Binary Cross-Entropy

1. Single Perceptron untuk Klasifikasi Biner

Single Perceptron adalah jaringan saraf tunggal yang melakukan:

- **Input**: Fitur-fitur data (x_1, x_2, ..., x_n)
- **Output**: ŷ = σ(w·x + b), dimana σ adalah fungsi aktivasi
- **Tujuan**: Mengklasifikasikan input ke dalam kelas 0 atau 1

2. Mengapa MSE Tidak Ideal untuk Klasifikasi?

MSE Formula: $L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Masalah dengan MSE untuk klasifikasi:

A. Masalah Kurva Loss

```python
# Contoh visual masalah MSE
# Untuk y_true = 1:
- Jika ŷ = 0.9 → MSE = (1-0.9)² = 0.01
- Jika ŷ = 0.1 → MSE = (1-0.1)² = 0.81
# Masalah: Penurunan loss tidak proporsional dengan "keyakinan" klasifikasi
```

B. Masalah Optimisasi

- MSE menghasilkan permukaan loss yang **tidak convex** untuk klasifikasi
- Banyak **local minima** yang menghambat konvergensi
- Gradien menjadi sangat kecil ketika prediksi sudah "cukup baik"

3. Binary Cross-Entropy: Solusi yang Tepat

BCE Formula: $L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$

Komponen BCE:

- **y log(ŷ)**: Memberikan penalty tinggi jika model yakin pada prediksi salah
- **(1-y) log(1-ŷ)**: Sama untuk kasus kelas negatif

4. Perbandingan Langsung

Aspek	MSE	Binary Cross-Entropy
Tujuan	Regresi	Klasifikasi Biner
Output Range	ℝ	[0,1] (probabilitas)
Sensitivitas	Linear	Eksponensial
Gradien	Kecil untuk prediksi baik	Besar untuk prediksi salah
Convergence	Lambat untuk klasifikasi	Cepat dan stabil

5. Intuisi Matematis

Gradien MSE:

$$\frac{\partial L_{MSE}}{\partial w} = -2(y - \hat{y}) \cdot \hat{y}(1 - \hat{y}) \cdot x$$

Faktor **ŷ(1-ŷ)** dari sigmoid bisa membuat gradien sangat kecil!

Gradien BCE:

$$\frac{\partial L_{BCE}}{\partial w} = (\hat{y} - y) \cdot x$$

Sederhana dan efektif - gradien proporsional dengan error!

Binary Cross-Entropy ini dengan cara yang mudah dipahami

1. Analogi: "Seberapa Terkejut Kita dengan Prediksi"

Bayangkan kita bermain tebak-tebakan:

```text
"Jika model 90% yakin pada jawaban BENAR → wajar
Jika model 90% yakin pada jawaban SALAH → TERKEJUT!"
```

Binary Cross-Entropy mengukur **"tingkat keterkejutan"** model ketika prediksinya salah.

2. Kasus per Kasus

Kasus 1: Target y = 1 (Harus prediksi 1)

Prediksi (ŷ)	Loss	Penjelasan
0.9 (hampir benar)	-log(0.9) ≈ 0.1	Sedikit terkejut
0.5 (ragu-ragu)	-log(0.5) ≈ 0.69	Cukup terkejut
0.1 (salah parah)	-log(0.1) ≈ 2.3	SANGAT TERKEJUT!

Rumusnya: `Loss = -log(ŷ)`

Kasus 2: Target y = 0 (Harus prediksi 0)

Prediksi (ŷ)	Loss	Penjelasan
0.1 (hampir benar)	-log(0.9) ≈ 0.1	Sedikit terkejut
0.5 (ragu-ragu)	-log(0.5) ≈ 0.69	Cukup terkejut
0.9 (salah parah)	-log(0.1) ≈ 2.3	SANGAT TERKEJUT!

Rumusnya: `Loss = -log(1-ŷ)`

3. Gabungkan Kedua Kasus

Kita butuh **SATU rumus** yang bekerja untuk y=1 dan y=0:

```text
Jika y = 1: Loss = -log(ŷ)
Jika y = 0: Loss = -log(1-ŷ)
```

Bisa ditulis sebagai:

```text
Loss = -[y × log(ŷ) + (1-y) × log(1-ŷ)]
```

Mengapa ini bekerja?

- **Jika y = 1**:
  `(1-y) = 0`, jadi rumus menjadi: `-[1 × log(ŷ) + 0 × log(1-ŷ)] = -log(ŷ)` ✓
- **Jika y = 0**:
  `y = 0`, jadi rumus menjadi: `-[0 × log(ŷ) + 1 × log(1-ŷ)] = -log(1-ŷ)` ✓

4. Visualisasi "Tingkat Keterkejutan"

```text
Skala Keterkejutan:
ŷ = 0.9 → Loss = 0.1 (😊 tenang)
ŷ = 0.7 → Loss = 0.36 (😐 biasa)
ŷ = 0.5 → Loss = 0.69 (😟 heran)
ŷ = 0.3 → Loss = 1.20 (😱 terkejut)
ŷ = 0.1 → Loss = 2.30 (🤯 SANGAT TERKEJUT!)
```

5. Untuk Banyak Data Point (n samples)

Kita rata-ratakan keterkejutan untuk semua data:

```text
Total Keterkejutan = Σ keterkejutan tiap data
Rata-rata Keterkejutan = Total Keterkejutan ÷ n
```

Jadi:

$$L = -\frac{1}{n}\sum_{i=1}^{n}[\, y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

6. Contoh Numerik Sederhana

Misal kita punya 3 data:

Data	y (target)	ŷ (prediksi)	Loss per data
1	1	0.9	-log(0.9) = 0.105

Data	y (target)	ŷ (prediksi)	Loss per data
2	0	0.2	-log(0.8) = 0.223
3	1	0.3	-log(0.3) = 1.204

Total Loss = (0.105 + 0.223 + 1.204) / 3 = 1.532 / 3 = **0.511**

7. Analogi Kehidupan: Sistem Penilaian Guru

Bayangkan guru memberi nilai:

- **Prediksi BENAR dengan keyakinan TINGGI** → Nilai bagus (loss kecil)
- **Prediksi BENAR dengan keyakinan RENDAH** → Nilai cukup (loss sedang)
- **Prediksi SALAH dengan keyakinan TINGGI** → Nilai jelek (loss besar)
- **Prediksi SALAH dengan keyakinan RENDAH** → Nilai cukup (loss sedang)

Binary Cross-Entropy seperti guru yang:

- **Menghukum lebih keras** ketika siswa sangat yakin pada jawaban salah
- **Memberi kredit** ketika siswa tidak terlalu yakin pada jawaban salah

8. Mengapa Pakai Logaritma?

Karena logaritma memberikan **penalty yang semakin besar** secara eksponensial ketika prediksi semakin salah:

```text
Jika ŷ mendekati 0 (salah parah): -log(ŷ) → BESAR SEKALI
Jika ŷ mendekati 1 (benar): -log(ŷ) → KECIL
```

Kesimpulan untuk SMA:

Binary Cross-Entropy = Pengukur "Keterkejutan" Model

- Semakin yakin model pada jawaban salah → semakin TERKEJUT kita → loss BESAR
- Semakin yakin model pada jawaban benar → wajar saja → loss KECIL
- Diambil rata-rata untuk semua data

Jadi rumus yang terlihat menakutkan ini sebenarnya punya makna yang sangat intuitif! 😊

🎯 MSE vs BCE untuk Klasifikasi

MSE (Mean Squared Error): "Pelatih Terlalu Lembut"

```text
Prediksi: ŷ = 0.9, Target: y = 1
Error = (1 - 0.9)² = 0.01 → Gradien kecil
```

Masalah: Meskipun prediksi belum tepat (belum 1.0), gradien sudah sangat kecil → konvergensi lambat

BCE (Binary Cross-Entropy): "Pelatih yang Tegas"

```text
Prediksi: ŷ = 0.9, Target: y = 1
Loss = -log(0.9) ≈ 0.1 → Masih ada gradien yang berarti
Prediksi: ŷ = 0.6, Target: y = 1
Loss = -log(0.6) ≈ 0.51 → Gradien lebih besar
Prediksi: ŷ = 0.1, Target: y = 1
Loss = -log(0.1) ≈ 2.3 → Gradien SANGAT BESAR
```

📊 Decision Boundary yang Jelas

Dengan BCE, model belajar untuk **"yakin dan tegas"**:

```text
Decision Boundary di ŷ = 0.5:

ŷ ≥ 0.5 → Kelas 1    (yakin positif)
ŷ < 0.5  → Kelas 0    (yakin negatif)
```

Contoh Proses Belajar:

```text
Data: [fitur1, fitur2, target]
1. [2.0, 1.0, 1] → Model prediksi: ŷ = 0.6 → Loss = 0.51
2. [1.5, 0.5, 1] → Model prediksi: ŷ = 0.7 → Loss = 0.36
3. [3.0, 2.0, 1] → Model prediksi: ŷ = 0.9 → Loss = 0.10
4. [0.5, 1.5, 0] → Model prediksi: ŷ = 0.4 → Loss = 0.51
```

⚡ Mekanisme "Cutoff" yang Efektif

BCE memaksa model untuk memilih dengan yakin:

python
```python
# Dengan BCE, model belajar:
if z (weighted sum) >> 0:
    ŷ ≈ 1.0  # Yakin kelas 1
elif z << 0:
    ŷ ≈ 0.0  # Yakin kelas 0
else:
    ŷ ≈ 0.5  # Ragu-ragu → butuh belajar lebih
```

Visual Decision Boundary:

text
```text
Kelas 0: • • • • • | • • • • • Kelas 1
        ŷ ≈ 0.1   |   ŷ ≈ 0.9
         Confidence tinggi   Confidence tinggi
```

🔄 Proses Optimisasi yang Efisien

Gradien BCE = (ŷ - y) · x:

- **Jika salah jauh** → (ŷ - y) besar → gradien besar → update besar
- **Jika sudah benar** → (ŷ - y) kecil → gradien kecil → update kecil
- **Jika ragu-ragu** → gradien cukup besar → masih belajar aktif

Berbeda dengan MSE:

- Gradien = 2(y - ŷ) · ŷ(1-ŷ) · x
- Faktor ŷ(1-ŷ) bisa membuat gradien sangat kecil meskipun prediksi salah!

🎪 Analogi Tournament Games

MSE seperti wasit yang:

- "Wah selisihnya hanya 0.1, tidak apa-apa"
- Tidak mendorong tim untuk menang telak

BCE seperti wasit yang:

- "Kalian harus menang dengan yakin! Jangan hanya 1-0!"
- Mendorang tim untuk bermain maksimal sampai yakin menang

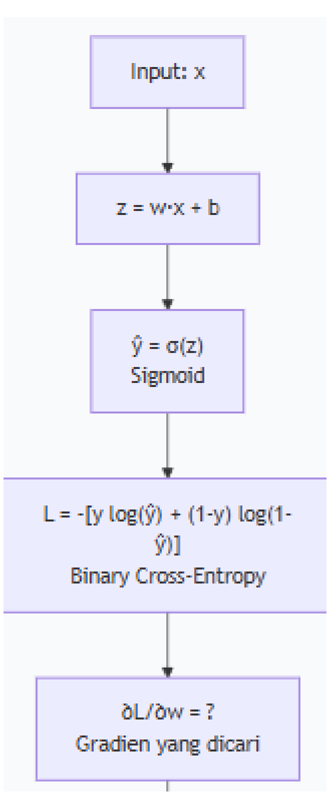
✅ Kesimpulan

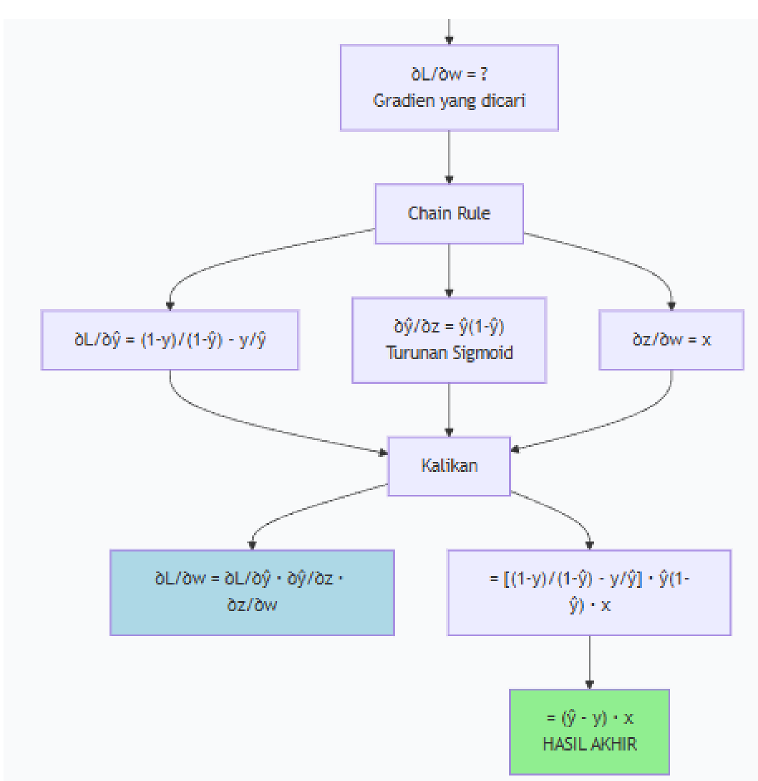
Benar sekali! BCE efektif untuk klasifikasi karena:

1. **Menciptakan decision boundary yang jelas** (0.5)
2. **Memberikan penalty proporsional dengan "keyakinan yang salah"**
3. **Gradien tetap berarti meskipun prediksi sudah "cukup baik"**
4. **Memaksa model untuk prediksi dengan confidence tinggi**

Jadi pilihan MSE vs BCE bukan hanya soal "mana yang bekerja", tapi **"mana yang bekerja EFISIEN dan EFEKTIF"** untuk tugas klasifikasi! 🚀

BCE = Loss function yang "tegas" untuk masalah klasifikasi yang butuh kepastian!





1. Setup Fungsi
Kita punya:

- **Binary Cross-Entropy Loss**:

$$L = -\frac{1}{n}\sum_{i=1}^{n}[\, y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

- **Output perceptron**:

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

- **Weighted sum**:

$$z = w \cdot x + b = \sum_j w_j\, x_j + b$$

2. Turunan Parsial untuk Satu Sample
Mari kita hitung untuk satu sample (hilangkan summation dan index i untuk sementara):

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

Kita ingin **∂L/∂w_j** (turunan terhadap weight ke-j).
Gunakan Chain Rule:

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

3. Hitung Masing-Masing Komponen
A. ∂L/∂ŷ

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

B. ∂ŷ/∂z (Turunan Sigmoid)

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

4. Gabungkan Semua Komponen

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

Sederhanakan:

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

5. Untuk Semua Samples
Untuk n samples, kita average:

$$\frac{\partial L}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \cdot x_{ij}$$

Dalam bentuk vektor:

$$\nabla_w L = \frac{1}{n} X^T (\hat{y} - y)$$

6. Verifikasi dengan Contoh Numerik
Misalkan:

- $y = 1$, $\hat{y} = 0.8$, $x = 2$

Hitung manual:

$$\frac{\partial L}{\partial w} = (0.8 - 1) \times 2 = -0.4$$

Bukti dengan chain rule:

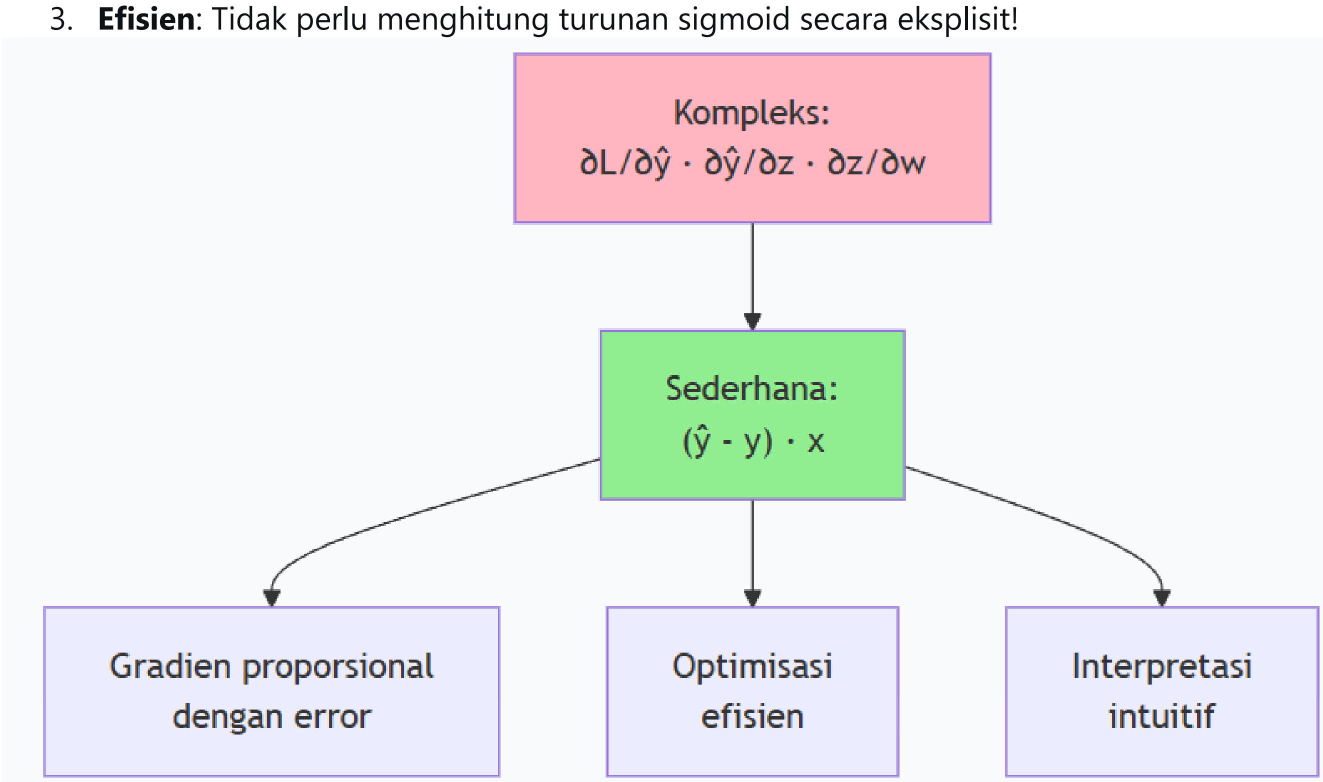
- ∂L/∂ŷ = (1-1)/(1-0.8) - 1/0.8 = 0 - 1.25 = -1.25
- ∂ŷ/∂z = 0.8 × (1-0.8) = 0.16
- ∂z/∂w = 2
- Hasil: -1.25 × 0.16 × 2 = -0.4 ✓

7. Keindahan Hasil Akhir

$$\frac{\partial L}{\partial w} = (\hat{y} - y) \cdot x$$

Mengapa ini indah?

1. **Sederhana**: Hanya (prediksi - target) × input
2. **Intuitif**:
   - Jika ŷ > y (overconfident), gradien positif → turunkan ŷ
   - Jika ŷ < y (underconfident), gradien negatif → naikkan ŷ
3. **Efisien**: Tidak perlu menghitung turunan sigmoid secara eksplisit!



8. Untuk Bias b

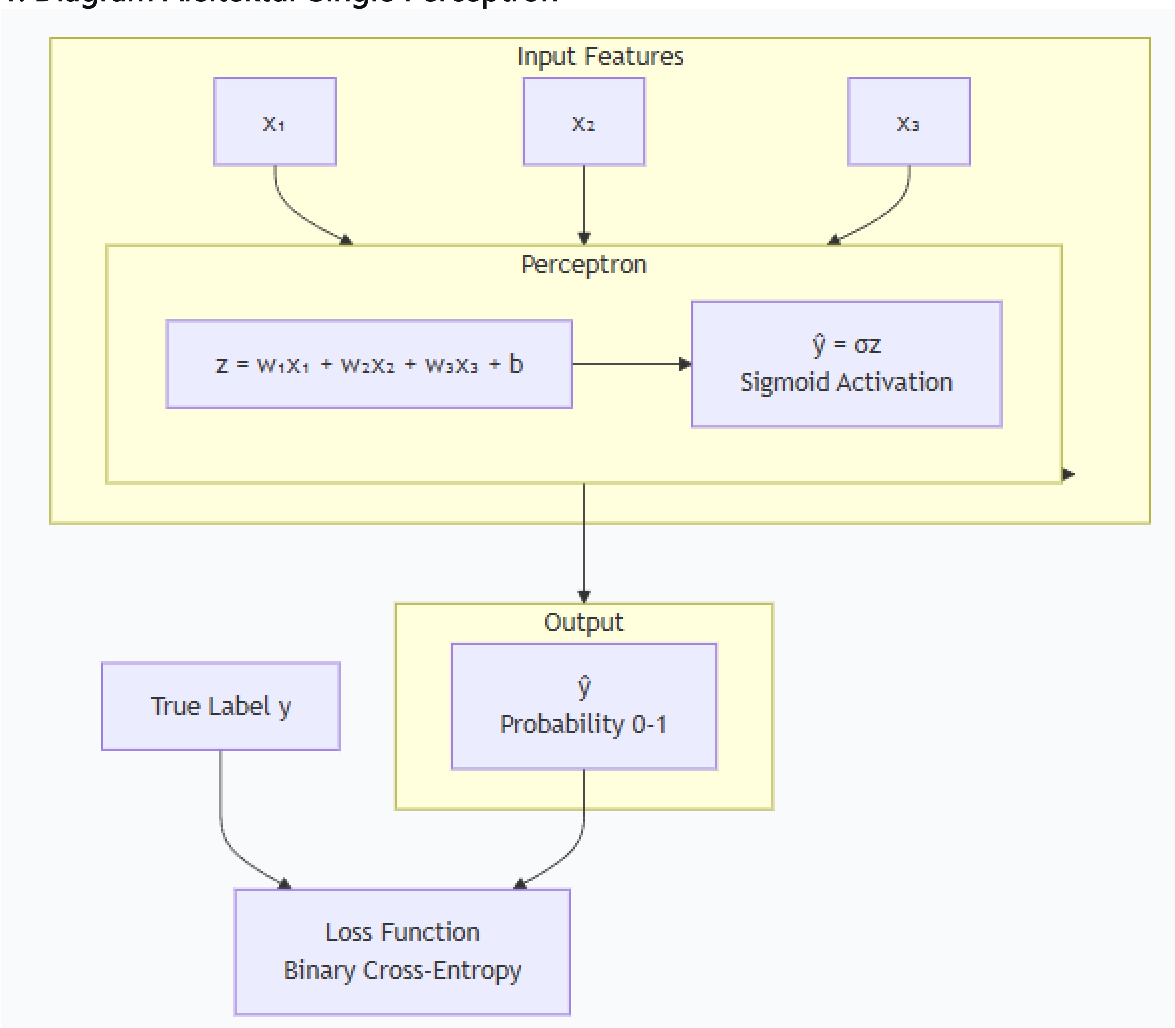
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b}$$
$$\frac{\partial z}{\partial b} = 1$$
$$\frac{\partial L}{\partial b} = (\hat{y} - y)$$

9. Kesimpulan
Magic terjadi karena:

- Turunan sigmoid: ∂ŷ/∂z = ŷ(1-ŷ)
- Faktor ŷ(1-ŷ) dari sigmoid **persis membatalkan** denominator dari ∂L/∂ŷ
- Hasil akhir menjadi sangat sederhana dan elegant!

Inilah mengapa Binary Cross-Entropy adalah partner sempurna untuk sigmoid dalam klasifikasi biner! 🎯

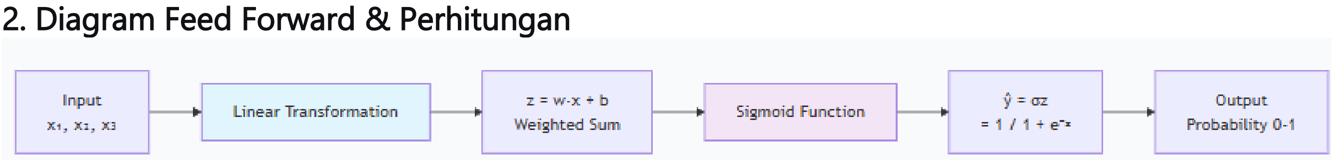
1. Diagram Arsitektur Single Perceptron



Notasi:

- x_1, x_2, x_3: Input features
- w_1, w_2, w_3: Weights
- b: Bias
- z: Weighted sum
- ŷ: Predicted probability
- y: True label (0 atau 1)

2. Diagram Feed Forward & Perhitungan



Rumus Feed Forward:

Step 1: Linear Combination

```text
z = w₁x₁ + w₂x₂ + w₃x₃ + b
```

Step 2: Sigmoid Activation

```text
ŷ = σ(z) = 1 / (1 + e^(-z))
```

Step 3: Output Interpretation

```text
If ŷ ≥ 0.5: Predict Class 1
If ŷ < 0.5: Predict Class 0
```

Contoh Numerik:

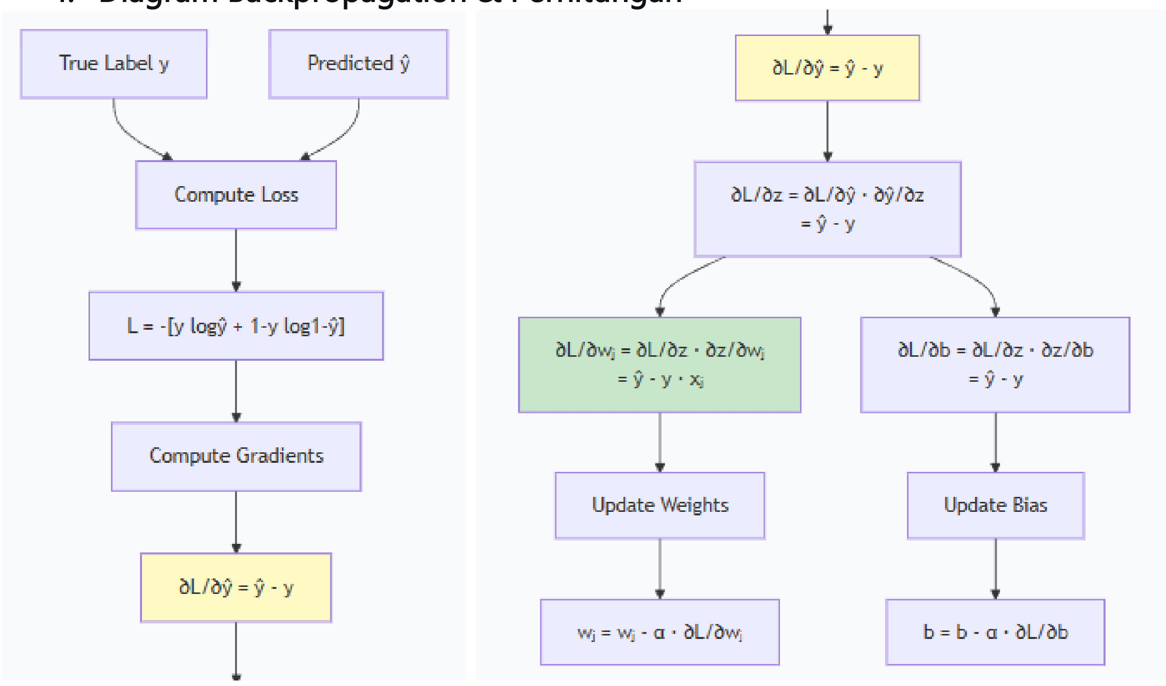
```text
Input: x = [2.0, 1.0, 0.5]
Weights: w = [0.5, -0.3, 0.8]
Bias: b = 0.1

z = (0.5×2.0) + (-0.3×1.0) + (0.8×0.5) + 0.1
  = 1.0 - 0.3 + 0.4 + 0.1 = 1.2
```

$\hat{y} = \sigma(1.2) = 1 / (1 + e^{-1.2}) \approx 0.77$

Prediction: Class 1 (since $0.77 \geq 0.5$)

4. Diagram Backpropagation & Perhitungan



Rumus Backpropagation:

Gradients Calculation:

text

$\partial L/\partial \hat{y} = \hat{y} - y$

$\partial \hat{y}/\partial z = \hat{y}(1-\hat{y})$

$\partial z/\partial w_j = x_j$

$\partial z/\partial b = 1$

Chain Rule:

text

$\partial L/\partial w_j = \partial L/\partial \hat{y} \cdot \partial \hat{y}/\partial z \cdot \partial z/\partial w_j$

$\quad = (\hat{y} - y) \cdot \hat{y}(1-\hat{y}) \cdot x_j$

$\quad = (\hat{y} - y) \cdot x_j$ ← Simplified!

Parameter Update:

text

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

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

Contoh Numerik Backprop:

text

True label: $y = 1$

Predicted: $\hat{y} = 0.77$

Input: $x = [2.0, 1.0, 0.5]$

Learning rate: $\alpha = 0.1$

$\partial L/\partial \hat{y} = 0.77 - 1 = -0.23$

Gradients:

$\partial L/\partial w_1 = -0.23 \times 2.0 = -0.46$

$\partial L/\partial w_2 = -0.23 \times 1.0 = -0.23$

$\partial L/\partial w_3 = -0.23 \times 0.5 = -0.115$

$\partial L/\partial b = -0.23 \times 1 = -0.23$

Update:

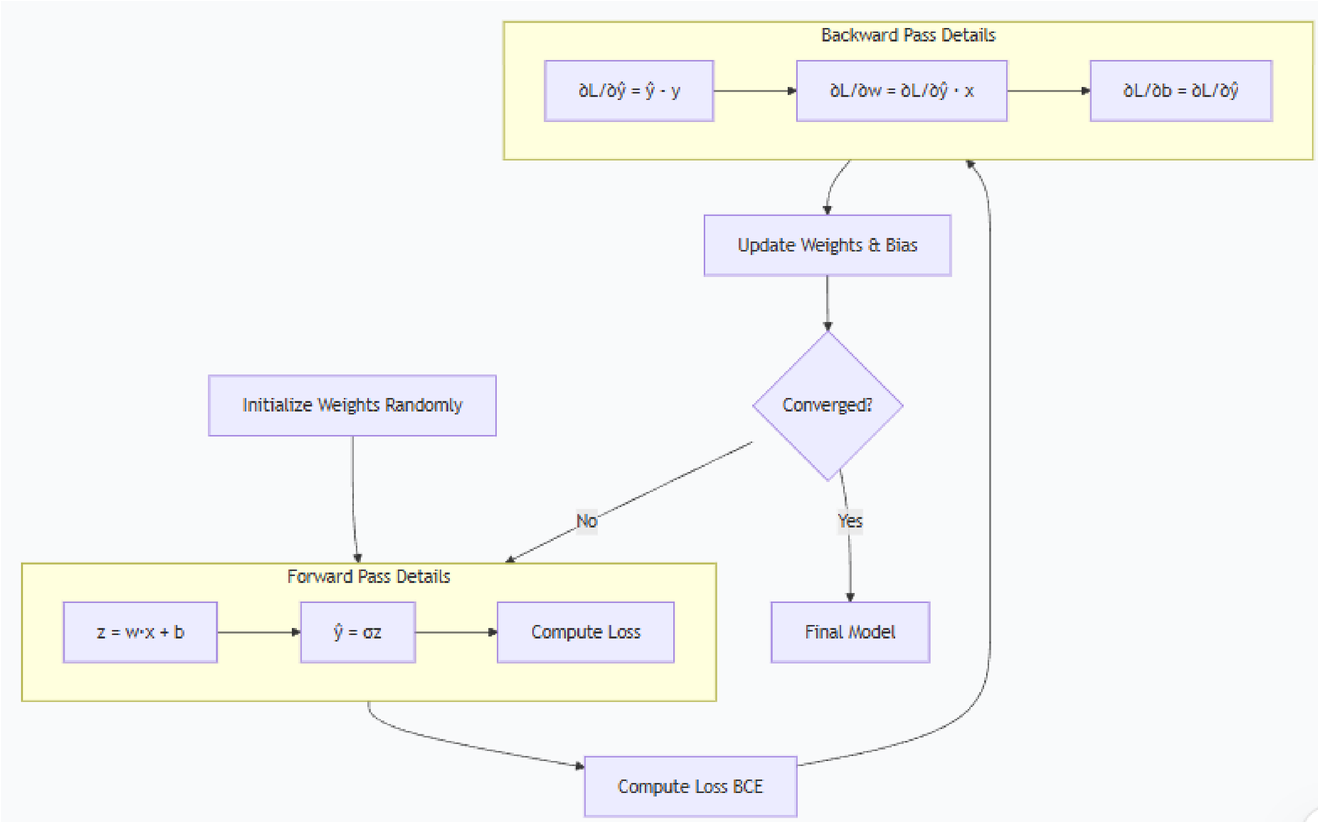
$w_1 = 0.5 - 0.1 \times (-0.46) = 0.5 + 0.046 = 0.546$

$w_2 = -0.3 - 0.1 \times (-0.23) = -0.3 + 0.023 = -0.277$

$w_3 = 0.8 - 0.1 \times (-0.115) = 0.8 + 0.0115 = 0.8115$

$b = 0.1 - 0.1 \times (-0.23) = 0.1 + 0.023 = 0.123$

4. Diagram Lengkap Training Loop



🎯 Key Takeaways:

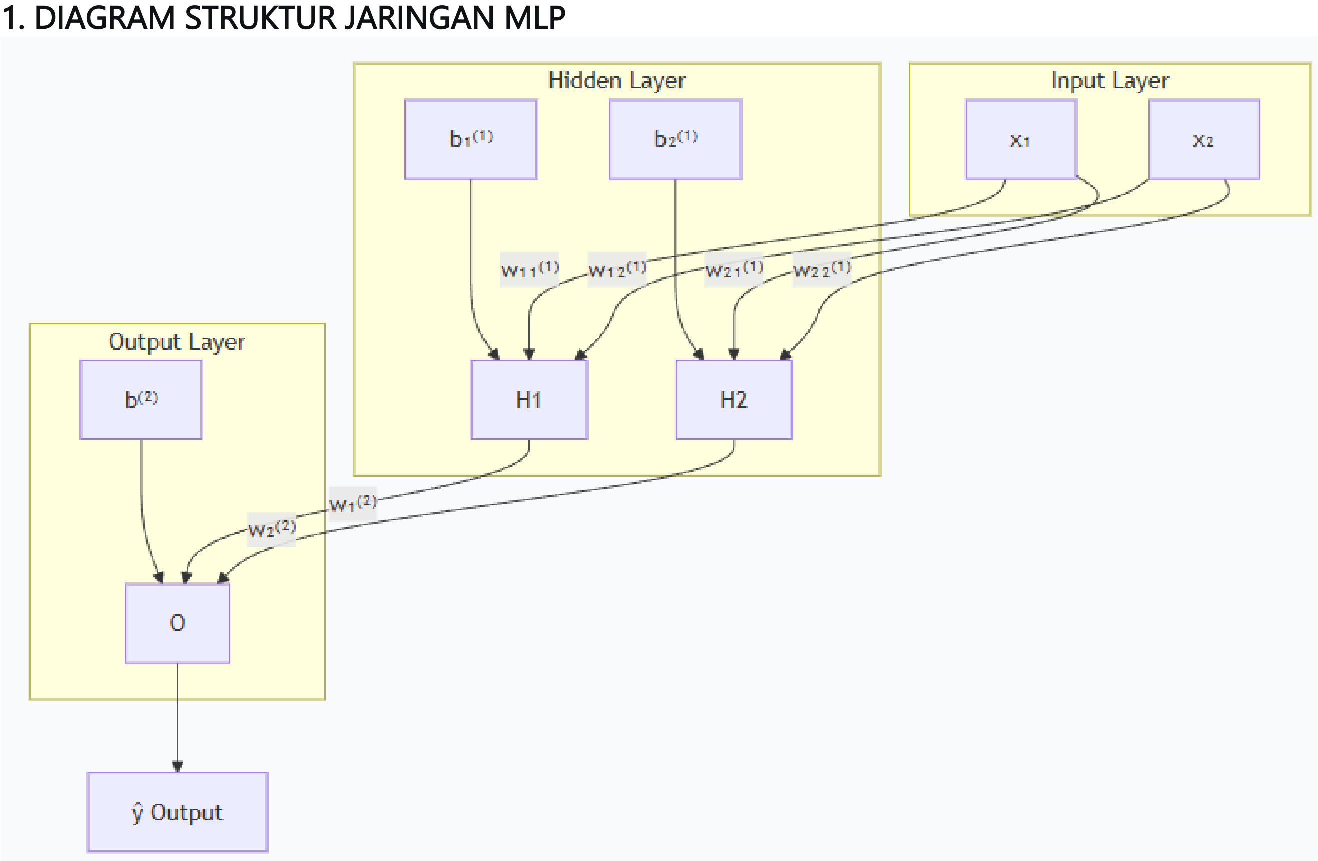
1. **Feed Forward**: Input → Weighted Sum → Sigmoid → Output
2. **Backpropagation**: Loss → Gradients → Update Weights
3. **Sigmoid**: Mengubah output menjadi probability (0-1)
4. **BCE Loss**: Optimal untuk klasifikasi biner
5. **Gradients**: Sederhana karena $(\hat{y} - y) \cdot x$

Single Perceptron adalah building block fundamental dari Deep Learning! 🚀

MLP

Berikut adalah penjelasan MLP (Multi-Layer Perceptron) secara bertahap menggunakan diagram untuk memudahkan pemahaman:

1. DIAGRAM STRUKTUR JARINGAN MLP

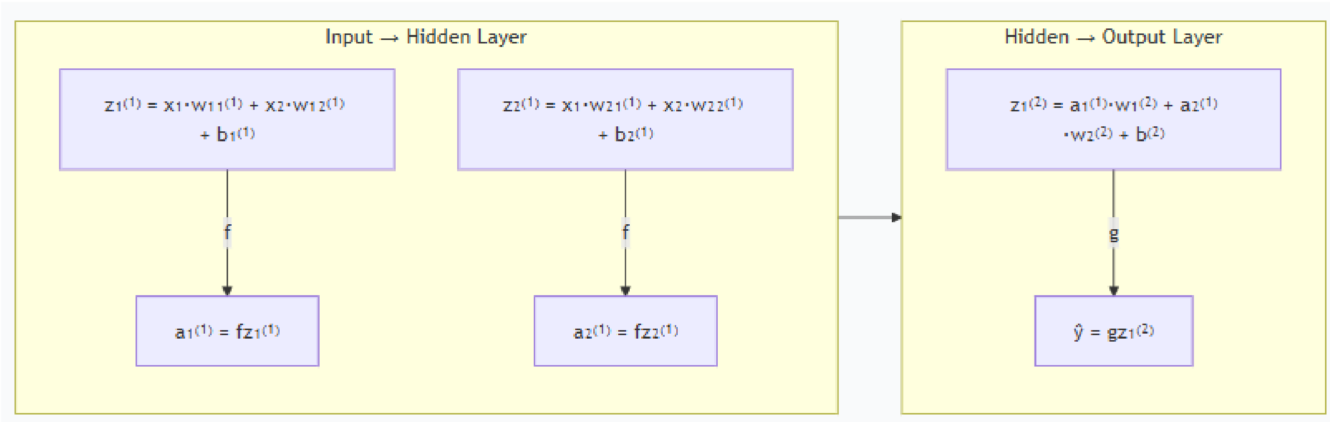


Keterangan:

- **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)

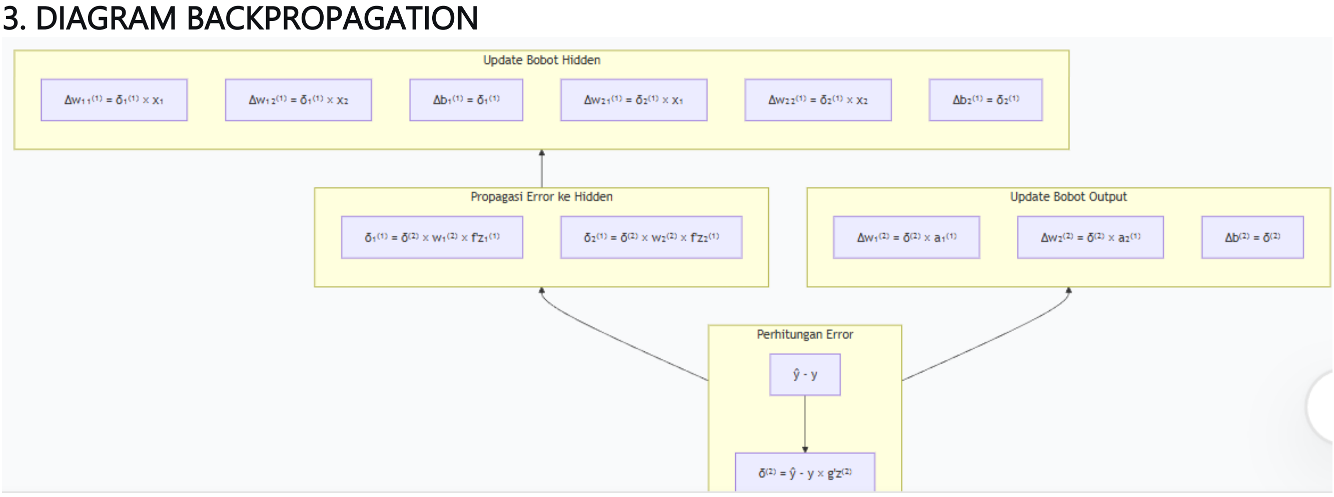
2. DIAGRAM FEED FORWARD



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

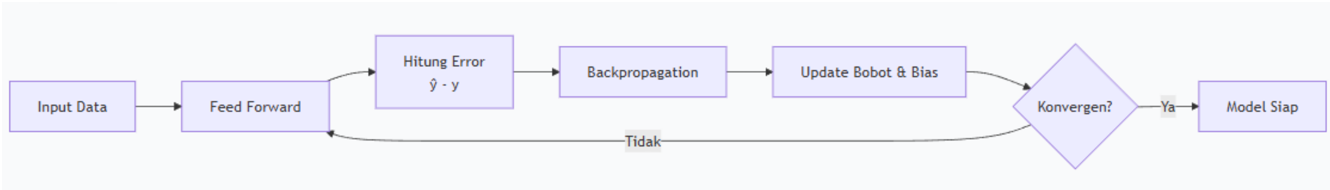
$\delta_1^{(1)} = (\delta^{(2)} \times w_1^{(2)}) \times f'(z_1^{(1)})$
$\delta_2^{(1)} = (\delta^{(2)} \times w_2^{(2)}) \times f'(z_2^{(1)})$

D. Update Bobot Hidden:

math

$\Delta w_{11}^{(1)} = \delta_1^{(1)} \times x_1$
$\Delta w_{12}^{(1)} = \delta_1^{(1)} \times x_2$
$\Delta b_1^{(1)} = \delta_1^{(1)}$
$\Delta w_{21}^{(1)} = \delta_2^{(1)} \times x_1$
$\Delta w_{22}^{(1)} = \delta_2^{(1)} \times x_2$
$\Delta b_2^{(1)} = \delta_2^{(1)}$

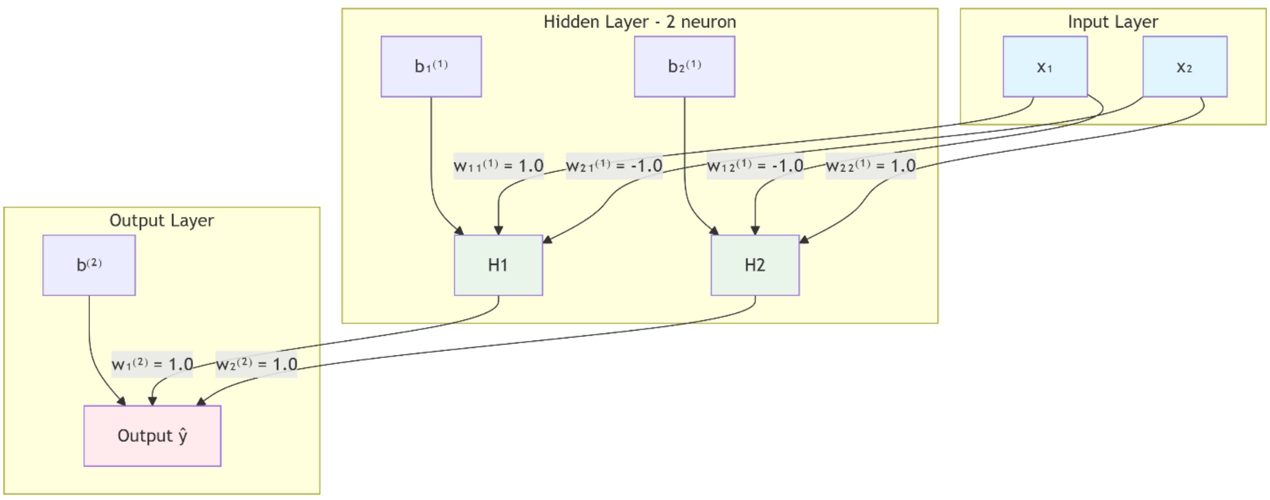
SIKLUS LENGKAP MLP



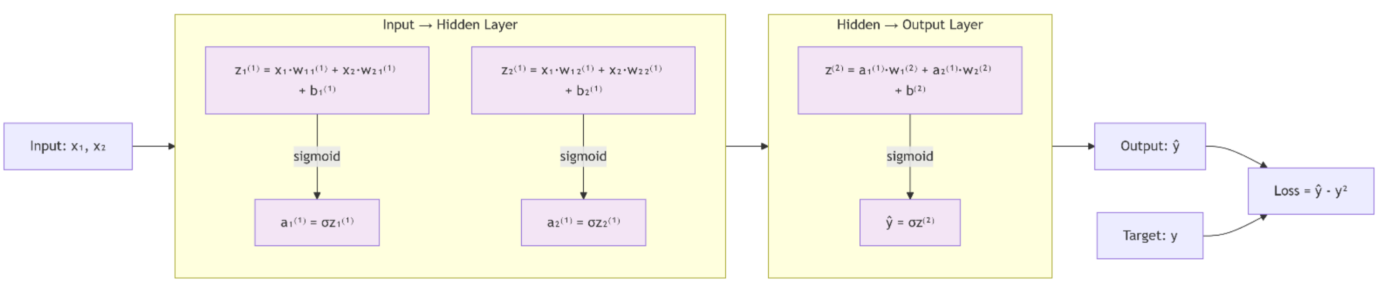
Dengan diagram-diagram ini, proses MLP menjadi lebih visual dan mudah dipahami tahap demi tahap!

Berikut diagram untuk MLP XOR dalam 3 tahap:

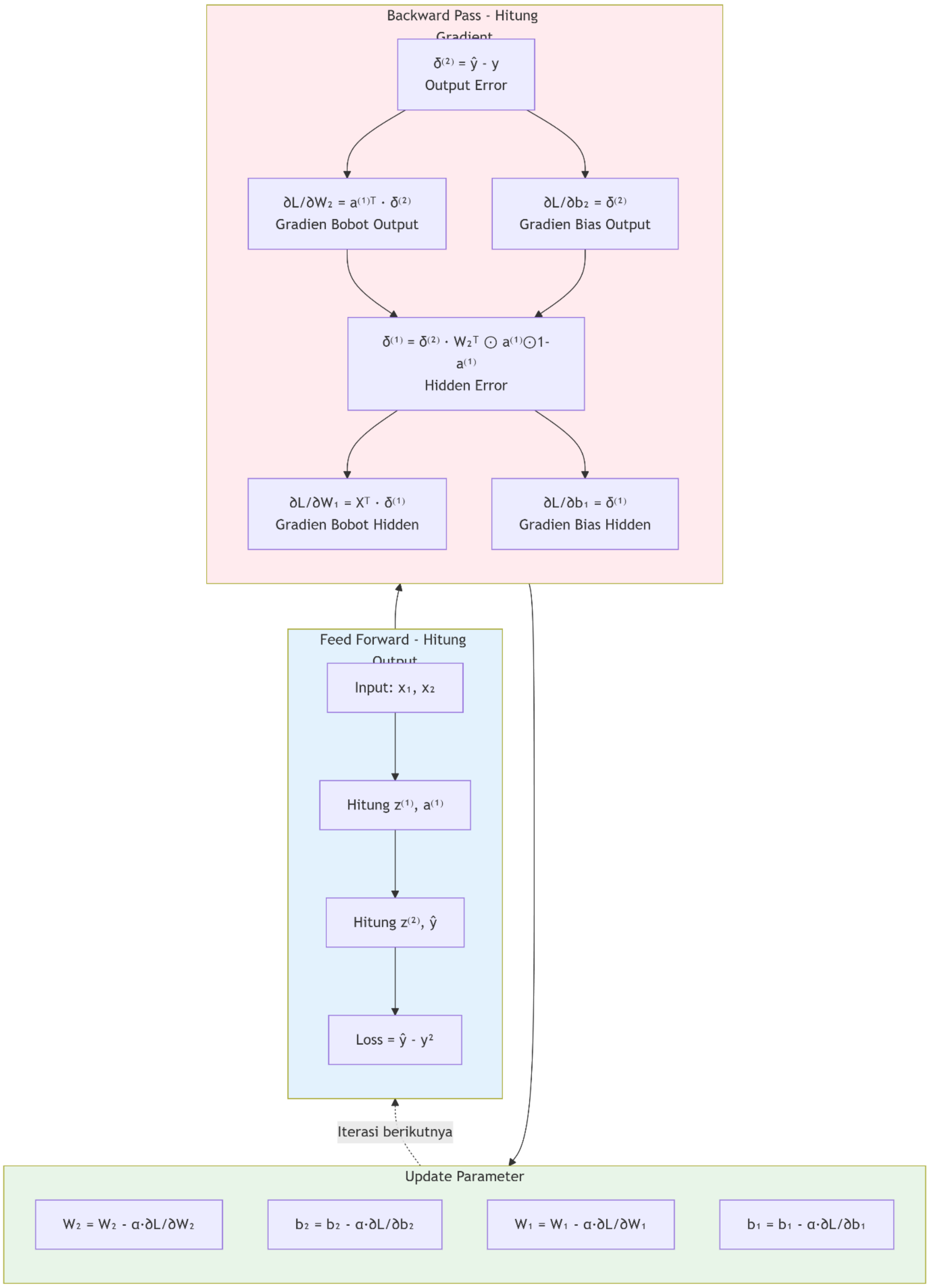
1. DIAGRAM ARSITEKTUR MLP UNTUK XOR



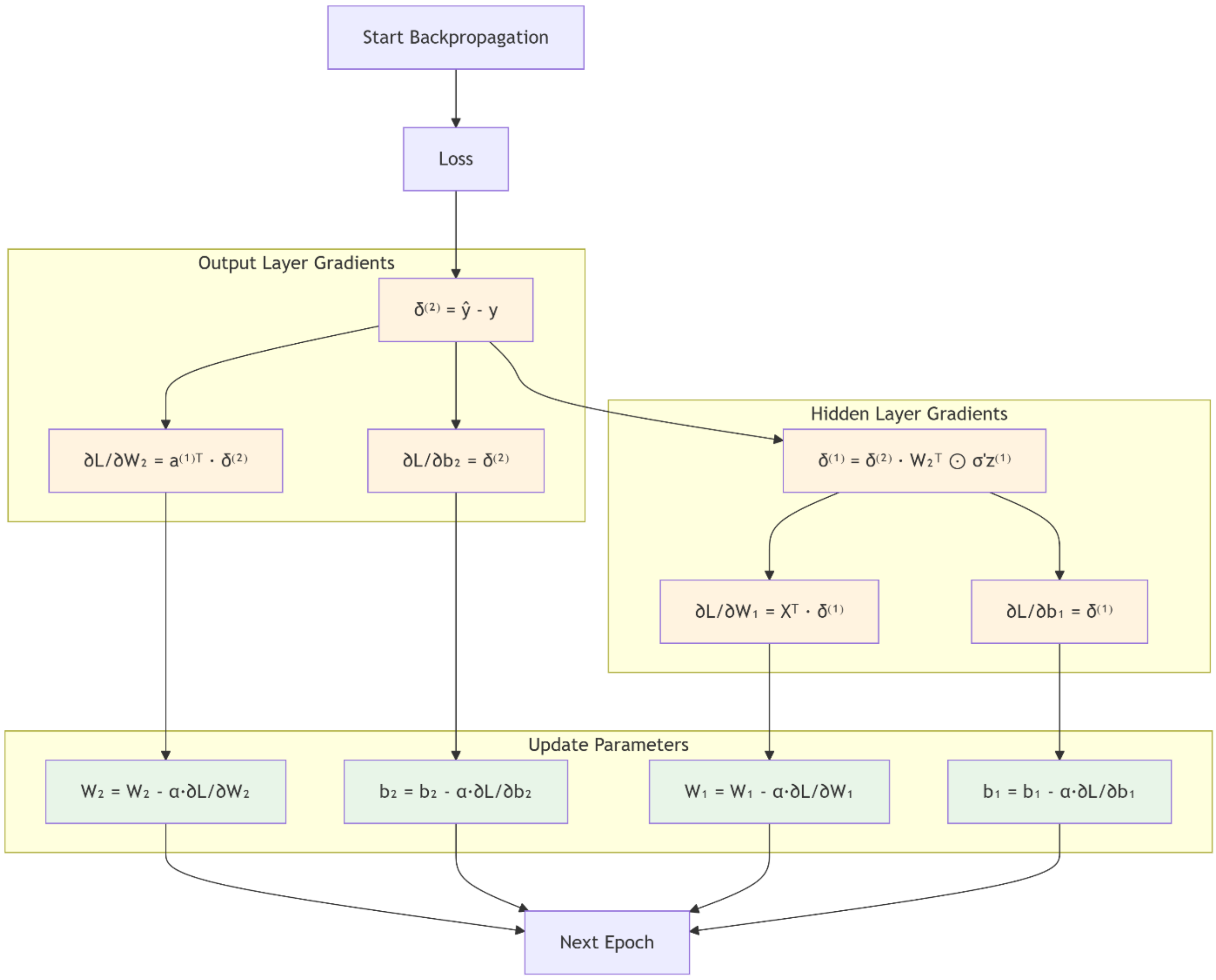
2. DIAGRAM FEED FORWARD



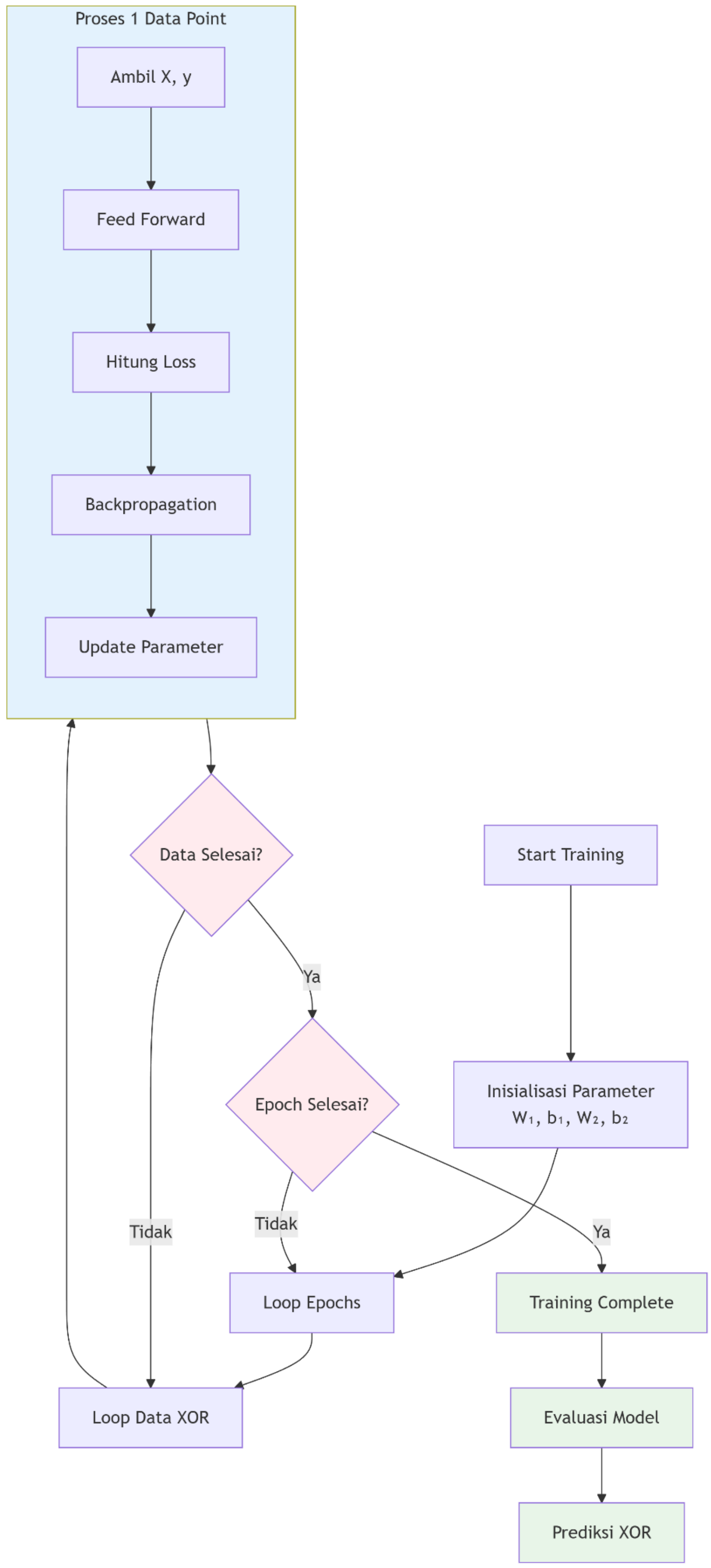
3. DIAGRAM BACKPROPAGATION



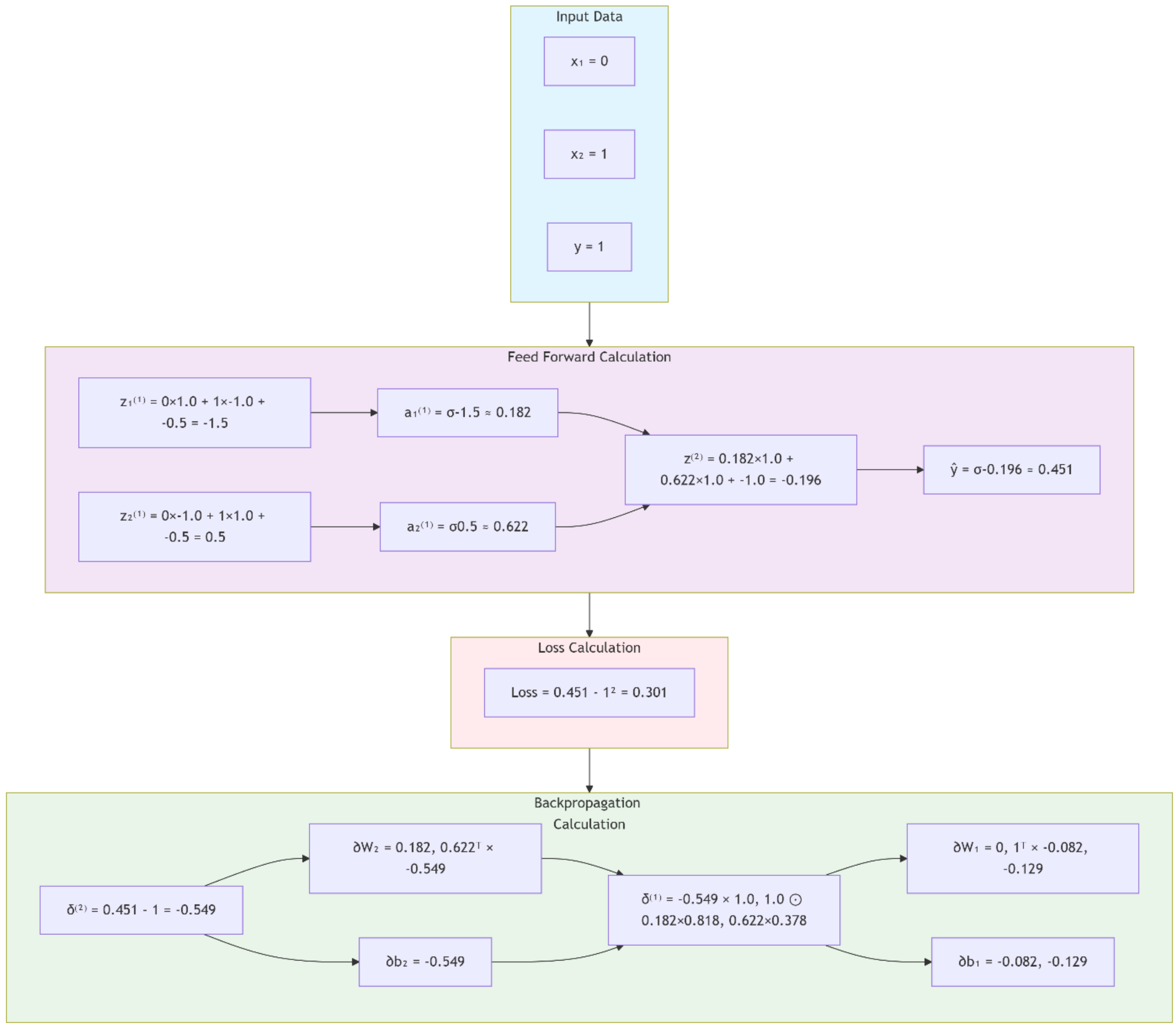
4. DIAGRAM DETAIL BACKPROPAGATION FLOW



5. DIAGRAM ALUR TRAINING LENGKAP



6. DIAGRAM CONTOH PERHITUNGAN UNTUK INPUT [0,1]



PENJELASAN DIAGRAM:

1. **Diagram 1**: Arsitektur jaringan dengan bobot awal
2. **Diagram 2**: Alur feed forward dari input ke output
3. **Diagram 3**: Proses backpropagation lengkap
4. **Diagram 4**: Detail perhitungan gradient
5. **Diagram 5**: Alur training keseluruhan
6. **Diagram 6**: Contoh perhitungan numerik untuk input [0,1]

Diagram-diagram ini menunjukkan bagaimana MLP belajar pola XOR melalui iterasi feed forward dan backpropagation! 🚀