Data suara: pipeline dari waveform ke prediksi

Representasi dan pra-pemrosesan

- **Waveform mentah:** Array amplitudo vs waktu; bisa langsung ke 1D-CNN atau model khusus audio.
- **Spektrogram/STFT:** Ubah sinyal waktu ke domain frekuensi-waktu; memudahkan CNN 2D seperti gambar.
- **Mel-spectrogram/MFCC:** Skala mel meniru persepsi manusia; MFCC umum untuk pengenalan ucapan/identitas speaker.
- **Normalisasi:**
  - **Standardisasi energi:** Mengatur loudness.
  - **Per-frame normalization:** Menjaga stabilitas fitur antar waktu.
- **Augmentasi:**
  - **Time shift / time stretch:** Menggeser atau merenggangkan tempo.
  - **Pitch shift:** Mengubah nada.
  - **SpecAugment:** Masking bagian waktu/frekuensi di spektrogram.

Arsitektur model yang cocok

- **1D-CNN untuk waveform:**
  - **Kelebihan:** Menangkap pola lokal (onset, transien) langsung dari sinyal.
  - **Catatan:** Butuh data besar; sensitif terhadap noise.
- **2D-CNN untuk spektrogram:**
  - **Kelebihan:** Memanfaatkan intuisi "gambar"; filter menangkap pola harmonik/format.
  - **Catatan:** Kualitas bergantung pada parameter STFT (window, hop).
- **RNN/LSTM/GRU di atas fitur:**
  - **Kelebihan:** Menangkap dependensi waktu panjang (frase).
  - **Catatan:** Lebih lambat, bisa diganti dengan transformer.
- **Transformer audio:**
  - **Kelebihan:** Attention untuk konteks panjang; kuat untuk ASR dan event detection.
  - **Catatan:** Perlu regularisasi dan batching hati-hati.

Tugas, label, dan loss

- **Klasifikasi suara (jenis suara, emosi, speaker):**
  - **Loss:** Cross-entropy.
  - **Output:** Softmax.
- **Deteksi peristiwa audio (kapan bunyi terjadi):**
  - **Loss:** Binary cross-entropy per frame; bisa pakai CTC jika urutan tidak selaras.
- **ASR (speech-to-text):**
  - **Loss:** CTC untuk alignment tidak eksplisit, atau seq2seq dengan attention/transformer.
  - **Output:** Urutan token.

Visualisasi untuk murid

- **Plot spektrogram:** Tunjukkan perubahan pola saat kata berbeda diucapkan.
- **Filter CNN sebagai "pendengar lokal":** Visualkan respons filter pada frekuensi tertentu.
- **Loss vs epoch dan contoh prediksi:** Dengarkan audio input, lihat label, bandingkan prediksi.

Data teks: pipeline dari karakter/katakata ke vektor

Representasi dan pra-pemrosesan

- **Tokenisasi:**
  - **Karakter/wordpiece/BPE:** Kompromi antara kosa kata dan generalisasi.
  - **Case folding, stripping punctuation:** Sesuaikan dengan tugas.
- **Vektorisasi:**
  - **One-hot:** Sederhana, sangat sparse.
  - **Embedding terlatih:** Word2Vec/GloVe (statik).
  - **Contextual embeddings:** Dari transformer (mis. per token).
- **Padding/truncation & masking:**
  - **Menjaga batch seragam:** Masking penting untuk RNN/transformer.
  - **Panjang maksimum:** Disesuaikan agar tidak meledak memori.
- **Augmentasi teks:**
  - **Synonym replacement, back-translation:** Hati-hati menjaga makna.
  - **Noising ringan:** Dropout token.

Arsitektur model yang cocok

- **MLP di atas fitur agregat:**
  - **Kelebihan:** Cepat untuk "bag-of-words" atau rata-rata embedding.
  - **Catatan:** Kehilangan urutan/konteks.
- **1D-CNN untuk teks:**
  - **Kelebihan:** Menangkap n-gram sebagai pola lokal; efektif untuk klasifikasi.
  - **Catatan:** Konteks panjang terbatas.
- **RNN/LSTM/GRU:**
  - **Kelebihan:** Urutan terjaga; baik untuk tagging, generasi sederhana.
  - **Catatan:** Sulit paralel, vanishing/exploding handled dengan gating.
- **Transformer (encoder/decoder):**
  - **Kelebihan:** Attention untuk konteks panjang; SOTA untuk hampir semua tugas NLP.
  - **Catatan:** Butuh mask, posisi (positional encoding), dan batching terencana.

Tugas, label, dan loss

- **Klasifikasi teks (sentimen, topik):**
  - **Loss:** Cross-entropy.
  - **Output:** Softmax.
- **Sequence labeling (POS, NER):**
  - **Loss:** Token-level cross-entropy; kadang CRF di atasnya.
  - **Output:** Label per token.
- **Sequence-to-sequence (terjemahan, ringkasan):**
  - **Loss:** Cross-entropy per langkah; teacher forcing saat training.
  - **Output:** Urutan token; perlu decoding (greedy/beam).

Visualisasi untuk murid

- **Attention heatmap:** Perlihatkan kata mana yang “diperhatikan”.
- **t-SNE/UMAP embedding:** Tunjukkan pengelompokan semantik.
- **Kurva loss dan contoh misclass:** Kaitkan kesalahan dengan kalimat ambigu.

Bagaimana GD/SGD bekerja di audio dan teks

- **Inti tetap sama:**
  - (\textbf{Forward}) menghitung prediksi dari input terwakili (spektrogram/embedding).
  - (\textbf{Loss}) membandingkan prediksi vs label.
  - (\textbf{Backward}) menghitung turunan (\nabla_{\theta} \mathcal{L}).
  - (\textbf{Update}) parameter: (\theta \leftarrow \theta - \eta \cdot \nabla_{\theta} \mathcal{L}).
- **Batching dan panjang urutan:**
  - **Audio/teks bervariasi panjangnya:** Gunakan padding + masking; atau bucketing per panjang agar stabil.
  - **CTC/seq2seq:** Gradien melewati waktu; perhatikan stabilitas dengan clipping dan proper initialization.
- **Regularisasi:**
  - **Dropout, weight decay:** Mencegah overfitting.
  - **SpecAugment pada audio, token dropout pada teks:** Augmentasi yang membantu generalisasi.

Tips praktis dan materi interaktif untuk siswa

Audio mini-lab

- **Tujuan:** Klasifikasi “kata sederhana” (mis. angka 0–9 berbahasa Indonesia).
- **Langkah:**
  - **Ekstrak mel-spectrogram:** Parameter window/hop jadi eksperimen.
  - **Model CNN 2D kecil:** 3–4 blok conv+pool, lalu dense.
  - **Visual:** Tampilkan spektrogram input dan peta aktivasi filter.
  - **Latih dengan SGD:** Bandingkan learning rate, momentum, dan augmentasi.

Teks mini-lab

- **Tujuan:** Klasifikasi sentimen kalimat pendek Bahasa Indonesia.
- **Langkah:**
  - **Tokenisasi wordpiece:** Batasi vocab agar mudah.
  - **Embedding trainable + 1D-CNN:** Filter ukuran 3/4/5 sebagai n-gram.
  - **Visual:** Attention sederhana via CAM/Grad-CAM pada CNN teks (menyorot token penting).
  - **Bandingkan dengan LSTM:** Tunjukkan perbedaan performa dan waktu.

Modulasi untuk Raspberry Pi/kelas

- **Ringankan komputasi:**
  - **Quantization/float16:** Turunkan beban; cocok untuk inference demo.
  - **Batch kecil dan panjang terbatas:** Hindari OOM.
- **Antarmuka edukatif:**
  - **Config-driven:** Pilih “audio” atau “teks” di config.json, tentukan fitur, model, hyperparameter.

- **Grafik otomatis:** Loss, akurasi, contoh salah, heatmap fitur.

Rekomendasi penutup

- **Mulai dari representasi yang jelas:** Mel-spectrogram untuk audio, embedding sederhana untuk teks.
- **Pilih model minimal yang menjelaskan konsep:** CNN kecil untuk audio, CNN/LSTM kecil untuk teks.
- **Tekankan visualisasi:** Bawa murid melihat "apa yang dipelajari" model agar GD terasa konkret.
- **Iterasi terkontrol:** Tuning satu variabel per eksperimen (learning rate, fitur, arsitektur) supaya siswa memahami sebab-akibat.

Kalau kamu mau, aku siapkan contoh kode Keras yang modular untuk dua jalur ini: loader fitur (audio/teks), model builder (CNN/LSTM), trainer dengan callback visualisasi, dan notebook ringkas untuk kelas.

Mantap, Edy 🚀. Karena kamu sudah punya fondasi Python, Keras/TensorFlow, Flask, dan React Vite, kita bisa bikin alur end-to-end untuk proyek **klasifikasi kata sederhana (angka 0–9 berbahasa Indonesia)**. Aku pecah jadi **6 tahap utama** supaya jelas:

---

1. Dataset & Pra-pemrosesan Audio 🎙️

- **Kumpulkan data suara**: rekaman orang mengucapkan angka "nol" sampai "sembilan" dalam Bahasa Indonesia. Bisa rekam sendiri atau cari dataset open-source lalu terjemahkan label.
- **Format standar**: WAV/16kHz mono biasanya cukup.
- **Ekstraksi fitur**:
  - Gunakan **Mel-spectrogram** atau **MFCC** (lebih ringan).
  - Normalisasi per frame agar stabil.
- **Augmentasi**: time-shift, pitch-shift, SpecAugment untuk memperkaya variasi.

---

2. Model Keras/TensorFlow 🧠

- **Input**: tensor 2D (frekuensi × waktu) dari spektrogram.
- **Arsitektur sederhana**:
  - `Conv2D → BatchNorm → ReLU → MaxPool` (ulang beberapa kali).
  - `Flatten → Dense → Softmax(10)` untuk klasifikasi angka 0–9.
- **Loss**: categorical cross-entropy.
- **Optimizer**: Adam atau SGD dengan learning rate kecil.
- **Callback**: EarlyStopping, ModelCheckpoint.

Contoh snippet (ringkas):

```
import tensorflow as tf
from tensorflow.keras import layers, models

def build_model(input_shape, num_classes=10):
    model = models.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
```

```
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

---

3. Training & Evaluasi

- **Split data**: train/val/test.
- **Visualisasi**: plot loss & akurasi per epoch.
- **Confusion matrix**: cek kesalahan antar angka (misalnya "dua" vs "tujuh").

---

4. Deploy via Flask API

- **Simpan model**: `model.save("model.h5")`.
- **Flask server**:
  - Endpoint `/predict` menerima file audio.
  - Preprocessing audio → ekstrak fitur → `model.predict`.
  - Return JSON: `{ "prediksi": "tiga", "prob": 0.92 }`.

Contoh ringkas:

```
from flask import Flask, request, jsonify
import librosa, numpy as np, tensorflow as tf

app = Flask(__name__)
model = tf.keras.models.load_model("model.h5")

@app.route("/predict", methods=["POST"])
def predict():
    file = request.files['audio']
    y, sr = librosa.load(file, sr=16000)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
    mfcc = np.expand_dims(mfcc, axis=(0,-1))  # batch & channel
    pred = model.predict(mfcc)
    label = np.argmax(pred)
    return jsonify({"prediksi": int(label), "prob": float(np.max(pred))})
```

---

5. Frontend React + Vite

- **UI sederhana**:
  - Tombol "rekam suara" atau "upload file".
  - Kirim audio ke Flask API.
  - Tampilkan hasil prediksi (angka + confidence).
- **Integrasi**:
  - Gunakan `fetch("/predict", { method: "POST", body: FormData })`.
  - Render hasil di komponen React.

---

6. Demo & Edukasi

- **Visualisasi spektrogram input** di frontend (pakai canvas).
- **Tampilkan probabilitas per kelas** (bar chart angka 0–9).

- **Interaktif untuk siswa**: mereka bisa ucapkan angka, lihat hasil klasifikasi, dan belajar bagaimana GD bekerja di balik layar.

Ringkasan alur

1. **Data audio** → pra-pemrosesan (MFCC/spectrogram).
2. **Model CNN Keras** → training klasifikasi angka.
3. **Evaluasi** → akurasi, confusion matrix.
4. **Flask API** → endpoint prediksi audio.
5. **React Vite frontend** → upload/rekam audio, tampilkan hasil.
6. **Visualisasi edukatif** → spektrogram + probabilitas.

👉 Mau aku bikinkan **contoh notebook end-to-end** (Python Keras + Flask stub + React fetch) supaya kamu bisa langsung jalankan di kelas, atau lebih detail di bagian **frontend React**?

Sistem klasifikasi suara angka 0–9 dengan Python, Keras, Flask, SQLite, dan React Vite

Kita bangun alur end-to-end: rekam suara di browser, simpan ke database dengan label angka, latih model CNN berbasis MFCC, simpan model, lalu prediksi suara baru dan tampilkan hasilnya di frontend.

Arsitektur dan alur kerja

- **Frontend (React Vite):** Rekam suara via microphone, input label angka, kirim audio ke API, tampilkan hasil prediksi.
- **Backend (Flask):** Endpoint untuk upload data, training, dan prediksi. Simpan metadata di **SQLite** dan file audio di folder server.
- **Pemrosesan audio:** Ekstraksi **MFCC** (atau Mel-spectrogram) dengan Librosa.
- **Model:** CNN sederhana untuk klasifikasi 10 kelas (0–9).
- **Database:** Tabel `audio_data(id, label, path, sr, duration)`.

Struktur proyek

- **server/**
  - app.py
  - models.py (opsional jika pakai SQLAlchemy)
  - audio_utils.py
  - saved_model/ (model tersimpan)
  - uploads/ (file WAV)
  - requirements.txt
- **client/**
  - index.html
  - src/main.jsx
  - src/App.jsx

Backend Flask + SQLite

Instalasi dependensi

```
pip install flask flask-cors librosa soundfile tensorflow sqlite3
# Jika sqlite3 sudah built-in di Python, tidak perlu install terpisah
pip install numpy scikit-learn
```

Skema database

```
# models.py (opsional, jika pakai sqlite3 langsung bisa di app.py)
import sqlite3

def get_db():
    conn = sqlite3.connect("audio.db")
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    conn = get_db()
    conn.execute("""
    CREATE TABLE IF NOT EXISTS audio_data (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        label INTEGER NOT NULL,
        path TEXT NOT NULL,
        sr INTEGER,
        duration REAL,
        created_at TEXT
    )
    """)
    conn.commit()
    conn.close()
```

Utilitas audio (MFCC)

```
# audio_utils.py
import numpy as np
import librosa

def load_wav_from_bytes(file_bytes, target_sr=16000):
    y, sr = librosa.load(file_bytes, sr=target_sr, mono=True)
    return y, sr

def extract_mfcc(y, sr, n_mfcc=40, hop_length=512, n_fft=1024):
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc,
                                hop_length=hop_length, n_fft=n_fft)
    # Normalisasi per-koefisien
    mfcc = (mfcc - np.mean(mfcc, axis=1, keepdims=True)) / (np.std(mfcc,
axis=1, keepdims=True) + 1e-6)
    return mfcc  # shape: (n_mfcc, time)

def pad_or_trim_feature(feat, max_time=100):
    # Pad/truncate di dimensi waktu (axis=1)
    t = feat.shape[1]
    if t < max_time:
        pad = np.zeros((feat.shape[0], max_time - t))
        feat = np.concatenate([feat, pad], axis=1)
    else:
        feat = feat[:, :max_time]
    return feat
```

Model Keras (CNN sederhana)

```
# app.py (bagian model builder)
import tensorflow as tf
from tensorflow.keras import layers, models

def build_model(input_shape=(40, 100, 1), num_classes=10):
    model = models.Sequential([
        layers.Conv2D(32, (3,3), padding="same", activation='relu',
input_shape=input_shape),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), padding="same", activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128, (3,3), padding="same", activation='relu'),
        layers.BatchNormalization(),
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Flask API

```
# app.py
import os, io, datetime, sqlite3
import numpy as np
from flask import Flask, request, jsonify
from flask_cors import CORS
import soundfile as sf
import tensorflow as tf
from audio_utils import load_wav_from_bytes, extract_mfcc, pad_or_trim_feature
from models import init_db

UPLOAD_DIR = "uploads"
MODEL_DIR = "saved_model"
DB_PATH = "audio.db"
os.makedirs(UPLOAD_DIR, exist_ok=True)
os.makedirs(MODEL_DIR, exist_ok=True)

app = Flask(__name__)
CORS(app)

# In-memory cached model
MODEL_PATH = os.path.join(MODEL_DIR, "mfcc_cnn.h5")
model = None

def get_db():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn

@app.before_first_request
def setup():
```

```
    init_db()

@app.route("/upload", methods=["POST"])
def upload():
    # FormData: audio (Blob), label (int)
    if 'audio' not in request.files or 'label' not in request.form:
        return jsonify({"error": "audio file and label required"}), 400

    file = request.files["audio"]
    label = int(request.form["label"])

    # Simpan file sebagai WAV di server
    raw = file.read()
    y, sr = load_wav_from_bytes(io.BytesIO(raw), target_sr=16000)
    duration = len(y) / sr

    # Filename unik
    fname =
f"{datetime.datetime.utcnow().strftime('%Y%m%d%H%M%S%f')}_{label}.wav"
    fpath = os.path.join(UPLOAD_DIR, fname)
    sf.write(fpath, y, sr)

    # Simpan metadata ke SQLite
    conn = get_db()
    conn.execute("INSERT INTO audio_data (label, path, sr, duration,
created_at) VALUES (?, ?, ?, ?, ?)",
                 (label, fpath, sr, duration,
datetime.datetime.utcnow().isoformat()))
    conn.commit()
    conn.close()

    return jsonify({"message": "uploaded", "path": fname, "label": label,
"sr": sr, "duration": duration})

@app.route("/train", methods=["POST"])
def train():
    # Optional: hyperparameter dari request.json
    max_time = int(request.json.get("max_time", 100))
    n_mfcc = int(request.json.get("n_mfcc", 40))
    epochs = int(request.json.get("epochs", 20))
    batch_size = int(request.json.get("batch_size", 32))
    val_split = float(request.json.get("val_split", 0.2))

    # Load data dari DB
    conn = get_db()
    rows = conn.execute("SELECT id, label, path FROM audio_data").fetchall()
    conn.close()

    if len(rows) < 20:
        return jsonify({"error": "not enough data to train"}), 400

    X, y = [], []
    for r in rows:
        y_wav, sr = sf.read(r["path"])
        if y_wav.ndim > 1:
            y_wav = y_wav[:,0]
        mfcc = extract_mfcc(y_wav, sr, n_mfcc=n_mfcc)
```

```
        feat = pad_or_trim_feature(mfcc, max_time=max_time)
        X.append(feat)
        y.append(r["label"])

    X = np.array(X, dtype=np.float32)  # shape: (N, n_mfcc, max_time)
    X = np.expand_dims(X, -1)          # add channel -> (N, n_mfcc, max_time,
1)
    y = np.array(y, dtype=np.int64)

    # Build or load model
    global model
    model = build_model(input_shape=(n_mfcc, max_time, 1), num_classes=10)

    history = model.fit(X, y, epochs=epochs, batch_size=batch_size,
validation_split=val_split,
                        callbacks=[
                            tf.keras.callbacks.EarlyStopping(patience=3,
restore_best_weights=True),
                            tf.keras.callbacks.ReduceLROnPlateau(patience=2)
                        ])

    # Simpan model
    model.save(MODEL_PATH)

    return jsonify({
        "message": "training complete",
        "epochs": len(history.history["loss"]),
        "train_acc": float(history.history["accuracy"][-1]),
        "val_acc": float(history.history["val_accuracy"][-1])
    })

@app.route("/predict", methods=["POST"])
def predict():
    global model
    if model is None:
        if not os.path.exists(MODEL_PATH):
            return jsonify({"error": "model not found. train first."}), 400
        model = tf.keras.models.load_model(MODEL_PATH)

    if 'audio' not in request.files:
        return jsonify({"error": "audio file required"}), 400

    file = request.files['audio']
    raw = file.read()
    y_wav, sr = load_wav_from_bytes(io.BytesIO(raw), target_sr=16000)
    mfcc = extract_mfcc(y_wav, sr, n_mfcc=model.input_shape[1])
    feat = pad_or_trim_feature(mfcc, max_time=model.input_shape[2])
    X = np.expand_dims(np.expand_dims(feat.astype(np.float32), -1), 0)

    probs = model.predict(X)[0]  # shape (10,)
    pred_label = int(np.argmax(probs))
    return jsonify({
        "prediksi": pred_label,
        "probabilitas": [float(p) for p in probs]
    })
```

Frontend React + Vite

Inisialisasi proyek

```
npm create vite@latest client -- --template react
cd client
npm install
npm install recorder-js
# atau gunakan MediaRecorder native (tanpa lib)
```

Rekam audio dan upload

```
// src/App.jsx
import { useEffect, useRef, useState } from "react";

export default function App() {
  const [recording, setRecording] = useState(false);
  const [label, setLabel] = useState("");
  const [pred, setPred] = useState(null);
  const mediaRecorderRef = useRef(null);
  const chunksRef = useRef([]);

  async function startRecording() {
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    const mr = new MediaRecorder(stream);
    chunksRef.current = [];
    mr.ondataavailable = (e) => {
      if (e.data.size > 0) chunksRef.current.push(e.data);
    };
    mr.onstop = () => {
      const blob = new Blob(chunksRef.current, { type: "audio/webm" });
      // Optional: convert to WAV server-side; kita kirim webm dan dibaca
librosa dari bytes
      uploadAudio(blob);
    };
    mediaRecorderRef.current = mr;
    mr.start();
    setRecording(true);
  }

  function stopRecording() {
    mediaRecorderRef.current?.stop();
    setRecording(false);
  }

  async function uploadAudio(blob) {
    if (!label || isNaN(+label)) {
      alert("Masukkan label angka 0-9");
      return;
    }
    const fd = new FormData();
    fd.append("audio", blob, "recording.webm");
    fd.append("label", label);
    const res = await fetch("http://localhost:5000/upload", {
      method: "POST",
      body: fd
    });
    const json = await res.json();
    console.log(json);
```

```
    alert("Upload berhasil");
  }

  async function trainModel() {
    const res = await fetch("http://localhost:5000/train", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ epochs: 15, batch_size: 32, n_mfcc: 40, max_time:
100 })
    });
    const json = await res.json();
    console.log(json);
    alert(`Training selesai. val_acc: ${json.val_acc?.toFixed(3)}`);
  }

  async function predictAudio() {
    // rekam singkat untuk prediksi
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    const mr = new MediaRecorder(stream);
    chunksRef.current = [];
    mr.ondataavailable = (e) => {
      if (e.data.size > 0) chunksRef.current.push(e.data);
    };
    mr.onstop = async () => {
      const blob = new Blob(chunksRef.current, { type: "audio/webm" });
      const fd = new FormData();
      fd.append("audio", blob, "predict.webm");
      const res = await fetch("http://localhost:5000/predict", {
        method: "POST",
        body: fd
      });
      const json = await res.json();
      setPred(json);
    };
    mr.start();
    setTimeout(() => mr.stop(), 1500); // rekam 1.5 detik
  }

  return (
    <div style={{ padding: 24 }}>
      <h2>Klasifikasi suara angka 0-9</h2>
      <div>
        <input
          type="number"
          min="0"
          max="9"
          placeholder="Label angka"
          value={label}
          onChange={(e) => setLabel(e.target.value)}
        />
        <div style={{ marginTop: 12 }}>
          {!recording ? (
            <button onClick={startRecording}>Mulai rekam & upload</button>
          ) : (
            <button onClick={stopRecording}>Stop & upload</button>
          )}
        </div>
```

```
      </div>
      <div style={{ marginTop: 24 }}>
        <button onClick={trainModel}>Train model</button>
        <button onClick={predictAudio} style={{ marginLeft: 8 }}>
          Prediksi suara
        </button>
      </div>
      {pred && (
        <div style={{ marginTop: 24 }}>
          <h4>Hasil prediksi</h4>
          <p>Label: {pred.prediksi}</p>
          <div style={{ display: "flex", gap: 8 }}>
            {pred.probabilitas?.map((p, i) => (
              <div key={i} style={{ textAlign: "center" }}>
                <div>{i}</div>
                <div style={{ background: "#4caf50", height: `${p * 100}px`,
width: 20 }} />
                <div>{p.toFixed(2)}</div>
              </div>
            ))}
          </div>
        </div>
      )}
    </div>
  );
}
```

Konfigurasi Vite dev server (CORS)

- Jalankan Flask di `http://localhost:5000` dan Vite di `http://localhost:5173`.
- Pastikan `flask-cors` aktif di backend. Jika perlu proxy, di `vite.config.js`:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/upload': 'http://localhost:5000',
      '/train': 'http://localhost:5000',
      '/predict': 'http://localhost:5000',
    }
  }
})
```

Catatan penting dan peningkatan kualitas

- **Format audio browser:** MediaRecorder default menghasilkan WebM/Opus. Librosa bisa membaca dari bytes jika FFmpeg mendukung. Alternatif aman: konversi server-side ke WAV (seperti contoh yang menulis WAV) dengan soundfile setelah memuat waveform dari bytes; jika tidak terbaca, pakai `pydub`/FFmpeg untuk decode.
- **Variasi data:** Kumpulkan banyak variasi speaker, tempo, noise. Lakukan augmentasi: time-shift, additive noise, pitch-shift, SpecAugment.
- **Balancing kelas:** Pastikan jumlah contoh merata untuk 0–9.
- **Validasi manual:** Render spektrogram di frontend untuk edukasi (bisa kirim fitur dari backend).

- **Model checkpoint:** Simpan history dan confusion matrix setelah training untuk evaluasi.
- **Performansi:** Untuk perangkat lemah, turunkan `n_mfcc`, `max_time`, dan gunakan float16 saat inference.

Alur penggunaan

1. Jalankan backend Flask.
2. Jalankan frontend Vite.
3. Rekam suara sambil memasukkan label angka, klik stop untuk upload.
4. Ulangi hingga terkumpul contoh cukup (minimal puluhan per kelas).
5. Klik “Train model”.
6. Gunakan “Prediksi suara” untuk mengetes, lihat hasil dan probabilitas per kelas.

Kalau kamu mau, aku siapkan versi siap-pakai dengan file terpisah, plus notebook untuk analisis hasil (loss, akurasi, confusion matrix) dan opsi mengganti MFCC ke Mel-spectrogram agar siswa bisa membandingkan.

Proyek siap-pakai: klasifikasi suara angka 0–9 dengan MFCC vs Mel-spectrogram

Berikut paket lengkap dengan file terpisah: backend Flask + SQLite, frontend React Vite, dan notebook analisis (loss, akurasi, confusion matrix) serta opsi mengganti MFCC ke Mel-spectrogram.

Struktur direktori

```
audio-digit-classifier/
├─ server/
│  ├─ app.py
│  ├─ audio_utils.py
│  ├─ db.py
│  ├─ model_builder.py
│  ├─ requirements.txt
│  ├─ uploads/
│  └─ saved_model/
├─ client/
│  ├─ index.html
│  ├─ vite.config.js
│  └─ src/
│     ├─ main.jsx
│     └─ App.jsx
└─ notebooks/
   └─ analysis.ipynb
```

Backend Flask dan SQLite

requirements.txt

```
flask
flask-cors
tensorflow
librosa
soundfile
numpy
scikit-learn
```

```
matplotlib
seaborn
```

db.py

```
import sqlite3
from pathlib import Path

DB_PATH = Path("audio.db").as_posix()

def get_db():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    conn = get_db()
    conn.execute("""
    CREATE TABLE IF NOT EXISTS audio_data (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        label INTEGER NOT NULL,
        path TEXT NOT NULL,
        sr INTEGER,
        duration REAL,
        created_at TEXT
    )""")
    conn.execute("""
    CREATE TABLE IF NOT EXISTS training_runs (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        started_at TEXT,
        finished_at TEXT,
        n_samples INTEGER,
        n_mfcc INTEGER,
        max_time INTEGER,
        feature_type TEXT, -- 'mfcc' atau 'mel'
        train_acc REAL,
        val_acc REAL,
        model_path TEXT,
        history_path TEXT
    )""")
    conn.commit()
    conn.close()
```

audio_utils.py

```
import io
import numpy as np
import librosa

def load_wav_from_bytes(file_bytes, target_sr=16000):
    # file_bytes: BytesIO atau bytes
    if isinstance(file_bytes, (bytes, bytearray)):
        file_bytes = io.BytesIO(file_bytes)
    y, sr = librosa.load(file_bytes, sr=target_sr, mono=True)
    return y, sr

def extract_mfcc(y, sr, n_mfcc=40, hop_length=512, n_fft=1024):
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc,
                                hop_length=hop_length, n_fft=n_fft)
```

```
    mfcc = (mfcc - np.mean(mfcc, axis=1, keepdims=True)) / (np.std(mfcc,
axis=1, keepdims=True) + 1e-6)
    return mfcc  # (n_mfcc, time)

def extract_mel(y, sr, n_mels=64, hop_length=512, n_fft=1024):
    mel = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels,
                                         hop_length=hop_length, n_fft=n_fft)
    mel_db = librosa.power_to_db(mel, ref=np.max)
    mel_db = (mel_db - np.mean(mel_db)) / (np.std(mel_db) + 1e-6)
    return mel_db  # (n_mels, time)

def pad_or_trim_feature(feat, max_time=100):
    t = feat.shape[1]
    if t < max_time:
        pad = np.zeros((feat.shape[0], max_time - t), dtype=feat.dtype)
        feat = np.concatenate([feat, pad], axis=1)
    else:
        feat = feat[:, :max_time]
    return feat
```

model_builder.py

```
import tensorflow as tf
from tensorflow.keras import layers, models

def build_cnn(input_shape=(40, 100, 1), num_classes=10):
    model = models.Sequential([
        layers.Conv2D(32, (3,3), padding="same", activation='relu',
input_shape=input_shape),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), padding="same", activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128, (3,3), padding="same", activation='relu'),
        layers.BatchNormalization(),
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

app.py

```
import os, io, datetime, json
import numpy as np
import soundfile as sf
from pathlib import Path
from flask import Flask, request, jsonify
from flask_cors import CORS
import tensorflow as tf

from db import get_db, init_db
from audio_utils import load_wav_from_bytes, extract_mfcc, extract_mel,
pad_or_trim_feature
from model_builder import build_cnn
```

```
UPLOAD_DIR = Path("uploads"); UPLOAD_DIR.mkdir(exist_ok=True)
MODEL_DIR = Path("saved_model"); MODEL_DIR.mkdir(exist_ok=True)
HISTORY_DIR = Path("saved_model"); HISTORY_DIR.mkdir(exist_ok=True)

app = Flask(__name__)
CORS(app)

MODEL_CACHE = None
MODEL_META = {"feature_type": "mfcc", "feat_size": 40, "max_time": 100}

@app.before_first_request
def setup():
    init_db()

@app.route("/upload", methods=["POST"])
def upload():
    # FormData: audio (Blob), label (int)
    if 'audio' not in request.files or 'label' not in request.form:
        return jsonify({"error": "audio file and label required"}), 400

    f = request.files["audio"]
    label = int(request.form["label"])
    raw = f.read()
    y, sr = load_wav_from_bytes(io.BytesIO(raw), target_sr=16000)
    duration = len(y) / sr

    fname =
f"{datetime.datetime.utcnow().strftime('%Y%m%d%H%M%S%f')}_{label}.wav"
    fpath = UPLOAD_DIR / fname
    sf.write(fpath.as_posix(), y, sr)

    conn = get_db()
    conn.execute(
        "INSERT INTO audio_data (label, path, sr, duration, created_at) VALUES
(?, ?, ?, ?, ?)",
        (label, fpath.as_posix(), sr, duration,
datetime.datetime.utcnow().isoformat())
    )
    conn.commit()
    conn.close()
    return jsonify({"message": "uploaded", "file": fname, "label": label,
"sr": sr, "duration": duration})

@app.route("/train", methods=["POST"])
def train():
    data = request.get_json(force=True)
    feature_type = data.get("feature_type", "mfcc")  # 'mfcc' atau 'mel'
    feat_size = int(data.get("feat_size", 40))       # n_mfcc atau n_mels
    max_time = int(data.get("max_time", 100))
    epochs = int(data.get("epochs", 20))
    batch_size = int(data.get("batch_size", 32))
    val_split = float(data.get("val_split", 0.2))

    conn = get_db()
    rows = conn.execute("SELECT id, label, path FROM audio_data").fetchall()
    conn.close()
```

```
    if len(rows) < 50:
        return jsonify({"error": "need at least ~50 samples"}), 400

    X, y = [], []
    for r in rows:
        wav, sr = sf.read(r["path"])
        if wav.ndim > 1:
            wav = wav[:,0]
        if feature_type == "mfcc":
            feat = extract_mfcc(wav, sr, n_mfcc=feat_size)
        else:
            feat = extract_mel(wav, sr, n_mels=feat_size)
        feat = pad_or_trim_feature(feat, max_time=max_time)
        X.append(feat)
        y.append(r["label"])

    X = np.array(X, dtype=np.float32)
    X = np.expand_dims(X, -1)
    y = np.array(y, dtype=np.int64)

    model = build_cnn(input_shape=(feat_size, max_time, 1), num_classes=10)
    history = model.fit(
        X, y, epochs=epochs, batch_size=batch_size,
validation_split=val_split,
        callbacks=[
            tf.keras.callbacks.EarlyStopping(patience=3,
restore_best_weights=True),
            tf.keras.callbacks.ReduceLROnPlateau(patience=2)
        ]
    )

    ts = datetime.datetime.utcnow().strftime("%Y%m%d%H%M%S")
    model_path = MODEL_DIR /
f"model_{feature_type}_{feat_size}_{max_time}_{ts}.h5"
    hist_path = HISTORY_DIR /
f"history_{feature_type}_{feat_size}_{max_time}_{ts}.json"

    model.save(model_path.as_posix())
    with open(hist_path, "w") as f:
        json.dump({k: [float(vv) for vv in v] for k, v in
history.history.items()}, f)

    conn = get_db()
    conn.execute("""
        INSERT INTO training_runs (started_at, finished_at, n_samples, n_mfcc,
max_time,
                                   feature_type, train_acc, val_acc,
model_path, history_path)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)""",
        (ts, datetime.datetime.utcnow().isoformat(), len(X), feat_size,
max_time,
         feature_type, float(history.history["accuracy"][-1]),
         float(history.history["val_accuracy"][-1]),
         model_path.as_posix(), hist_path.as_posix()))
    conn.commit()
    conn.close()
```

```
    global MODEL_CACHE, MODEL_META
    MODEL_CACHE = model
    MODEL_META = {"feature_type": feature_type, "feat_size": feat_size,
"max_time": max_time,
                  "model_path": model_path.as_posix(), "history_path":
hist_path.as_posix()}

    return jsonify({
        "message": "training complete",
        "train_acc": float(history.history["accuracy"][-1]),
        "val_acc": float(history.history["val_accuracy"][-1]),
        "model_path": model_path.name,
        "history_path": hist_path.name
    })

@app.route("/predict", methods=["POST"])
def predict():
    if 'audio' not in request.files:
        return jsonify({"error": "audio file required"}), 400

    # Load model dari cache atau dari file terakhir
    global MODEL_CACHE, MODEL_META
    if MODEL_CACHE is None:
        # Cari run terakhir
        conn = get_db()
        row = conn.execute("SELECT * FROM training_runs ORDER BY id DESC LIMIT
1").fetchone()
        conn.close()
        if not row:
            return jsonify({"error": "no trained model"}), 400
        MODEL_META = {
            "feature_type": row["feature_type"],
            "feat_size": row["n_mfcc"],
            "max_time": row["max_time"]
        }
        MODEL_CACHE = tf.keras.models.load_model(row["model_path"])

    raw = request.files["audio"].read()
    y, sr = load_wav_from_bytes(io.BytesIO(raw), target_sr=16000)
    if MODEL_META["feature_type"] == "mfcc":
        feat = extract_mfcc(y, sr, n_mfcc=MODEL_META["feat_size"])
    else:
        feat = extract_mel(y, sr, n_mels=MODEL_META["feat_size"])
    feat = pad_or_trim_feature(feat, max_time=MODEL_META["max_time"])

    X = np.expand_dims(np.expand_dims(feat.astype(np.float32), -1), 0)
    probs = MODEL_CACHE.predict(X)[0]
    pred_label = int(np.argmax(probs))
    return jsonify({
        "prediksi": pred_label,
        "probabilitas": [float(p) for p in probs],
        "feature_type": MODEL_META["feature_type"]
    })

@app.route("/runs", methods=["GET"])
def runs():
    conn = get_db()
```

```
    rows = conn.execute("SELECT id, started_at, feature_type, n_mfcc,
max_time, train_acc, val_acc, model_path, history_path FROM training_runs
ORDER BY id DESC").fetchall()
    conn.close()
    return jsonify([dict(r) for r in rows])

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Frontend React + Vite

index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Klasifikasi Suara Angka</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/upload': 'http://localhost:5000',
      '/train': 'http://localhost:5000',
      '/predict': 'http://localhost:5000',
      '/runs': 'http://localhost:5000'
    }
  }
})
```

src/main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(<App />)
```

src/App.jsx

```
import { useEffect, useRef, useState } from "react";

export default function App() {
  const [recording, setRecording] = useState(false);
  const [label, setLabel] = useState("");
  const [pred, setPred] = useState(null);
  const [runs, setRuns] = useState([]);
  const [featureType, setFeatureType] = useState("mfcc");
  const [trainingInfo, setTrainingInfo] = useState(null);
```

```
  const mediaRecorderRef = useRef(null);
  const chunksRef = useRef([]);

  useEffect(() => {
    fetch("/runs").then(r => r.json()).then(setRuns).catch(()=>{});
  }, []);

  async function startRecording(toPredict=false) {
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    const mr = new MediaRecorder(stream);
    chunksRef.current = [];
    mr.ondataavailable = (e) => { if (e.data.size > 0)
chunksRef.current.push(e.data); };
    mr.onstop = () => {
      const blob = new Blob(chunksRef.current, { type: "audio/webm" });
      toPredict ? sendForPredict(blob) : sendForUpload(blob);
    };
    mediaRecorderRef.current = mr;
    mr.start();
    setRecording(true);
  }

  function stopRecording() {
    mediaRecorderRef.current?.stop();
    setRecording(false);
  }

  async function sendForUpload(blob) {
    if (!label || isNaN(+label)) {
      alert("Masukkan label angka 0-9");
      return;
    }
    const fd = new FormData();
    fd.append("audio", blob, "recording.webm");
    fd.append("label", label);
    const res = await fetch("/upload", { method: "POST", body: fd });
    const json = await res.json();
    alert(json.message || "Upload selesai");
  }

  async function trainModel() {
    const payload = { feature_type: featureType, feat_size: featureType ===
"mfcc" ? 40 : 64, max_time: 100, epochs: 15, batch_size: 32 };
    const res = await fetch("/train", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(payload)
    });
    const json = await res.json();
    setTrainingInfo(json);
    fetch("/runs").then(r => r.json()).then(setRuns);
  }

  async function sendForPredict(blob) {
    const fd = new FormData();
    fd.append("audio", blob, "predict.webm");
    const res = await fetch("/predict", { method: "POST", body: fd });
```

```
    const json = await res.json();
    setPred(json);
  }

  return (
    <div style={{ padding: 24, fontFamily: "system-ui" }}>
      <h2>Klasifikasi Suara Angka 0–9</h2>

      <div style={{ display: "flex", gap: 24, flexWrap: "wrap" }}>
        <div>
          <h4>Rekam & Simpan Dataset</h4>
          <input type="number" min="0" max="9" placeholder="Label angka"
            value={label} onChange={(e) => setLabel(e.target.value)} />
          <div style={{ marginTop: 8 }}>
            {!recording ? (
              <button onClick={() => startRecording(false)}>Mulai
rekam</button>
            ) : (
              <button onClick={stopRecording}>Stop & upload</button>
            )}
          </div>
        </div>

        <div>
          <h4>Training</h4>
          <div style={{ marginBottom: 8 }}>
            <label>Pilih fitur: </label>
            <select value={featureType}
onChange={(e)=>setFeatureType(e.target.value)}>
              <option value="mfcc">MFCC</option>
              <option value="mel">Mel-spectrogram</option>
            </select>
          </div>
          <button onClick={trainModel}>Train model</button>
          {trainingInfo && (
            <div style={{ marginTop: 8 }}>
              <div>Train acc:
{Number(trainingInfo.train_acc).toFixed(3)}</div>
              <div>Val acc: {Number(trainingInfo.val_acc).toFixed(3)}</div>
              <div>Model: {trainingInfo.model_path}</div>
            </div>
          )}
        </div>

        <div>
          <h4>Prediksi Cepat</h4>
          <button onClick={() => startRecording(true)}>Rekam untuk
prediksi</button>
          {pred && (
            <div style={{ marginTop: 8 }}>
              <div>Prediksi: {pred.prediksi} ({pred.feature_type})</div>
              <div style={{ display: "flex", gap: 6, alignItems: "flex-end",
marginTop: 8 }}>
                {pred.probabilitas?.map((p, i) => (
                  <div key={i} style={{ textAlign: "center" }}>
                    <div style={{ fontSize: 12 }}>{i}</div>
```

```
                    <div style={{ background: "#4caf50", height: `${p *
120}px`, width: 16 }} />
                    <div style={{ fontSize: 12 }}>{p.toFixed(2)}</div>
                  </div>
                ))}
              </div>
            </div>
          )}
        </div>

        <div style={{ minWidth: 260 }}>
          <h4>Riwayat Training</h4>
          <ul>
            {runs.map(r => (
              <li key={r.id}>
                #{r.id} {r.feature_type} n={r.n_mfcc} T={r.max_time}
val_acc={Number(r.val_acc).toFixed(3)}
              </li>
            ))}
          </ul>
        </div>
      </div>
    </div>
  );
}
```

Notebook analisis hasil

File: notebooks/analysis.ipynb (konten sel Python di bawah; salin ke notebook)

```
# Sel 1: Import dan load history terakhir
import json, sqlite3, os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import soundfile as sf

DB_PATH = "../server/audio.db"

conn = sqlite3.connect(DB_PATH)
conn.row_factory = sqlite3.Row
row = conn.execute("SELECT * FROM training_runs ORDER BY id DESC LIMIT
1").fetchone()
conn.close()
print("Last run:", dict(row))

with open(row["history_path"], "r") as f:
    hist = json.load(f)

# Sel 2: Plot loss dan akurasi
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(hist["loss"], label="train")
plt.plot(hist["val_loss"], label="val")
plt.title("Loss per epoch"); plt.xlabel("Epoch"); plt.ylabel("Loss");
plt.legend()
```

```
plt.subplot(1,2,2)
plt.plot(hist["accuracy"], label="train")
plt.plot(hist["val_accuracy"], label="val")
plt.title("Accuracy per epoch"); plt.xlabel("Epoch"); plt.ylabel("Acc");
plt.legend()
plt.show()

# Sel 3: Bangun ulang fitur dan evaluasi confusion matrix (train split
sederhana)
import librosa
from audio_utils import extract_mfcc, extract_mel, pad_or_trim_feature
import tensorflow as tf

feat_type = row["feature_type"]
feat_size = row["n_mfcc"]
max_time = row["max_time"]

model = tf.keras.models.load_model(row["model_path"])

conn = sqlite3.connect(DB_PATH); conn.row_factory = sqlite3.Row
rows = conn.execute("SELECT label, path FROM audio_data").fetchall()
conn.close()

X, y = [], []
for r in rows:
    wav, sr = sf.read(r["path"])
    if wav.ndim > 1: wav = wav[:,0]
    if feat_type == "mfcc":
        feat = extract_mfcc(wav, sr, n_mfcc=feat_size)
    else:
        feat = extract_mel(wav, sr, n_mels=feat_size)
    feat = pad_or_trim_feature(feat, max_time=max_time)
    X.append(feat); y.append(r["label"])

X = np.expand_dims(np.array(X, dtype=np.float32), -1)
y = np.array(y, dtype=np.int64)

probs = model.predict(X, batch_size=32)
y_pred = probs.argmax(axis=1)

cm = confusion_matrix(y, y_pred, labels=list(range(10)))
print(classification_report(y, y_pred))
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (0-9)")
plt.xlabel("Predicted"); plt.ylabel("True")
plt.show()

# Sel 4: Bandingkan MFCC vs Mel-spectrogram (jika ada kedua run)
conn = sqlite3.connect(DB_PATH); conn.row_factory = sqlite3.Row
rows_run = conn.execute("SELECT feature_type, val_acc, n_mfcc, max_time FROM
training_runs ORDER BY id DESC LIMIT 10").fetchall()
conn.close()

for r in rows_run:
    print(f"run: {r['feature_type']} n={r['n_mfcc']} T={r['max_time']}
val_acc={r['val_acc']:.3f}")
```

Tips:

- Jika ingin split train/val/test yang konsisten, tambahkan kolom split pada tabel `audio_data` atau lakukan split deterministik di notebook, lalu evaluasi hanya test.

Cara menjalankan

1. Backend:
   - Buat venv, install requirements: `pip install -r server/requirements.txt`
   - Jalankan server: `python server/app.py` (port 5000)
2. Frontend:
   - Inisialisasi: `npm install` di folder `client`
   - Jalankan dev: `npm run dev` (port 5173)
3. Gunakan UI:
   - Rekam suara dengan label angka 0–9, upload berulang hingga terkumpul cukup data.
   - Pilih fitur MFCC atau Mel, klik Train model.
   - Rekam suara untuk Prediksi, lihat hasil dan probabilitas.
4. Analisis:
   - Buka `notebooks/analysis.ipynb`
   - Jalankan sel untuk melihat loss, akurasi, confusion matrix, dan ringkasan run (MFCC vs Mel).

Catatan kualitas dan edukasi

- Untuk decoding WebM/Opus, solusi di atas mengandalkan kemampuan Librosa/ffmpeg via soundfile; jika ada masalah pembacaan, pertimbangkan konversi server-side ke WAV tegas menggunakan ffmpeg/pydub.
- Pastikan jumlah contoh per kelas seimbang, variasikan speaker, tempo, dan noise ringan; tambahkan augmentasi jika diperlukan.
- Tampilkan spektrogram input di frontend untuk membantu siswa mengaitkan pola visual dengan hasil klasifikasi. Jika kamu mau, aku tambahkan endpoint `/feature_preview` untuk mengirim spektrogram ke UI sebagai gambar heatmap.