INSTITUT TEKNOLOGI SUMATERA

WORKSHOP DEEP LEARNING · SESI 1

Artificial Neural Network

Struktur, Fungsi Aktivasi, *Feed Forward* & *Backpropagation* untuk Kasus Telekomunikasi

Martin C.T. Manullang
Workshop Deep Learning untuk Mahasiswa Teknik Telekomunikasi — Institut Teknologi Sumatera
Semester Ganjil 2026/2027

mctm.web.id/dl-workshop

Capaian Pembelajaran Sesi Ini

Apa yang bisa kamu lakukan setelah 120 menit ini?

Setelah sesi ini, peserta mampu:

1. Menjelaskan **struktur ANN**: neuron, bobot, bias, dan lapisan.
2. Membedakan **fungsi aktivasi** dan memilihnya sesuai kebutuhan.
3. Menghitung ***feed forward*** dan memahami intuisi ***backpropagation***.
4. Membangun & melatih ANN dengan **PyTorch** untuk **deteksi intrusi jaringan**.

Menit	Agenda
0–10	Motivasi: AI di telekomunikasi
10–30	Struktur ANN
30–45	Fungsi aktivasi
45–60	*Feed forward* & *loss*
60–80	*Backpropagation*
80–90	Studi kasus NSL-KDD
90–120	Hands-on Notebook PyTorch

Peta Materi Hari Ini

Dari satu neuron sampai model deteksi intrusi

- Motivasi
- Struktur ANN
- Fungsi Aktivasi
- Feed Forward
- Backpropagation
- Studi Kasus: Deteksi Intrusi
- Hands-on
- Penutup

BAGIAN 1

Motivasi

AI di Dunia Telekomunikasi

Jaringan modern menghasilkan data dalam jumlah masif

1 Deteksi Intrusi

Mengenali koneksi berbahaya (DoS, *port scan*) di trafik jaringan. **Kasus hari ini.**

2 Prediksi QoS

Memperkirakan *throughput*, *latency*, atau kualitas link dari hasil *drive test*.

3 Prediksi Trafik

Meramal beban sel/ BTS per jam untuk alokasi sumber daya. **Sesi 2 (LSTM).**

4 Lapisan Fisik

Klasifikasi modulasi, estimasi kanal, dan *beamforming* di 5G/6G.

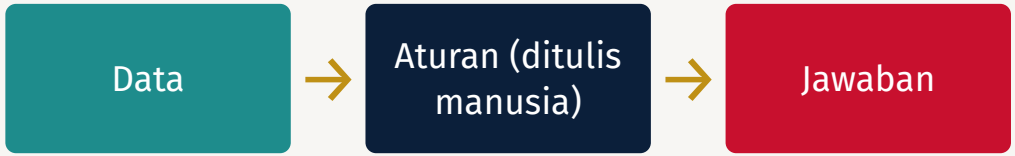
Masalahnya

Pola di data ini **rumit & berdimensi tinggi** — sulit ditulis sebagai aturan `if-else` manual. Kita butuh model yang **belajar sendiri dari data**.

Dari Aturan ke Belajar

Pemrograman tradisional vs machine learning

Pemrograman tradisional




```
if snr > 20 and rssi > -80:
    kualitas = "baik"
```

Mudah untuk 2 fitur. Bagaimana dengan **41 fitur** koneksi jaringan?

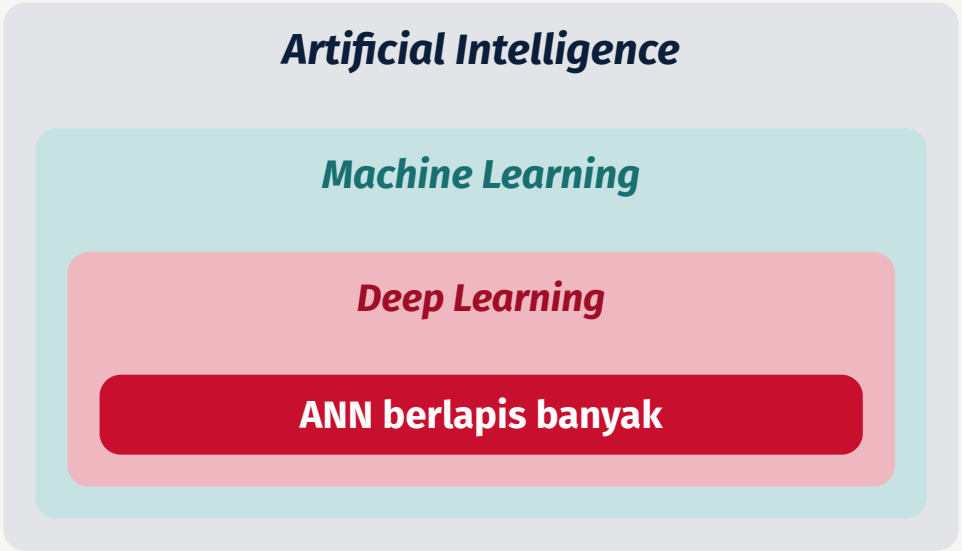
Machine learning

Ide kunci

Kita memberi **contoh berlabel** (koneksi → normal/serangan). Model mencari sendiri aturan terbaik, yang tersimpan sebagai **bobot** (*weights*).

AI, ML, dan Deep Learning

Posisi neural network *dalam peta kecerdasan buatan*



- **AI**: sistem yang meniru kecerdasan manusia.
- **ML**: AI yang **belajar dari data**, bukan diprogram eksplisit.
- ***Deep Learning***: ML memakai ***neural network* dengan banyak lapisan**.

Hari ini

Kita bangun fondasinya: **ANN** (*Artificial Neural Network*) atau **MLP** (*Multi-Layer Perceptron*).

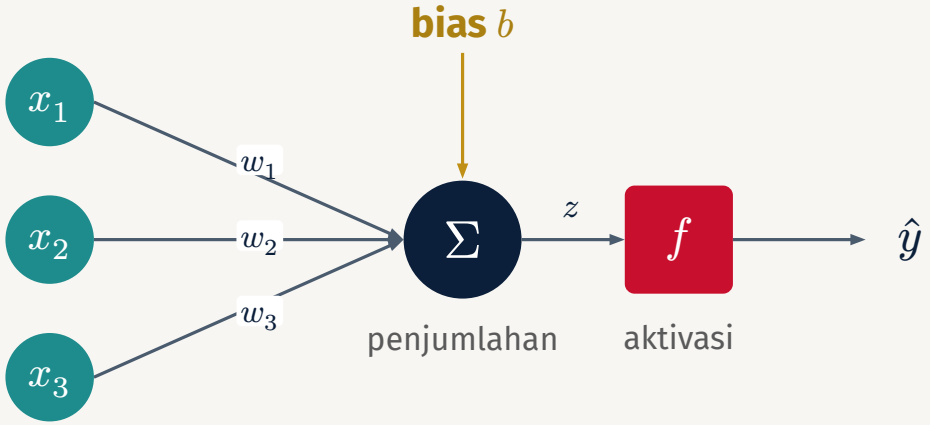
BAGIAN 2

Struktur ANN

Neuron Biologis vs Buatan

Inspirasi dari otak, disederhanakan jadi matematika

Biologis	Buatan	Peran
Dendrit	Input x_i	Menerima sinyal
Sinapsis	Bobot w_i	Kuat/lemahnya sinyal
Soma	Penjumlahan Σ	Mengumpulkan sinyal
Ambang aktivasi	Bias b + fungsi f	Memutuskan “menyala”
Akson	Output $\hat{y}$	Meneruskan hasil



Catatan: neuron buatan **hanya terinspirasi**, bukan simulasi otak yang sebenarnya.

Anatomi Neuron Buatan

Dua langkah: jumlahkan berbobot, lalu aktivasi

Langkah 1 — Penjumlahan berbobot

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

$$z = \sum_{i=1}^{n} w_i x_i + b$$

Langkah 2 — Fungsi aktivasi

$$\hat{y} = f(z)$$

- **Input** x_i: fitur data, mis. SNR, RSSI, jumlah paket.
- **Bobot** w_i: seberapa **penting** fitur tersebut (bisa negatif).
- **Bias** b: menggeser ambang keputusan.
- **Aktivasi** f: mengubah z jadi output (mis. peluang 0–1).

Yang dipelajari

Saat *training*, yang diubah-ubah hanyalah w **dan** b. Itulah “pengetahuan” model.

Normalisasi Input Dulu

Samakan skala fitur sebelum masuk neuron

Fitur telekomunikasi punya satuan & rentang berbeda:

- SNR: 0 s.d. 30 dB
- RSSI: -100 s.d. -40 dBm
- `src_bytes`: 0 s.d. jutaan *byte*

Tanpa normalisasi, fitur berangka besar akan **mendominasi** dan *training* jadi tidak stabil.

Min-max scaling ke rentang 0–1

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Contoh

SNR $= 24$ dB $\rightarrow x_1 = \frac{24-0}{30-0} = \boxed{0.8}$

RSSI $= -64$ dBm $\rightarrow x_2 = \frac{-64+100}{-40+100} = \boxed{0.6}$

Contoh: Kualitas Link Radio

Satu neuron memutuskan link baik *atau* buruk

Input: $x = (0.8, 0.6)$ Bobot: $w = (2.0, 1.5)$ Bias: $b = -1.5$

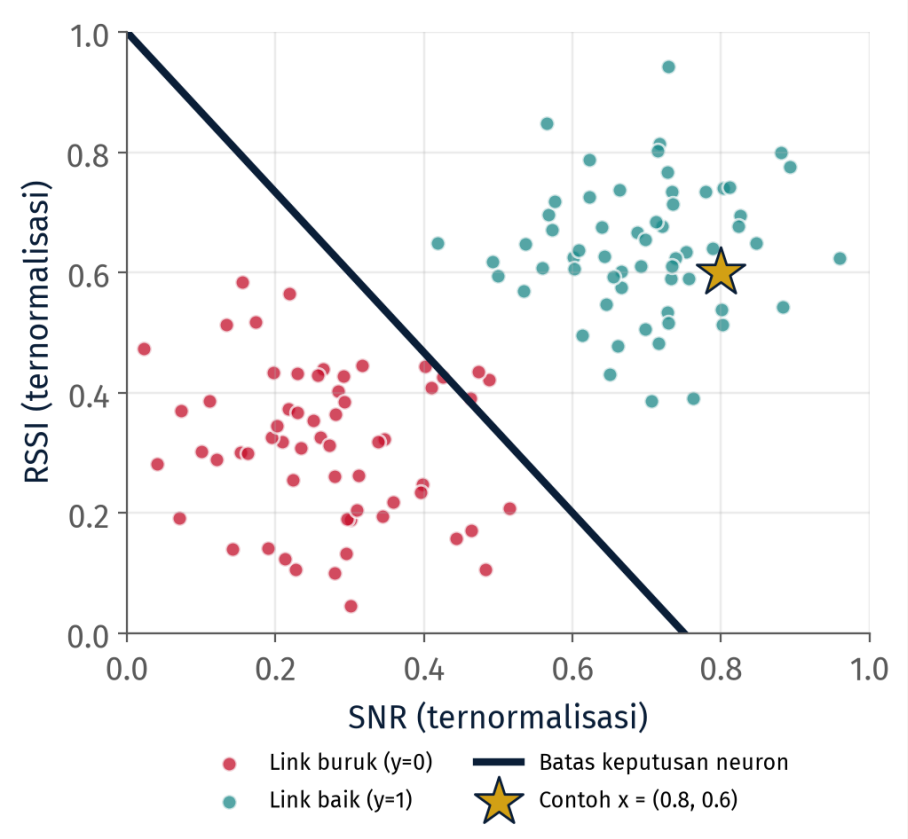
Hitung

$$z = 2.0 \cdot 0.8 + 1.5 \cdot 0.6 - 1.5 = 1.6 + 0.9 - 1.5 = \boxed{1.0}$$

$$\hat{y} = \sigma(1.0) = \frac{1}{1 + e^{-1.0}} = \boxed{0.731}$$

Karena $\hat{y} > 0.5$ → prediksi **link baik** (peluang 73%).

Garis navy di kanan adalah **batas keputusan**: $2.0x_1 + 1.5x_2 - 1.5 = 0$.



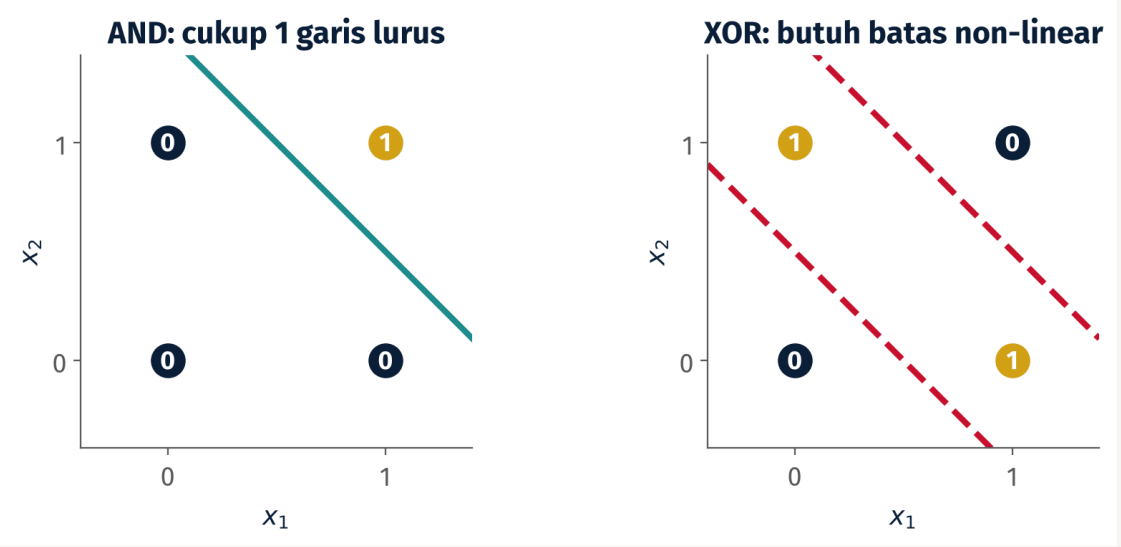
Perceptron dan Keterbatasannya

Neuron pertama (Rosenblatt, 1958) hanya bisa garis lurus

- **Perceptron** = satu neuron dengan aktivasi *step* (output 0 atau 1).
- Batas keputusannya selalu **garis lurus** (*linear*).
- Cocok untuk AND, OR ✓
- **Gagal** untuk XOR × (Minsky & Papert, 1969)

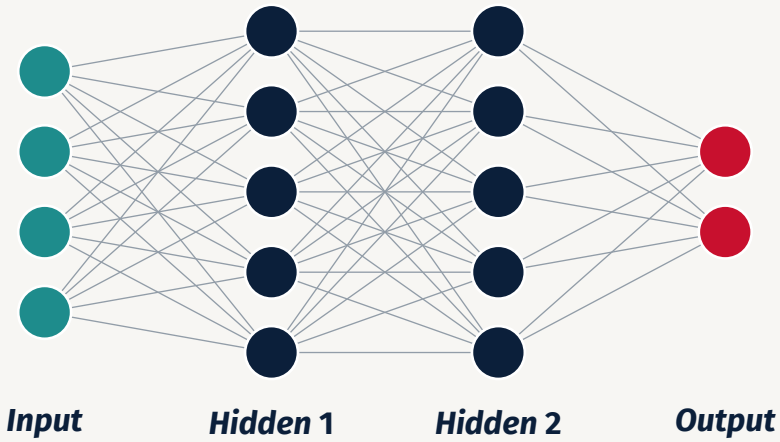
Solusinya

Tumpuk neuron dalam beberapa lapisan + aktivasi **non-linear** → *Multi-Layer Perceptron* (MLP).



Lapisan dalam ANN

Input layer, hidden layer, *dan* output layer



- ***Input layer***: satu neuron per fitur (tidak menghitung apa-apa).
- ***Hidden layer***: tempat pola diekstraksi; bisa lebih dari satu.
- ***Output layer***: jumlah neuron sesuai tugas (1 untuk biner, K untuk K kelas).
- ***Fully connected***: setiap neuron tersambung ke **semua** neuron lapisan berikutnya.

Kenapa "*deep*"?

Semakin banyak *hidden layer*, semakin "dalam" jaringannya.

Menghitung Jumlah Parameter

Setiap sambungan = satu bobot, setiap neuron = satu bias

Rumus per lapisan

$$\text{parameter} = n_{\text{in}} \times n_{\text{out}} + n_{\text{out}}$$

Contoh jaringan kecil 2-2-1:

$$(2 \times 2 + 2) + (2 \times 1 + 1) = 6 + 3 = \boxed{9}$$

Model **hands-on** kita (122-64-32-1):

Lapisan	Hitungan	Parameter
Input → *Hidden* 1	$122 \times 64 + 64$	7.872
Hidden 1 → *Hidden* 2	$64 \times 32 + 32$	2.080
Hidden 2 → *Output*	$32 \times 1 + 1$	33
Total		**9.985**

Hampir 10 ribu angka yang akan disetel otomatis oleh *training*.

BAGIAN 3

Fungsi Aktivasi

Kenapa Perlu Non-linearitas?

Tanpa aktivasi, sedalam apa pun jaringannya tetap linear

Tanpa fungsi aktivasi

$$\hat{y} = W_2(W_1 x) = \underbrace{(W_2 W_1)}_{W'} x$$

Dua lapisan linear = **satu** lapisan linear.
Menumpuk layer jadi percuma.

Dengan fungsi aktivasi

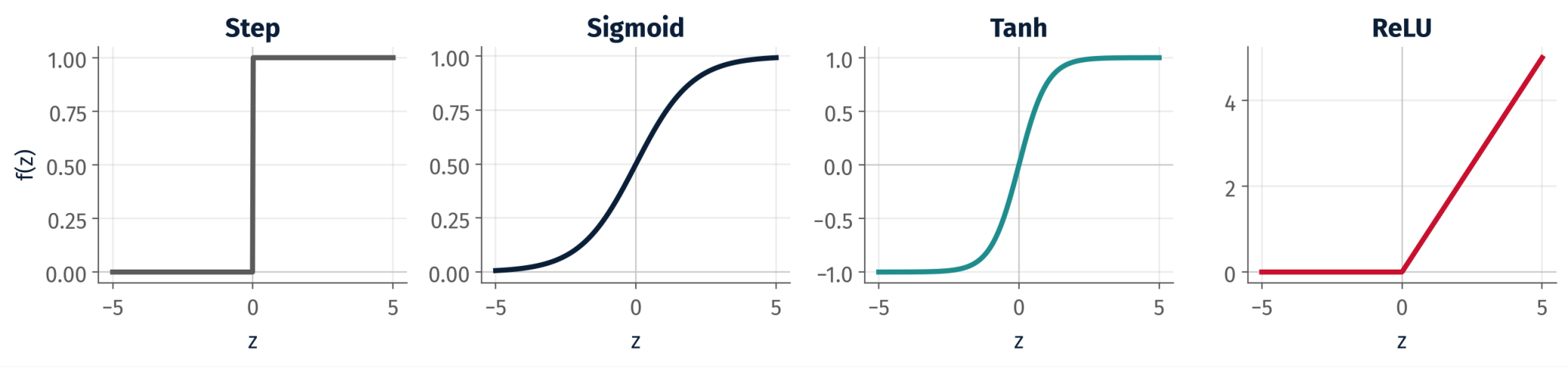
$$\hat{y} = W_2\, f(W_1 x)$$

f non-linear "membengkokkan" ruang fitur, sehingga batas keputusan bisa **melengkung** (menyelesaikan XOR).

Analogi: garis lurus hanya bisa membelah peta jadi dua. Dengan **banyak potongan garis yang ditekuk**, kita bisa menggambar batas wilayah serumit apa pun.

Fungsi Aktivasi Populer

Empat fungsi yang paling sering kamu temui



$$\text{step}(z) = \begin{cases} 1 \text{ jika } z \geq 0 \\ 0 \text{ jika } z < 0 \end{cases} \qquad \sigma(z) = \frac{1}{1+e^{-z}} \qquad \tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \qquad \text{ReLU}(z) = \max(0, z)$$

Membandingkan Fungsi Aktivasi

Kapan memakai yang mana?

Fungsi	Rentang	Kelebihan	Kekurangan	Dipakai di
Step	$\{0, 1\}$	Sederhana, historis	Gradien nol → tidak bisa di-*train*	Perceptron klasik
Sigmoid	$(0, 1)$	Output = peluang	*Vanishing gradient*, lambat	*Output* biner
Tanh	$(-1, 1)$	Terpusat di nol	Masih *vanishing gradient*	RNN/LSTM (Sesi 2)
ReLU	$[0, \infty)$	Cepat, gradien tidak hilang untuk $z > 0$	Neuron bisa "mati" ($z < 0$)	***Hidden layer* (default)**
Softmax	$(0, 1)$, total 1	Distribusi peluang K kelas	Hanya untuk *output*	*Output* multi-kelas

Aturan praktis

Hidden layer → **ReLU**. *Output* biner → **sigmoid**. *Output* multi-kelas → **softmax**. Regresi → **linear** (tanpa aktivasi).

Aktivasi di Lapisan Output

Bentuk output mengikuti jenis tugas

Tugas	Output layer
Regresi (*throughput* Mbps)	1 neuron, linear
Klasifikasi biner (normal/serangan)	1 neuron, sigmoid
Multi-kelas (Normal/DoS/Probe)	K neuron, softmax

Contoh softmax: 3 kelas

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

Kelas	Normal	DoS	Probe
z (*logit*)	2.0	1.0	0.1
e^z	7.389	2.718	1.105
Peluang	**0.659**	0.242	0.099

Prediksi: **Normal** (66%). Total peluang = 1.

Masalah *Vanishing Gradient*

Mengapa ReLU menggantikan sigmoid di hidden layer

- Turunan sigmoid **maksimal 0.25**: $\sigma'(z) = \sigma(z)(1-\sigma(z))$.
- *Backprop* mengalikan turunan dari lapisan ke lapisan.
- 10 lapisan sigmoid: $0.25^{10} \approx 0.000001$ → gradien **menghilang**, lapisan awal berhenti belajar.

ReLU sebagai solusi

$$\mathrm{ReLU}'(z) = \begin{cases} 1 \text{ jika } z > 0 \\ 0 \text{ jika } z \leq 0 \end{cases}$$

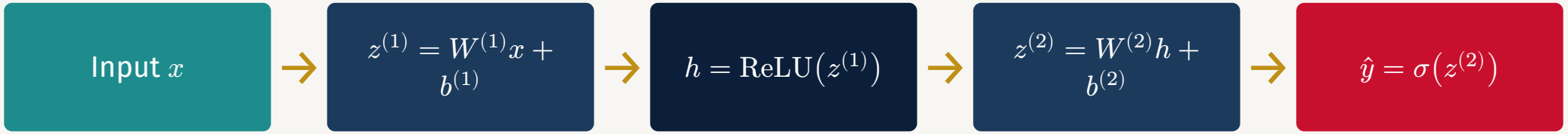
Untuk $z > 0$ gradien diteruskan **utuh** (dikali 1). Inilah salah satu alasan jaringan **dalam** bisa dilatih sejak sekitar tahun 2010.

BAGIAN 4

Feed Forward

Alur *Feed Forward*

Data mengalir satu arah: input → hidden → output



Bentuk matriks

Satu lapisan = **perkalian matriks + bias + aktivasi**:

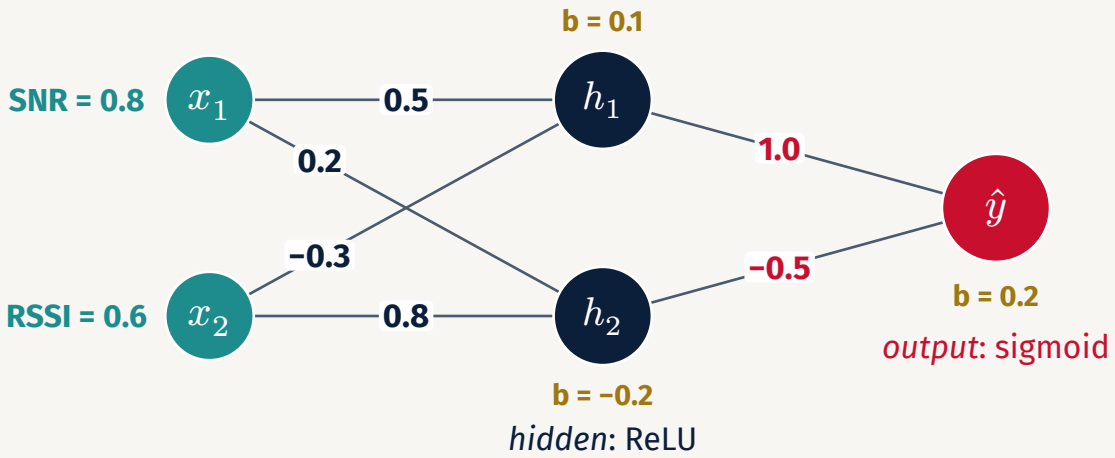
$$h = f(Wx + b)$$

W berukuran $n_{\text{out}} \times n_{\text{in}}$.

- Tiap lapisan mengubah representasi data.
- GPU sangat cepat untuk perkalian matriks → itulah kenapa *deep learning* butuh GPU.
- Di PyTorch: satu baris `nn.Linear(n_in, n_out)`.

Jaringan Contoh 2-2-1

Kita hitung manual: link radio dengan SNR & RSSI



Parameter awal

$$W^{(1)} = \begin{pmatrix} 0.5 & -0.3 \\ 0.2 & 0.8 \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0.1 \\ -0.2 \end{pmatrix}$$

$$w^{(2)} = (1.0, -0.5), \quad b^{(2)} = 0.2$$

Input $x = (0.8, 0.6)$, label sebenarnya $y = 1$ (link **baik**).

Bobot awal biasanya **acak**; di sini dipilih agar mudah dihitung.

Hitung *Hidden Layer*

Langkah 1: dari input ke h_1 dan h_2

Neuron h_1

$$z_1 = 0.5 \cdot 0.8 + (-0.3) \cdot 0.6 + 0.1$$

$$= 0.40 - 0.18 + 0.1 = \boxed{0.32}$$

$$h_1 = \mathrm{ReLU}(0.32) = \boxed{0.32}$$

Neuron h_2

$$z_2 = 0.2 \cdot 0.8 + 0.8 \cdot 0.6 + (-0.2)$$

$$= 0.16 + 0.48 - 0.2 = \boxed{0.44}$$

$$h_2 = \mathrm{ReLU}(0.44) = \boxed{0.44}$$

Karena kedua z positif, ReLU meneruskan nilainya apa adanya. Jika $z < 0$, ReLU akan mengeluarkan 0.

Hitung *Output Layer*

Langkah 2: dari h ke prediksi $\hat{y}$

Neuron output

$$z = 1.0 \cdot 0.32 + (-0.5) \cdot 0.44 + 0.2$$

$$= 0.32 - 0.22 + 0.2 = \boxed{0.30}$$

$$\hat{y} = \sigma(0.30) = \frac{1}{1 + e^{-0.30}} = \boxed{0.574}$$

- Model memprediksi peluang link baik **57.4%**.
- Label sebenarnya $y = 1$ (100% baik).
- Prediksi **benar arah** (> 0.5), tapi **kurang yakin**.

Pertanyaan

Seberapa “salah” prediksi ini? Kita butuh **angka** untuk mengukurnya → ***loss function***.

Mengukur Kesalahan: *Loss*

Satu angka yang ingin kita buat sekecil mungkin

Mean Squared Error (regresi)

$$L_{\text{MSE}} = (y - \hat{y})^2 = (1 - 0.574)^2 = \boxed{0.181}$$

Binary Cross-Entropy (klasifikasi)

$$L_{\text{BCE}} = -[y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})]$$

$$= -\ln(0.574) = \boxed{0.554}$$

Perilaku BCE saat $y = 1$:

$\hat{y}$	L_{BCE}	Arti
0.99	0.01	yakin & benar ✓
0.574	0.554	ragu-ragu
0.50	0.693	menebak
0.01	4.605	yakin & **salah** ×

BCE menghukum keras prediksi yang yakin tapi salah — cocok untuk klasifikasi.

BAGIAN 5

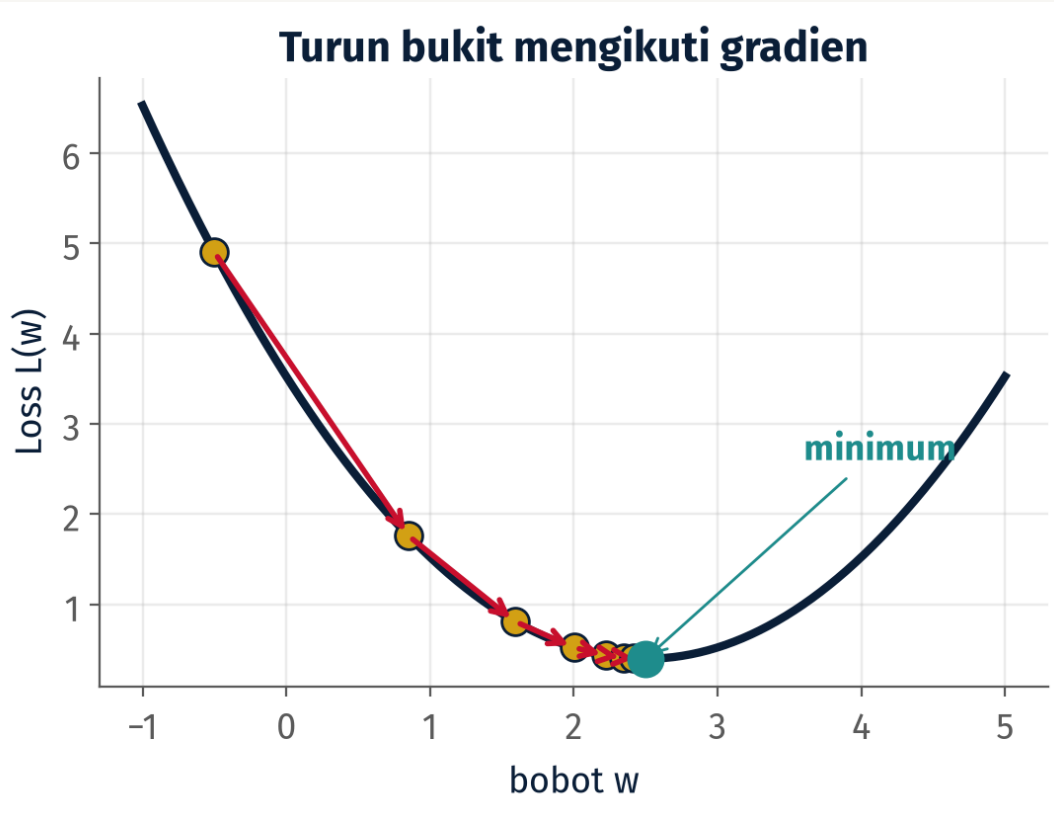
Backpropagation

Prediksi kita 0.574, padahal targetnya 1.

Bobot mana yang harus diubah,
dan seberapa banyak?

Intuisi: Turun Bukit

Gradient descent *mencari lembah* loss *terendah*



Bayangkan kamu di bukit berkabut dan ingin turun ke lembah:

1. Rasakan **kemiringan** tanah di kakimu (**gradien**).
2. Melangkah ke arah **menurun**.
3. Ulangi sampai dasar.

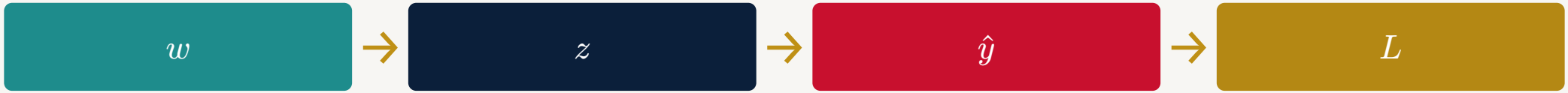
Aturan update

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

η = *learning rate* (besar langkah).

Chain Rule: Kunci Backprop

Menelusuri pengaruh tiap bobot terhadap loss



Mengubah w → mengubah z → mengubah $\hat{y}$ → mengubah L (efek domino)

Chain rule

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

Hasil yang indah

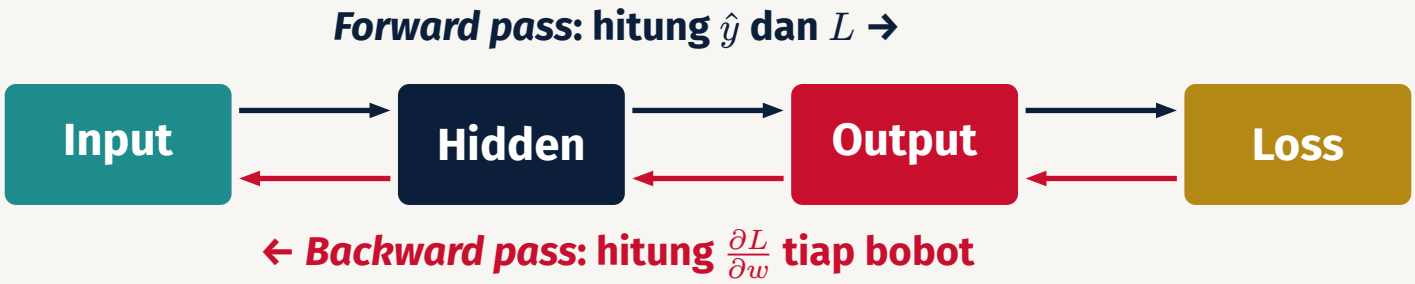
Untuk sigmoid + BCE, dua suku pertama menyederhana menjadi:

$$\frac{\partial L}{\partial z} = \hat{y} - y$$

“Prediksi dikurangi target” — sangat intuitif!

Alur Maju dan Mundur

Forward *menghitung prediksi*, backward *membagi "kesalahan"*



- *Backpropagation* (Rumelhart, Hinton & Williams, 1986) = *chain rule* yang dihitung **mundur** lapis demi lapis, memakai ulang hasil lapisan sesudahnya.
- Di PyTorch cukup satu baris: `loss.backward()` — gradien semua bobot dihitung otomatis (**autograd**).

Backprop di *Output Layer*

Mulai dari kesalahan di ujung jaringan

Sinyal error output

$$\delta_{\text{out}} = \hat{y} - y = 0.574 - 1 = \boxed{-0.426}$$

Tanda negatif artinya: **naikkan** z agar $\hat{y}$ mendekati 1.

Gradien bobot output

$$\frac{\partial L}{\partial w_1^{(2)}} = \delta_{\text{out}} \cdot h_1 = -0.426 \cdot 0.32 = \boxed{-0.136}$$

$$\frac{\partial L}{\partial w_2^{(2)}} = \delta_{\text{out}} \cdot h_2 = -0.426 \cdot 0.44 = \boxed{-0.187}$$

$$\frac{\partial L}{\partial b^{(2)}} = \delta_{\text{out}} = \boxed{-0.426}$$

Pola umum: **gradien bobot = error neuron tujuan × aktivasi neuron asal**.

Backprop ke *Hidden Layer*

Error dibagi mundur sesuai besar bobotnya

Error di tiap neuron hidden

$$\delta_j = \delta_{\text{out}} \cdot w_j^{(2)} \cdot \text{ReLU}'(z_j)$$

$$\delta_1 = -0.426 \cdot 1.0 \cdot 1 = \boxed{-0.426}$$

$$\delta_2 = -0.426 \cdot (-0.5) \cdot 1 = \boxed{0.213}$$

Gradien bobot $W^{(1)}$

$$\frac{\partial L}{\partial W_{ji}^{(1)}} = \delta_j \cdot x_i$$

$$\frac{\partial L}{\partial W^{(1)}} = \begin{pmatrix} -0.340 & -0.255 \\ 0.170 & 0.128 \end{pmatrix}$$

h_2 tersambung dengan bobot **negatif** ke output, jadi error-nya berbalik tanda: model ingin h_2 **mengecil**.

Update Bobot ($\eta = 0.5$)

Satu langkah gradient descent, *lalu cek hasilnya*

Parameter	Lama	Gradien	Baru
$w_1^{(2)}$	1.000	−0.136	**1.068**
$w_2^{(2)}$	−0.500	−0.187	**−0.406**
$b^{(2)}$	0.200	−0.426	**0.413**
$W_{11}^{(1)}$	0.500	−0.340	**0.670**
$W_{22}^{(1)}$	0.800	0.128	**0.736**

Baru = Lama − 0.5 × Gradien (semua 9 parameter di-*update*).

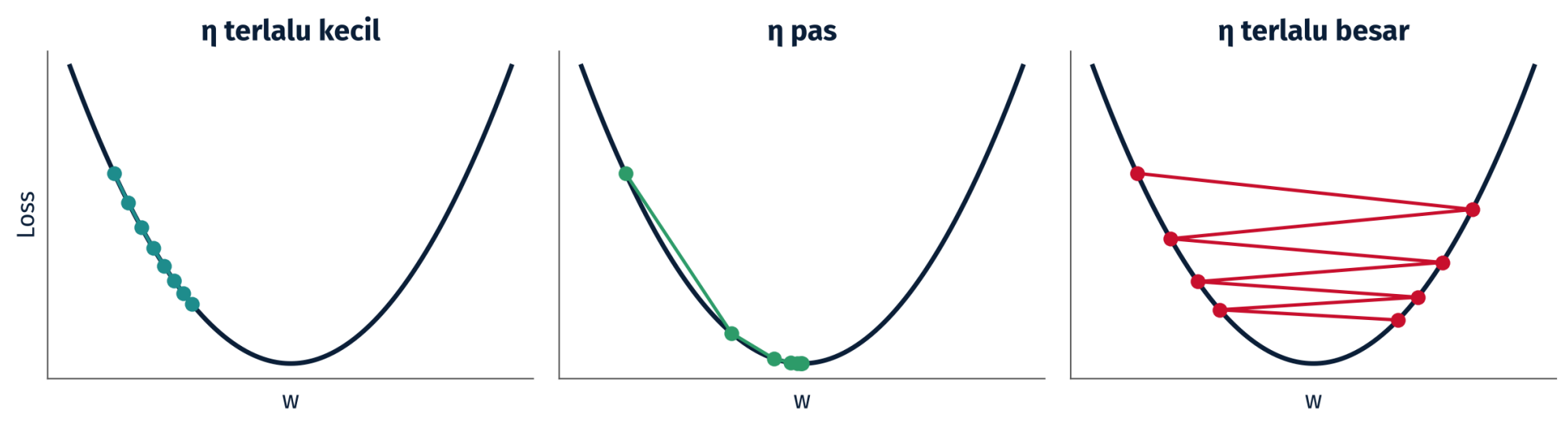
Forward lagi dengan bobot baru

	Sebelum	Sesudah
$\hat{y}$	0.574	**0.753**
L_{BCE}	0.554	**0.283**

Satu langkah saja, *loss* turun hampir **setengahnya**. *Training* = mengulang ini **ribuan kali** pada banyak data.

Memilih *Learning Rate*

Besar langkah menentukan cepat & stabilnya training



Terlalu kecil: sangat lambat, butuh banyak *epoch*.

Pas: turun cepat & stabil ke minimum. Nilai umum: 10^{-3} (Adam).

Terlalu besar: melompat-lompat, *loss* bisa meledak (*diverge*).

Epoch, Batch, dan Iterasi

Cara data dimasukkan ke training *secara bertahap*

Istilah	Arti
Batch	Sekelompok sampel yang diproses sekaligus sebelum 1 kali *update*
Iterasi	Satu kali *update* bobot (1 *batch*)
Epoch	Seluruh data *training* sudah dilihat 1 kali

Contoh hands-on

- Data latih: **100.778** koneksi (80% KDDTrain+)
- *Batch size*: **64**
- Iterasi per *epoch*: $\lceil \frac{100778}{64} \rceil = 1575$
- 10 *epoch* → **15.750** kali *update* bobot

Memakai *mini-batch* = *Stochastic Gradient Descent* (SGD) dan variannya, mis. Adam.

Siklus *Training* ANN

Lima langkah yang diulang di setiap iterasi

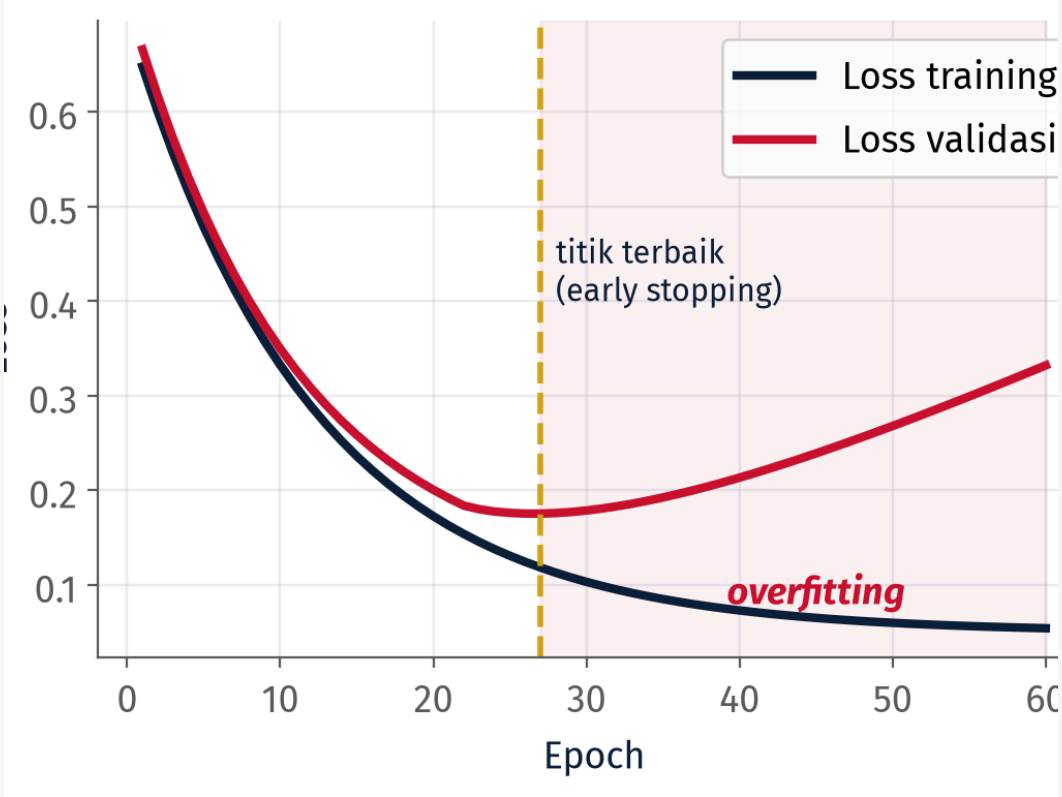
1. Forward prediksi $\hat{y}$ → **2. Loss** bandingkan dgn y → **3. Backward** hitung gradien → **4. Update** $w \leftarrow w - \eta \nabla L$ → **5. Reset** nol-kan gradien

```
for xb, yb in train_loader:   # tiap batch
    y_hat = model(xb)           # 1. forward
    loss = loss_fn(y_hat, yb)   # 2. loss
    optimizer.zero_grad()       # 5. reset
    loss.backward()             # 3. backward
    optimizer.step()            # 4. update
```

- Semua perhitungan manual tadi dilakukan PyTorch **otomatis**.
- `zero_grad()` wajib: PyTorch **menjumlahkan** gradien jika tidak di-*reset*.
- Kita cukup memahami **konsepnya** dan memilih arsitektur, *loss*, & *optimizer*.

Waspada *Overfitting*

Model menghafal data latih, gagal di data baru



- Pisahkan data: **train** / **validasi** / **test**.
- *Loss* train terus turun, tapi *loss* validasi naik → **overfitting**.

Cara mengatasi

- *Early stopping*
- *Dropout*
- Regularisasi (*weight decay*)
- Tambah data / sederhanakan model

Coba Sendiri: Visualisasi Interaktif

Lihat neuron belajar secara langsung di browser

1 TensorFlow Playground

playground.tensorflow.org

- Ubah jumlah *hidden layer* & neuron.
- Bandingkan aktivasi ReLU vs sigmoid.
- Coba dataset “XOR” dan “spiral”.
- Amati efek *learning rate*.

2 Backprop Explainer

xnought.github.io/backprop-explainer

- Animasi *forward* & *backward pass*.
- Lihat gradien mengalir mundur.
- Amati kurva *loss* turun per *epoch*.

BAGIAN 6

Studi Kasus: Deteksi Intrusi

Deteksi Intrusi Jaringan

Intrusion Detection System *(IDS) berbasis ANN*

- Operator jaringan memantau jutaan koneksi setiap hari.
- Serangan (DoS, *port scan*, pembobolan *password*) harus terdeteksi **cepat**.
- IDS berbasis aturan (*signature*) sulit mengenali pola serangan baru.

Tugas ANN kita

Diberi 41 fitur sebuah koneksi → prediksi **normal** ($y = 0$) atau **serangan** ($y = 1$). **Klasifikasi biner**.

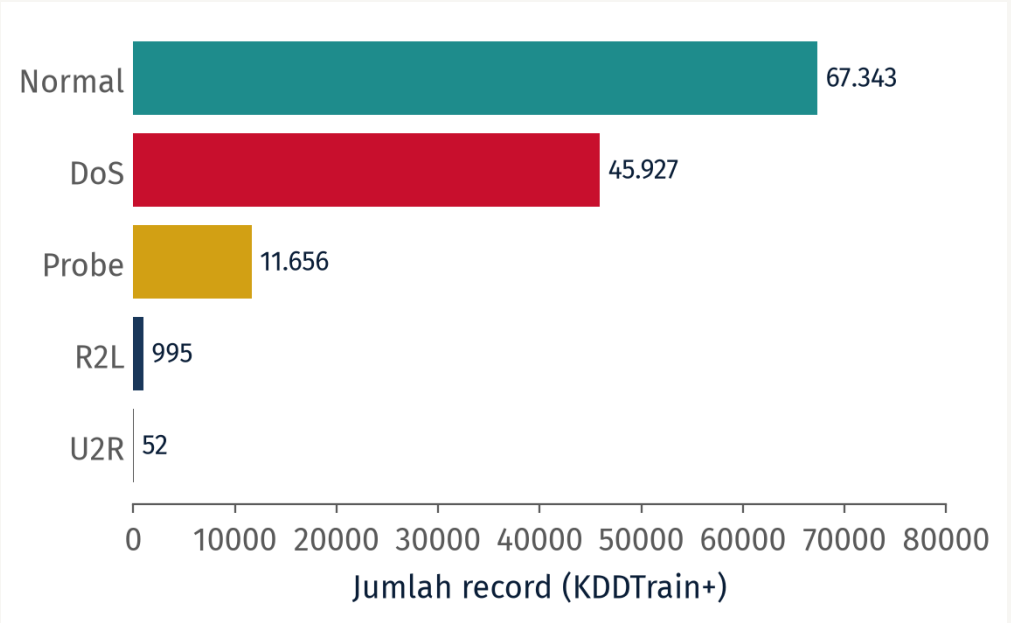
Pola yang sama dipakai di *Network Detection & Response* (NDR) dan *firewall* modern berbasis ML.

Mengenal Dataset NSL-KDD

Benchmark *klasik deteksi intrusi (Tavallaee dkk., 2009)*

Kelompok	Contoh fitur
Dasar	`duration`, `protocol_type`, `service`, `flag`, `src_bytes`, `dst_bytes`
Konten	`num_failed_logins`, `logged_in`, `root_shell`
Trafik	`count`, `srv_count`, `serror_rate`, `dst_host_count`

- **41 fitur**: 38 numerik + 3 kategorikal.
- KDDTrain+: **125.973** record, KDDTest+: **22.544** record.
- Versi perbaikan KDD Cup 99 (duplikat dihapus).



Hands-on: semua kategori serangan digabung jadi label **serangan** (biner).

Kategori Serangan Jaringan

Empat keluarga serangan di NSL-KDD

DoS

Denial of Service: membanjiri server hingga layanan lumpuh.

Contoh: `neptune` (SYN *flood*), `smurf`.

Probe

Memindai jaringan untuk mencari celah.

Contoh: `portsweep`, `ipsweep`, `nmap`.

R2L

Remote to Local: masuk dari luar tanpa akun sah.

Contoh: `guess_passwd`, `ftp_write`.

U2R

User to Root: user biasa menaikkan hak akses jadi *root*.

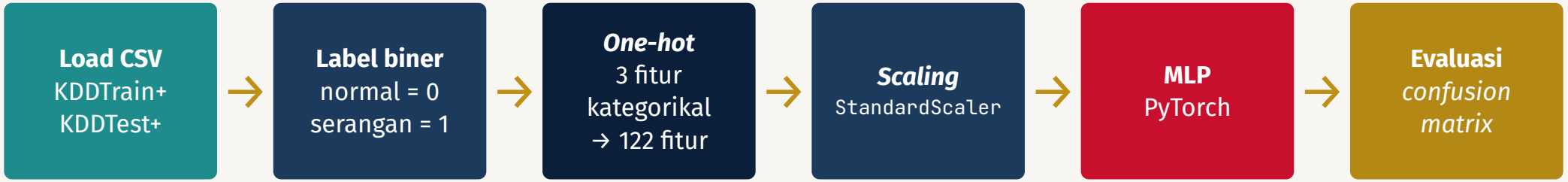
Contoh: `buffer_overflow`.

Tantangan

Data **tidak seimbang**: U2R hanya 52 record dari 125 ribu. Akurasi tinggi belum tentu berarti model bagus → perlu metrik lain.

Pipeline Hands-on

Dari file CSV sampai evaluasi model



One-hot encoding

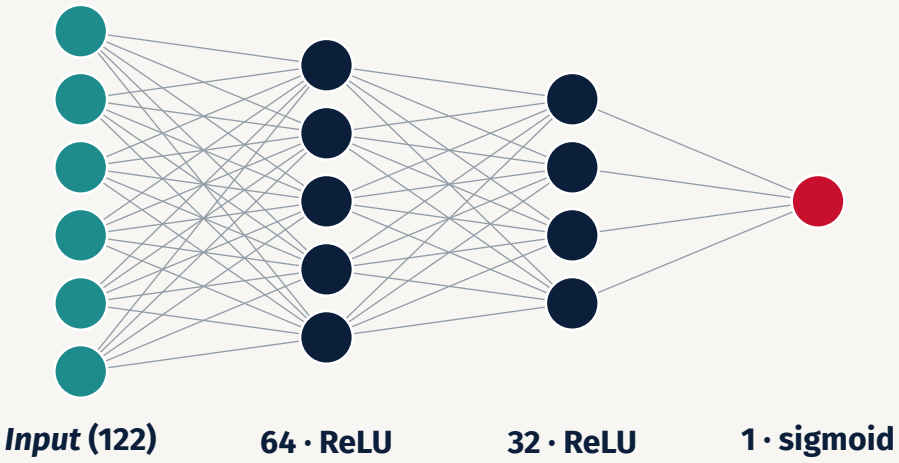
Fitur teks seperti `protocol_type` ∈ {tcp, udp, icmp} diubah jadi 3 kolom 0/1. Neuron hanya bisa menghitung **angka**.

Standardization

$x' = \frac{x - \mu}{\sigma}$ → rata-rata 0, simpangan baku 1. `src_bytes` (jutaan) tidak lagi mendominasi.

Arsitektur MLP Kita

122 input → 64 → 32 → 1 output



Komponen	Pilihan
Hidden aktivasi	ReLU
Output aktivasi	Sigmoid
Loss	BCE
Optimizer	Adam, $\eta = 10^{-3}$
Batch size	64
Epoch	10
Parameter	9.985

ANN dengan PyTorch

Mendefinisikan model cukup beberapa baris

```python
import torch
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(122, 64),    # input -> hidden 1
    nn.ReLU(),
    nn.Linear(64, 32),     # hidden 1 -> hidden 2
    nn.ReLU(),
    nn.Linear(32, 1),      # hidden 2 -> output
)

loss_fn = nn.BCEWithLogitsLoss()  # sigmoid + BCE
optimizer = torch.optim.Adam(model.parameters(),
lr=1e-3)
```

- `nn.Linear(n_in, n_out)` = $Wx + b$ (satu lapisan).
- `nn.ReLU()` = fungsi aktivasi.
- `BCEWithLogitsLoss` menggabungkan **sigmoid + BCE** agar lebih stabil secara numerik.
- `Adam` = varian *gradient descent* dengan *learning rate* adaptif.

Cek jumlah parameter

```python
sum(p.numel() for p in model.parameters())
# 9985
```

Metrik Evaluasi Klasifikasi

Akurasi saja tidak cukup untuk keamanan jaringan

	Prediksi: Serangan	Prediksi: Normal
Asli: Serangan	**TP** = 520	**FN** = 80 serangan lolos!
Asli: Normal	**FP** = 30 alarm palsu	**TN** = 370

Angka ilustrasi (1.000 koneksi uji).

Metrik	Rumus	Nilai
Akurasi	$\frac{TP+TN}{\text{semua}}$	0.890
Precision	$\frac{TP}{TP+FP}$	0.945
Recall	$\frac{TP}{TP+FN}$	0.867
F1	$2 \cdot \frac{P \cdot R}{P+R}$	0.904

Di IDS, ***recall*** penting: setiap FN adalah serangan yang tidak terdeteksi.

BAGIAN 7

Hands-on

Persiapan Lingkungan dengan uv

Satu alat untuk Python, virtual environment, *dan paket*

1. Instal uv (sekali saja)

```
# macOS / Linux
curl -LsSf https://astral.sh/uv/install.sh | sh
# Windows (PowerShell)
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
```

2. Unduh repo & instal dependensi

```
# tautan repo: mctm.web.id/dl-workshop
git clone <URL-repo-workshop>
cd dl-workshop
uv sync   # buat .venv + instal semua paket
```

3. Jalankan Jupyter

```
uv run jupyter lab
```

Kenapa uv?

- Sangat cepat (ditulis dengan Rust).
- `uv sync` membaca `pyproject.toml` → semua peserta punya versi paket yang **sama**.
- Tidak perlu `pip`, `venv`, atau `conda` terpisah.

Lakukan sebelum sesi

Unduhan PyTorch cukup besar (±200 MB–1 GB). Jalankan `uv sync` dengan Wi-Fi yang stabil.

Alur Notebook Hands-on

Isi bagian `TODO` secara bertahap

1 Eksplorasi data

Baca CSV, lihat fitur, hitung distribusi normal vs serangan.

2 Pra-pemrosesan

Label biner, *one-hot encoding*, *standardization*.

3 Bangun model

Definisikan MLP 122-64-32-1 dengan `nn.Sequential`.

4 *Training*

Tulis *training loop*, pantau *loss* train & validasi.

5 Evaluasi

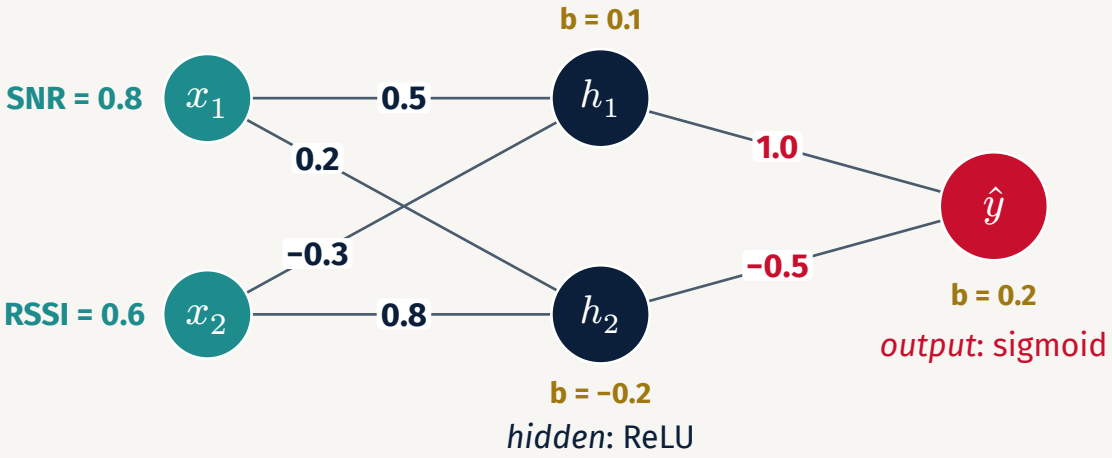
Confusion matrix, *precision*, *recall*, F1 di KDDTest+.

6 Eksperimen

Ubah neuron, aktivasi, *learning rate* — amati efeknya.

Latihan 1: *Forward Pass*

Latihan *Gunakan jaringan 2-2-1 dan bobot yang sama*



Soal

Sebuah link memiliki SNR dan RSSI ternormalisasi $x = (0.2, 0.3)$ dengan label **link buruk** ($y = 0$).

1. Hitung h_1 dan h_2.
2. Hitung $\hat{y}$.
3. Hitung L_{BCE}. Apakah prediksinya benar?

Petunjuk: $\sigma(0.27) \approx 0.567$.

Solusi 1: *Forward Pass*

Pembahasan langkah demi langkah

Hidden layer

$$z_1 = 0.5 \cdot 0.2 - 0.3 \cdot 0.3 + 0.1 = 0.11 \rightarrow h_1 = \boxed{0.11}$$

$$z_2 = 0.2 \cdot 0.2 + 0.8 \cdot 0.3 - 0.2 = 0.08 \rightarrow h_2 = \boxed{0.08}$$

Output layer

$$z = 1.0 \cdot 0.11 - 0.5 \cdot 0.08 + 0.2 = 0.27$$

$$\hat{y} = \sigma(0.27) = \boxed{0.567}$$

Loss ($y = 0$)

$$L_{\text{BCE}} = -\ln(1 - 0.567) = \boxed{0.837}$$

Interpretasi

$\hat{y} > 0.5$ → model memprediksi **baik**, padahal **buruk**. *Loss* besar (0.837 > 0.554) → *backprop* akan mengoreksi bobot lebih kuat. Model belum di-*train*!

Latihan 2: Rancang Arsitektur

Latihan *Dari biner ke multi-kelas*

Soal

Kita ingin memprediksi **5 kelas** NSL-KDD (Normal, DoS, Probe, R2L, U2R) dengan MLP **122-128-64**.

1. Berapa neuron dan fungsi aktivasi di *output layer*?
2. *Loss function* apa yang cocok?
3. Hitung total parameter model.

Kerjakan 3 menit, lalu diskusikan dengan teman sebelahmu.

Solusi 2: Rancang Arsitektur

Pembahasan

1. **5 neuron** + **softmax** (satu peluang per kelas).
2. ***Cross-Entropy Loss*** (versi multi-kelas dari BCE). Di PyTorch: `nn.CrossEntropyLoss()` (sudah termasuk softmax).
3. Lihat tabel di kanan.

Lapisan	Hitungan	Parameter
122 → 128	$122 \times 128 + 128$	15.744
128 → 64	$128 \times 64 + 64$	8.256
64 → 5	$64 \times 5 + 5$	325
Total		**24.325**

BAGIAN 8

Penutup

Ringkasan Sesi 1

Empat ide yang perlu dibawa pulang

1 Struktur

Neuron = $f(\sum w_i x_i + b)$. Neuron disusun dalam lapisan *input*, *hidden*, *output*.

2 Aktivasi

Non-linearitas membuat ANN bisa belajar pola rumit. ReLU di *hidden*, sigmoid/softmax di *output*.

3 *Feed forward & loss*

Hitung prediksi lapis demi lapis, ukur kesalahan dengan MSE atau BCE.

4 *Backprop*

Chain rule menghitung gradien; *gradient descent* memperbarui bobot berulang-ulang.

Sesi 2: RNN & LSTM

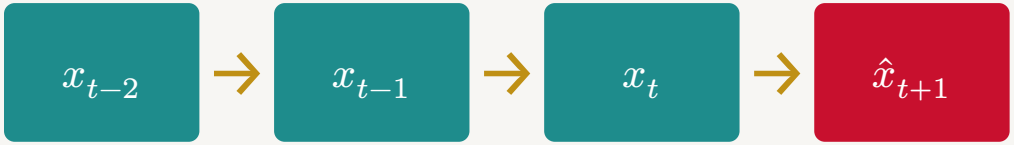
Ketika urutan waktu menjadi penting

Keterbatasan ANN

ANN melihat setiap sampel **terpisah** — tidak punya **memori**. Padahal trafik jaringan jam 20.00 sangat dipengaruhi trafik jam 19.00.

Sesi berikutnya

- *Recurrent Neural Network* (RNN)
- *Long Short-Term Memory* (LSTM)
- Hands-on: **prediksi trafik jaringan / beban sel** (*time series forecasting*)



LSTM “mengingat” pola masa lalu untuk meramal masa depan.

Referensi

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. deeplearningbook.org

Nielsen, M. (2015). *Neural Networks and Deep Learning*. Determination Press. neuralnetworksanddeeplearning.com

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*.

Zhang, C., Patras, P., & Haddadi, H. (2019). Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(3), 2224–2287.

PyTorch Documentation. pytorch.org/docs/stable/nn.html · uv Documentation. docs.astral.sh/uv

Terima Kasih

Saatnya *hands-on* — buka notebook Sesi 1!

mctm.web.id/dl-workshop