Pertemuan 12 — *Deep Neural Network*

Kedalaman, jalur residual, dan *transfer learning*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Menyambung dari Pertemuan 11
2. Mengapa kedalaman membantu: komposisi
3. *Feature* bertingkat dari umum ke spesifik
4. *Vanishing* dan *exploding gradient*
5. Penanganan, termasuk *gradient clipping*
6. *Residual connection* dan masalah degradasi
7. Membandingkan model dangkal dan dalam secara adil
8. Pola arsitektur: *stem*, badan, kepala
9. Kebutuhan komputasi, memori, dan data
10. *Transfer learning*: hemat data
11. Etika penggunaan data
12. Praktikum dan progres tugas besar UAS

Sasaran pertemuan ini

Anda dapat menjelaskan dengan hitungan mengapa *gradient* lenyap atau meledak pada jaringan dalam, menunjukkan bagaimana *residual connection* memperbaikinya, membandingkan jaringan dangkal dan dalam dengan protokol yang adil, serta memakai model *pretrained* dengan *transfer learning* secara bertanggung jawab terhadap data yang dipakai.

Menyambung dari Pertemuan 11

Pada Pertemuan 11 Anda melatih MLP secara stabil: *initialization* Xavier dan He, Adam, *batch normalization*, *dropout*, *early stopping*, dan membaca kurva *loss*. Semua alat itu bekerja baik pada jaringan dangkal.

Yang terjadi pada 2012–2015

AlexNet (8 *layer*) memenangi ImageNet 2012 dengan selisih lebar. Tiga tahun kemudian pemenangnya ResNet dengan 152 *layer*. Dalam waktu singkat "dalam" berubah dari delapan menjadi ratusan *layer*, dan *deep learning* menjadi nama seluruh bidang.

Yang sering gagal

Menumpuk *layer* begitu saja sering membuat jaringan lebih **buruk**, bahkan pada *training data*. *Gradient*-nya lenyap atau meledak sebelum sampai ke *layer* awal.

Pertanyaan hari ini

Bila satu *hidden layer* yang cukup lebar sudah dapat menghampiri fungsi apa pun, untuk apa menumpuk *layer*? Bila menumpuk itu berguna, mengapa sulit dilatih?

Sumber: Krizhevsky dkk. 2012; He dkk. 2016; LeCun, Bengio, Hinton 2015.

Mengapa kedalaman membantu

Lebar saja sudah cukup, secara teori

Universal approximation theorem dari Pertemuan 10 menjamin bahwa satu *hidden layer* yang cukup lebar dapat menghampiri fungsi kontinu apa pun. Masalahnya ada pada kata “cukup lebar”, yang bisa berarti **sangat** lebar.

Jaringan lebar

Setiap neuron menyumbang satu “tekukan” pada fungsi. Untuk membuat fungsi dengan m potongan linier pada *input* satu dimensi, diperlukan sekitar m neuron. Tidak ada pemakaian ulang.

Jaringan dalam

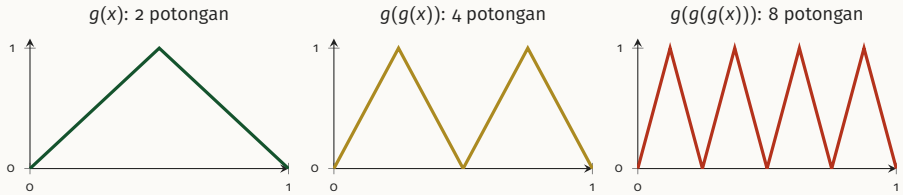
Hasil satu *layer* menjadi bahan *layer* berikutnya. Inilah **komposisi**: pola yang dibangun sekali dipakai ulang berkali-kali oleh *layer* di atasnya.

Pertanyaan yang tepat untuk jaringan dalam adalah soal **efisiensi**: berapa neuron yang dibutuhkan untuk membangun fungsi yang rumit. Contoh berikut menjawabnya dengan angka.

Contoh: fungsi tenda dan gigi gergaji

Fungsi tenda pada $[0, 1]$ dapat dibangun dari **dua neuron ReLU**:

$$g(x) = \begin{cases} 2x & x \le 0{,}5 \\ 2(1-x) & x > 0{,}5 \end{cases} \quad = \quad 2\,\mathrm{ReLU}(x) - 4\,\mathrm{ReLU}(x - 0{,}5)$$



Setiap komposisi melipat selang $[0, 1]$ menjadi dua dan menggandakan jumlah gigi. Jaringan k *layer* dengan dua neuron per *layer*, total $2k$ neuron, menghasilkan 2^k potongan linier.

Menghitung dengan tangan: komposisi dan *cost*-nya

Periksa g, lalu susun

$$g(0{,}25) = 2(0{,}25) - 4 \cdot 0 = 0{,}5$$
$$g(0{,}75) = 1{,}5 - 4(0{,}25) = 0{,}5$$

Sekarang ikuti satu titik $x = 0{,}3$ melewati tiga *layer*:

$$g(0{,}3) = 0{,}6$$
$$g(g(0{,}3)) = g(0{,}6) = 2(1 - 0{,}6) = 0{,}8$$
$$g(g(g(0{,}3))) = g(0{,}8) = 2(1 - 0{,}8) = 0{,}4$$

Layer kedua dan ketiga memakai rumus yang sama persis, tetapi *input*-nya sudah "dilipat" oleh *layer* sebelumnya.

Satu *layer* ReLU dengan m neuron pada *input* satu dimensi paling banyak menghasilkan $m + 1$ potongan. Untuk meniru gigi gergaji 2^k potongan ia memerlukan sekitar 2^k neuron.

k layer	dalam ($2k$ neuron)	dangkal ($\approx 2^k$)
3	6	8
6	12	64
10	20	1.024
20	40	1.048.576

Empat puluh neuron melawan sejuta. Telgarsky (2016) memformalkan argumen ini, dan Montúfar dkk. (2014) menunjukkan jumlah daerah linier jaringan ReLU dapat tumbuh eksponensial terhadap kedalaman.

Kode: menghitung potongan linier gigi gergaji

```python
import numpy as np

def relu(z):
    return np.maximum(z, 0.0)

def tenda(x):                  # dua neuron ReLU
    return 2 * relu(x) - 4 * relu(x - 0.5)

def jumlah_potongan(y, x):
    miring = np.round(np.diff(y) / np.diff(x), 6)
    return 1 + np.count_nonzero(np.diff(miring))

x = np.linspace(0, 1, 2**12 + 1)
y = x.copy()
for k in range(1, 7):
    y = tenda(y)               # tambah satu lapisan
    print(k, "lapis:", jumlah_potongan(y, x))
```

***Output*-nya** $2, 4, 8, 16, 32, 64$ untuk $k = 1, \ldots, 6$.

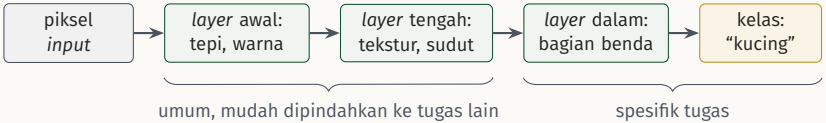
Kisi titik x dibuat dengan $2^{12} + 1$ titik agar setiap tekukan jatuh tepat di titik kisi; kemiringan dibulatkan supaya *error* pembulatan tidak terhitung sebagai tekukan baru.

Coba sendiri: ganti loop dengan satu *layer* lebar berisi m neuron acak, dan periksa bahwa jumlah potongannya tidak pernah melewati $m + 1$.

Sumber: Kode lengkap: `sample-code-buku`, `bab14-deep-neural-network/gigi_gergaji.py`.

Feature bertingkat: dari umum ke spesifik

Banyak data dunia nyata tersusun bertingkat: piksel membentuk tepi, tepi membentuk tekstur, tekstur membentuk bagian benda, bagian membentuk benda utuh. Huruf membentuk kata, lalu frasa, lalu kalimat.



Zeiler dan Fergus (2014) memperlihatkannya secara visual pada jaringan konvolusi. Sifat ini nanti menjadi dasar *transfer learning*.

Ringkasan

Dengan jumlah parameter sama, jaringan dalam dapat jauh lebih mampu karena hasil antara dipakai ulang. Itulah *inductive bias* utama *deep learning*: fungsi yang dicari dianggap tersusun dari fungsi-fungsi yang lebih sederhana.

Vanishing dan *exploding gradient*

Gradient layer awal adalah hasil kali yang panjang

Ambil jaringan paling sederhana: satu neuron per *layer*, L *layer*, $z_\ell = w_\ell\, a_{\ell-1}$ dan $a_\ell = g(z_\ell)$. Aturan rantai dari Pertemuan 10 memberi

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_L} \cdot \underbrace{\prod_{\ell=2}^{L} \Big(w_\ell\, g'(z_\ell) \Big)}_{L-1 \text{ faktor}} \cdot g'(z_1)\, a_0$$

Faktor rata-rata < 1

Hasil kali menyusut secara eksponensial. *Layer* awal menerima *gradient* yang nyaris nol dan praktis berhenti belajar: ***vanishing gradient***.

Faktor rata-rata > 1

Hasil kali membesar secara eksponensial. Satu langkah pembaruan merusak parameter, *loss* melonjak atau menjadi `nan`: ***exploding gradient***.

Pada jaringan lebar, perannya digantikan hasil kali matriks $W^{(\ell)\top}\,\mathrm{diag}(g'(\mathbf{z}^{(\ell)}))$; yang menentukan adalah besar nilai singularnya.

Sumber: Bengio dkk. 1994; Pascanu dkk. 2013.

Seberapa cepat *gradient* lenyap: sigmoid dalam kasus terbaik

Derivative sigmoid paling besar $g'(0) = \sigma(0)(1-\sigma(0)) = 0{,}5 \times 0{,}5 = 0{,}25$. Ambil kasus **terbaik**: semua $w_\ell = 1$ dan semua $z_\ell = 0$, sehingga setiap faktor tepat 0,25.

L	$0{,}25^L$ (sigmoid)	$1{,}5^L$ (faktor besar)
5	$9{,}8 \times 10^{-4}$	7,6
10	$9{,}5 \times 10^{-7}$	57,7
20	$9{,}1 \times 10^{-13}$	3.325
50	$7{,}9 \times 10^{-31}$	$6{,}4 \times 10^{8}$

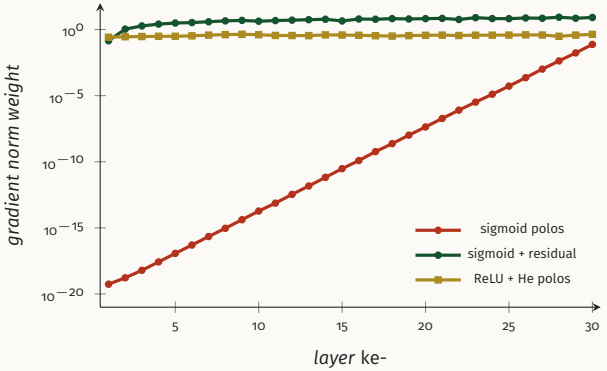
Cara membacanya

Pada jaringan sigmoid dua puluh *layer*, *layer* pertama menerima sekitar satu per triliun dari *gradient layer* terakhir, dan itu kasus terbaik. Dengan *learning rate* 0,1, *weight*-nya bergeser sekitar 10^{-13} per langkah: secara praktis membeku.

Sebaliknya, faktor 1,5 (misalnya *weight* terlalu besar dengan aktivasi linier) memberi $1{,}5^{20} \approx 3.300$: *gradient* meledak.

Bandingkan tanh: $\tanh'(0) = 1$, sehingga dalam kasus terbaik faktornya 1. Itulah salah satu alasan tanh lebih disukai daripada sigmoid di *hidden layer*, dan ReLU lebih disukai lagi.

Diukur langsung: *gradient norm* per *layer*



Jaringan 30 *layer* selebar 256, diukur saat *initialization* pada satu *mini-batch* acak berukuran 512. Sumbu tegak berskala logaritma.

- **Sigmoid polos**: *layer* pertama sekitar 10^{-20}, *layer* terakhir 0,075. Selisihnya delapan belas orde besaran.
- **Sigmoid + residual**: jaringan yang sama ditambah jalur identitas; semua *layer* di orde 0,1 sampai 10.
- **ReLU + He** tanpa residual juga stabil di awal, karena *derivative* ReLU di sisi positif bernilai 1.

Kode: mengukur *gradient norm* per *layer*

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

class Blok(nn.Module):
    def __init__(self, akt, res, n=256):
        super().__init__()
        self.lin = nn.Linear(n, n)
        self.akt = akt
        self.residual = res
        if isinstance(akt, nn.ReLU):
            nn.init.kaiming_normal_(self.lin.weight)
        else: nn.init.xavier_normal_(self.lin.weight)

    def forward(self, x):
        h = self.akt(self.lin(x))
        return x + h if self.residual else h
```

```python
def norma_gradien(akt, res, D=30):
    blok = [Blok(akt, res) for _ in range(D)]
    model = nn.Sequential(*blok,
                          nn.Linear(256, 10))
    x = torch.randn(512, 256)
    y = torch.randint(0, 10, (512,))
    F = nn.functional
    F.cross_entropy(model(x), y).backward()
    return [model[i].lin.weight.grad.norm()
            for i in range(D)]

sig = nn.Sigmoid()
print(norma_gradien(sig, False)[:3])
print(norma_gradien(sig, True)[:3])
```

Satu kali `backward()` pada data acak sudah cukup untuk mendiagnosis. Biasakan memeriksa *gradient norm* per *layer* setiap kali jaringan dalam Anda menolak belajar.

Sumber: Kode lengkap: `sample-code-buku`, `bab14-deep-neural-network/gradien_per_lapis.py`.

Penanganan: kombinasi teknik yang sebagian besar sudah Anda kenal

Teknik	Cara kerja	Mengatasi
ReLU dan variannya	*derivative* 1 di sisi positif, tidak jenuh	*vanishing*
Initialization Xavier/He	menjaga *variance* sinyal maju dan mundur (Pertemuan 11)	keduanya
Batch/*layer normalization*	mengembalikan skala aktivasi setiap *layer*	keduanya
Residual connection	jalur identitas membawa *gradient* tanpa diperkecil	terutama *vanishing*
Gradient clipping	memotong *gradient norm* di atas ambang	*exploding*
Gating (LSTM, GRU)	gerbang yang dipelajari mengatur aliran informasi sepanjang waktu	keduanya, pada RNN
Learning rate kecil dan *warmup*	mencegah langkah awal yang merusak	*exploding*

Tidak ada satu teknik yang menyelesaikan semuanya. Resep standar jaringan dalam hari ini menggabungkan ReLU (atau GELU), *initialization* He, *normalization*, jalur residual, dan *warmup*; *clipping* ditambahkan sebagai sabuk pengaman.

Gradient clipping: murah dan selalu layak dipasang

Bila *gradient norm* gabungan $\|\mathbf{g}\|$ melebihi ambang c, *gradient* diskalakan:

$$\mathbf{g} \leftarrow c\,\frac{\mathbf{g}}{\|\mathbf{g}\|} \qquad \text{bila } \|\mathbf{g}\| > c$$

Arahnya tetap, hanya panjangnya dibatasi.

Contoh angka

$\mathbf{g} = (30, 40)$, sehingga $\|\mathbf{g}\| = 50$. Dengan $c = 1$: $\mathbf{g} \leftarrow (30, 40)/50 = (0{,}6;\ 0{,}8)$. Normanya kini 1, arahnya sama. *Gradient* $(0{,}3;\ 0{,}4)$ dengan norma 0,5 dibiarkan apa adanya.

Di PyTorch cukup satu baris di antara `backward()` dan `step()`:

```
opt.zero_grad()
loss = loss_fn(model(xb), yb)
loss.backward()
torch.nn.utils.clip_grad_norm_(
    model.parameters(), max_norm=1.0)
opt.step()
```

Fungsi ini mengembalikan norma **sebelum** dipotong. Catat nilai itu di log *training*: lonjakan normanya sering menjadi tanda pertama *training* akan rusak.

Residual connection

Masalah degradasi: lebih dalam, lebih buruk di *training data*

Dengan ReLU, *initialization* yang baik, dan *batch normalization*, jaringan belasan *layer* dapat dilatih. He dkk. (2016) lalu menemukan hal yang aneh pada CIFAR-10: jaringan "polos" 56 *layer* punya *error* ***training*** lebih tinggi daripada jaringan 20 *layer*.

Ini tidak sama dengan *overfitting*

Overfitting justru menurunkan *error training* sambil menaikkan *error* validasi. Di sini *error training* sendiri yang naik. Gejala ini diberi nama **masalah degradasi**.

Seharusnya tidak mungkin

Ambil jaringan 20 *layer* terlatih, tambahkan 36 *layer* yang masing-masing menghitung identitas. Hasilnya sama persis. Jadi solusi yang sama baiknya **ada**; yang gagal adalah *gradient descent* dalam menemukannya.

Ringkasan

Bagi tumpukan *layer* tak linier, mempelajari fungsi identitas $\mathbf{h}(\mathbf{x}) = \mathbf{x}$ ternyata sulit. Masalahnya ada pada *optimization*, dan karena itu solusinya harus mengubah apa yang dioptimasi.

Gagasannya: pelajari sisanya saja

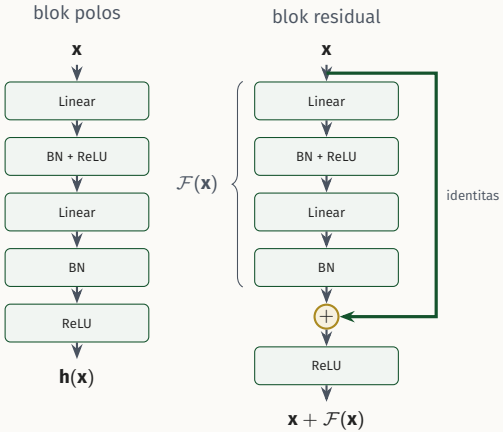
Alih-alih meminta beberapa *layer* mempelajari $\mathbf{h}(\mathbf{x})$ secara langsung, minta mereka mempelajari ***residual***, selisihnya dari identitas:

$$\mathcal{F}(\mathbf{x}) = \mathbf{h}(\mathbf{x}) - \mathbf{x} \qquad \Longrightarrow \qquad \boxed{\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x}; W)}$$

Sambungan yang menambahkan $\mathbf{x}$ langsung ke *output* disebut ***skip connection*** atau ***shortcut***.

Bila identitas memang yang terbaik, jaringan cukup mendorong *weight* $\mathcal{F}$ ke nol, dan itu mudah, apalagi dengan *weight decay*.

Bila dimensi *input* dan *output* blok berbeda, jalur identitas diganti *projection shortcut*: satu *layer* linier (pada CNN, konvolusi 1×1) yang menyamakan dimensi.



Mengapa *gradient* mengalir lewat blok residual

Turunkan $\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x})$ terhadap *input*-nya:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = I + \frac{\partial \mathcal{F}}{\partial \mathbf{x}} \qquad \text{dibandingkan blok polos:} \qquad \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$$

Untuk dua blok bertumpuk, jabarkan hasil kalinya:

$$\left(I + \frac{\partial \mathcal{F}_2}{\partial \mathbf{x}}\right)\left(I + \frac{\partial \mathcal{F}_1}{\partial \mathbf{x}}\right) = \underbrace{I}_{\text{jalur langsung}} + \frac{\partial \mathcal{F}_1}{\partial \mathbf{x}} + \frac{\partial \mathcal{F}_2}{\partial \mathbf{x}} + \frac{\partial \mathcal{F}_2}{\partial \mathbf{x}}\frac{\partial \mathcal{F}_1}{\partial \mathbf{x}}$$

Suku I membawa *gradient* dari *output* ke *layer* mana pun **tanpa diperkecil sama sekali**. Pada L blok, penjabarannya memuat 2^L jalur dengan panjang berbeda-beda, dan selalu ada satu jalur yang tidak melewati faktor kecil apa pun.

Ringkasan

Jaringan polos memaksa *gradient* melewati satu jalur panjang berisi L faktor. Jaringan residual menyediakan banyak jalur, termasuk jalan tol tanpa hambatan. He dkk. (2016b) menunjukkan bahwa dengan susunan *pre-activation* (*normalization* dan aktivasi di dalam cabang $\mathcal{F}$), sinyal merambat langsung antara dua blok mana pun, maju maupun mundur.

Contoh terhitung: tiga *layer* sigmoid, polos dan residual

Rantai skalar, $a_0 = 1$, semua $w_\ell = 1$. Polos: $a_\ell = \sigma(a_{\ell-1})$. Residual: $a_\ell = a_{\ell-1} + \sigma(a_{\ell-1})$. Faktor lokal mundurnya $\sigma'(z)$ untuk polos dan $1 + \sigma'(z)$ untuk residual, dengan $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

Polos

ℓ	z_ℓ	$a_\ell = \sigma(z_\ell)$	$\sigma'(z_\ell)$
1	1,0000	0,7311	0,1966
2	0,7311	0,6750	0,2194
3	0,6750	0,6626	0,2236

$$\frac{\partial a_3}{\partial a_0} = 0{,}1966 \times 0{,}2194 \times 0{,}2236 = \mathbf{0{,}0096}$$

Residual

ℓ	z_ℓ	$\sigma(z_\ell)$	$a_\ell = z_\ell + \sigma(z_\ell)$	$1 + \sigma'(z_\ell)$
1	1,0000	0,7311	1,7311	1,1966
2	1,7311	0,8495	2,5806	1,1278
3	2,5806	0,9296	3,5102	1,0654

$$\frac{\partial a_3}{\partial a_0} = 1{,}1966 \times 1{,}1278 \times 1{,}0654 = \mathbf{1{,}438}$$

Baru tiga *layer*, rantai polos sudah memperkecil *gradient* sekitar seratus kali, sedangkan rantai residual menjaganya di sekitar 1. Perhatikan pula baris ketiga residual: walaupun $\sigma'(2{,}58) = 0{,}065$ sudah kecil karena sigmoid mulai jenuh, faktornya tetap di atas 1 berkat suku identitas.

Satu catatan: jalur residual juga bisa meledak

Bila setiap cabang menyumbang faktor a, hasil kali $(1+a)^L$ dapat membesar. Dengan faktor lokal $a = 0{,}25$ (sigmoid, $w = 1$, $z = 0$):

L	polos $0{,}25^L$	residual $1{,}25^L$
5	$9{,}8 \times 10^{-4}$	3,05
10	$9{,}5 \times 10^{-7}$	9,31
20	$9{,}1 \times 10^{-13}$	86,7
50	$7{,}9 \times 10^{-31}$	$7{,}0 \times 10^{4}$

Bila cabangnya diredam seratus kali, $a = 0{,}0025$, maka $1{,}0025^{50} = 1{,}13$.

Karena itu dalam praktik

- *Normalization* diletakkan **di dalam** cabang $\mathcal{F}$, sehingga sumbangan cabang terkendali.
- Parameter skala γ pada BN terakhir di setiap cabang sering diinisialisasi **nol**. Setiap blok lalu mulai sebagai identitas murni, dan jaringan 100 *layer* berperilaku seperti jaringan dangkal di awal *training*.
- Pada *transformer*, cabangnya kerap diskalakan dengan faktor kecil yang bergantung pada jumlah blok.

Kode: blok residual dan MLP dalam di PyTorch

```python
import torch.nn as nn

class BlokResidual(nn.Module):
    def __init__(self, lebar):
        super().__init__()
        self.f = nn.Sequential(
            nn.Linear(lebar, lebar), nn.BatchNorm1d(lebar),
            nn.ReLU(),
            nn.Linear(lebar, lebar), nn.BatchNorm1d(lebar))
        self.akt = nn.ReLU()

    def forward(self, x):
        return self.akt(x + self.f(x))

def buat_model(n_blok, lebar=256):
    stem = [nn.Flatten(), nn.Linear(784, lebar),
            nn.ReLU()]
    badan = [BlokResidual(lebar) for _ in range(n_blok)]
    kepala = [nn.Linear(lebar, 10)]
    return nn.Sequential(*stem, *badan, *kepala)
```

Hal yang perlu diperhatikan

- `self.f[-1]` adalah BN terakhir; *weight*-nya adalah γ. Mengisinya dengan nol membuat $\mathcal{F}(\mathbf{x}) = 0$ saat awal.
- Varian **polos** untuk pembanding cukup mengganti `x + self.f(x)` dengan `self.f(x)`. Semua *layer* lain identik, sehingga perbedaan hasil hanya berasal dari jalur identitas.
- Pola *stem*, badan, kepala di `buat_model` adalah pola umum yang akan kita bahas sebentar lagi.

Sumber: Kode lengkap: `sample-code-buku, bab14-deep-neural-network/residual_mlp.py`.

Contoh terhitung: berapa parameter model tadi?

Lebar 256, *input* 784 piksel, 10 kelas. *Layer* linier $n_{\text{in}} \to n_{\text{out}}$ punya $n_{\text{in}} n_{\text{out}} + n_{\text{out}}$ parameter; BN punya γ dan β per *feature*, jadi 2×256.

$$
\begin{aligned}
\textit{stem}&: 784 \cdot 256 + 256 = 200.960 \\
\text{satu blok}&: 2(256 \cdot 256 + 256) + 2(2 \cdot 256) \\
&= 131.584 + 1.024 = 132.608 \\
\text{kepala}&: 256 \cdot 10 + 10 = 2.570
\end{aligned}
$$

dangkal, 1 blok: $200.960 + 132.608 + 2.570 = \mathbf{336.138}$

dalam, 20 blok: $200.960 + 20(132.608) + 2.570 = \mathbf{2.855.690}$

Cost model 20 blok

Satu langkah *training* memerlukan sekitar $6P$ FLOP per contoh (maju $2P$, mundur sekitar $4P$):

$$6 \times 2.855.690 \approx 1{,}7 \times 10^7 \text{ FLOP per contoh}$$

Memori dengan Adam fp32, 16 byte per parameter:

$$16 \times 2.855.690 \approx 45{,}7 \text{ MB}$$

belum termasuk aktivasi. Masih sangat ringan untuk laptop.

Varian polos 20 blok punya parameter yang sama persis, 2.855.690; jalur identitas tidak menambah parameter.

Membandingkan dangkal dan dalam secara adil

Protokol yang dianjurkan

1. Data, *split*, praproses, dan anggaran *training* (*epoch* atau waktu) sama.
2. Setel *learning rate* untuk **masing-masing** model; jangan memakai satu nilai yang kebetulan cocok untuk salah satunya.
3. Laporkan jumlah parameter, *training time*, *training loss*, dan *validation metric*.
4. Ulangi dengan beberapa *seed*, laporkan rata-rata dan simpangan baku.
5. Bila bisa, bandingkan juga model dengan jumlah parameter setara, agar efek kedalaman terpisah dari efek ukuran.

Fashion-MNIST, *seed* 0, 5 *epoch*, AdamW $\eta = 10^{-3}$:

Varian	Param.	*Train loss*	*Test accuracy*
dangkal, 1 blok	336 rb	0,258	87,6%
polos, 20 blok	2,86 jt	0,698	76,7%
residual, 20 blok	2,86 jt	0,288	87,2%

Pelajaran 1. Degradasi muncul jelas: jaringan polos 20 blok punya *training loss* hampir tiga kali jaringan dangkal, padahal parameternya delapan kali lebih banyak. Versi residual dengan *layer* identik tidak mengalaminya.

Pelajaran 2. Pada Fashion-MNIST yang diratakan, kedalaman belum memberi keuntungan dalam lima *epoch*. Kesimpulan yang sah menunggu beberapa *seed* dan *training* lebih panjang.

Dalam tidak selalu lebih baik

Hati-hati sebelum menumpuk *layer*

Untuk data tabular berukuran kecil sampai sedang, MLP dua atau tiga *layer*, atau *gradient boosting* dari Pertemuan 4, sering sama baik atau lebih baik daripada jaringan dalam. Kedalaman paling menguntungkan ketika datanya punya struktur bertingkat (citra, suara, teks) dan jumlahnya besar.

Kedalaman cenderung menang bila

- *input*-nya mentah dan bertingkat: piksel, gelombang suara, token teks;
- datanya puluhan ribu contoh atau lebih, atau ada model *pretrained*;
- anggaran komputasi tersedia untuk menyetel dan mengulang.

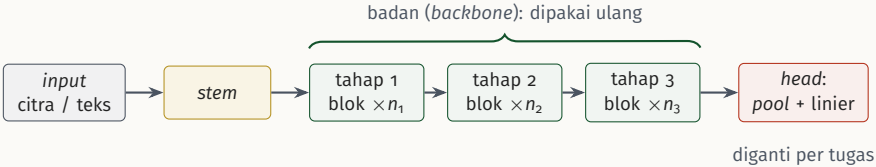
Model dangkal cenderung cukup bila

- *feature*-nya sudah bermakna, seperti kolom tabel hasil *feature engineering*;
- datanya ratusan sampai beberapa ribu baris;
- keputusan perlu dijelaskan dan dilatih ulang sering.

Aturan kerja dari Pertemuan 7 masih berlaku: mulai dari patokan sederhana, lalu buktikan dengan angka validasi bahwa kedalaman memberi perbaikan nyata.

Arsitektur umum dan kebutuhan sumber daya

Pola *stem*, badan, kepala



- ***Stem***: mengubah *input* mentah menjadi representasi awal. Proyeksi linier pada MLP, konvolusi besar pada CNN, *embedding* token pada *transformer*.
- **Badan** (*backbone*): tumpukan blok berulang, biasanya residual, sering dikelompokkan dalam beberapa tahap dengan lebar atau resolusi berbeda. Hampir semua parameter dan komputasi ada di sini.
- **Kepala** (*head*): *layer* akhir yang spesifik tugas, misalnya *pooling* lalu satu *layer* linier ke K kelas.

Ringkasan

Pemisahan ini penting secara praktis. Badan yang sudah dilatih pada data besar dapat dipakai ulang untuk tugas lain cukup dengan mengganti kepalanya. Itulah *transfer learning*.

Tonggak arsitektur: kedalaman dari 7 ke ratusan *layer*

Arsitektur	Tahun	Kedalaman	Gagasan kunci
LeNet-5	1998	7 *layer*	konvolusi dan *pooling* untuk angka tulisan tangan
AlexNet	2012	8 *layer*	ReLU, *dropout*, *training* di GPU; sekitar 60 juta parameter
VGG	2015	16–19 *layer*	tumpukan konvolusi 3×3 yang seragam; sekitar 138 juta parameter (VGG-16)
ResNet	2016	18–152 *layer*	*residual connection*; pemenang ILSVRC 2015
DenseNet	2017	121–264 *layer*	setiap *layer* menerima *output* semua *layer* sebelumnya
Vision Transformer	2020	12–32 blok	*self-attention* pada potongan citra; tetap memakai residual dan LN

Residual connection tidak berhenti di ResNet. Setiap blok *transformer*, termasuk model bahasa besar yang kita bahas di Pertemuan 14 dan 15, memakai pola $\mathbf{x} + \mathcal{F}(\mathbf{x})$ yang sama bersama *layer normalization*. Detail *layer* konvolusi menunggu Pertemuan 13.

Sumber: LeCun dkk. 1998; Krizhevsky dkk. 2012; Simonyan dan Zisserman 2015; He dkk. 2016; Huang dkk. 2017; Dosovitskiy dkk. 2021.

Menaksir *cost* sebelum melatih

Operasi

Layer linier $n_{\text{out}} \times n_{\text{in}}$ memerlukan sekitar $2n_{\text{in}}n_{\text{out}}$ FLOP per contoh: satu kali dan satu tambah per *weight*. Jadi *forward* MLP sekitar $2P$ FLOP, *backward* kira-kira dua kalinya, sehingga satu langkah *training* sekitar **6P** FLOP per contoh. Kaplan dkk. (2020) memakai taksiran yang sama, $C \approx 6PD$ dengan D jumlah token, untuk model bahasa.

Memori

Adam fp32: 4 byte nilai, 4 byte *gradient*, 8 byte dua momen Adam, total sekitar **16** byte per parameter, belum termasuk aktivasi yang disimpan untuk *backward*.

Contoh: memori *training*

- MLP MNIST Pertemuan 10, 235.146 parameter: $16 \times 235.146 \approx 3{,}8$ MB.
- Ukuran ResNet-50, 25,6 juta parameter: ≈ 410 MB, ditambah aktivasi yang untuk citra 224×224 dan *batch* besar bisa beberapa GB.
- Model bahasa 7 miliar parameter: $16 \times 7 \times 10^9 = 112$ GB. Melebihi satu GPU 80 GB, sehingga *training*-nya dibagi ke banyak GPU atau memakai *precision* lebih rendah.

Inferensi saja, model 7 miliar fp16: $2 \times 7 \times 10^9 = 14$ GB.

Data, *scaling laws*, dan di mana Anda bisa melatih

Jaringan dalam lapar data. AlexNet dilatih dengan sekitar 1,2 juta citra berlabel dari ImageNet. Penelitian ***scaling laws*** menemukan bahwa pada model bahasa, *loss* turun mengikuti hukum pangkat yang cukup teratur terhadap ukuran model, jumlah data, dan komputasi.

Untuk anggaran komputasi tertentu, ukuran model dan jumlah token sebaiknya dinaikkan seimbang (Hoffmann dkk. 2022).

Ukuran	Contoh	Tempat melatih
$< 10^6$ param.	MLP MNIST	CPU laptop, menit
10^6–10^8	ResNet-18 s.d. ResNet-50	satu GPU; *fine-tuning* hitungan jam
10^8–10^{10}	model bahasa kecil–menengah	beberapa GPU; *fine-tuning* hemat parameter di satu GPU
$> 10^{10}$	model fondasi	*cluster* GPU organisasi besar

Ringkasan

Bagi kebanyakan organisasi, melatih jaringan dalam besar dari nol jarang masuk akal: *computational cost* dan energinya besar, dan data berlabel sebanyak itu jarang ada. Jalan yang jauh lebih umum adalah memakai model yang sudah dilatih pihak lain.

Sumber: Kaplan dkk. 2020; Hoffmann dkk. 2022; Strubell dkk. 2019.

Transfer learning

Gagasan *transfer learning*

Transfer learning: memakai pengetahuan yang dipelajari model pada **tugas sumber** (biasanya data sangat besar) untuk **tugas sasaran** (biasanya data sedikit). Pada jaringan dalam, "pengetahuan" itu adalah *weight* badan jaringan.

Karena *feature layer* awal bersifat umum, badan yang dilatih pada ImageNet berguna untuk mengenali citra daun padi, motif batik, atau foto produk UMKM, walaupun kelas-kelas itu tidak ada di ImageNet.

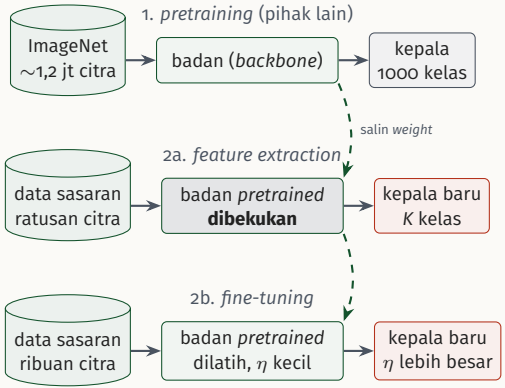
Yosinski dkk. (2014) mengukurnya: *feature layer* awal dapat dipindahkan dengan baik, sedangkan *feature layer* akhir makin spesifik terhadap tugas sumber.

Sumber model *pretrained*

- `torchvision.models`: ResNet, EfficientNet, ViT, dengan praproses bawaan.
- `timm`: ratusan arsitektur citra.
- Hugging Face Hub (`transformers`): model teks dan suara.

Untuk model besar, *fine-tuning* hemat parameter seperti LoRA hanya melatih sebagian kecil parameter.

Dua cara: *feature extraction* dan *fine-tuning*



Feature extraction. Bekukan (*freeze*) seluruh badan, ganti kepala sesuai jumlah kelas sasaran, latih kepala saja. Murah, cepat, aman bila data sangat sedikit.

Fine-tuning. Mulai dari *weight pretrained*, latih seluruh jaringan atau beberapa tahap terakhir dengan η kecil. Lebih kuat bila data cukup banyak atau domainnya agak berbeda.

Praktik lazim: *learning rate* badan lebih kecil daripada kepala (*discriminative learning rate*), dan beberapa *epoch* pertama hanya melatih kepala agar kepala yang masih acak tidak merusak badan.

Memilih strategi: berapa banyak data, seberapa mirip domainnya

	Domain mirip (foto benda sehari-hari)	**Domain berbeda** (citra medis, satelit, mikroskop)
Data sedikit	*feature extraction*; kepala linier sering sudah cukup	*feature extraction* dari *layer* lebih awal, atau *fine-tuning* sangat hati-hati dengan *augmentation* kuat
Data banyak	*fine-tuning* penuh dengan η kecil	*fine-tuning* penuh; bila sangat banyak, pertimbangkan model *pretrained* khusus domain

Contoh: 800 foto daun padi, empat jenis penyakit

Domainnya foto tanaman, cukup mirip foto sehari-hari, dan datanya sedikit. Rencana yang masuk akal: ResNet-18 *pretrained*, mulai dengan *feature extraction* sebagai patokan, lalu *fine-tuning* tahap terakhir dengan η kecil dan *augmentation* (putar, balik, variasi kecerahan). Bagi data per **lahan atau tanggal foto**, jangan per foto acak, agar foto yang hampir kembar tidak bocor ke *test data*. Laporkan F1 per kelas karena penyakit langka biasanya sedikit fotonya.

Kode: ResNet-18 *pretrained* dalam dua mode

```
import torch, torch.nn as nn
from torchvision import models

bobot = models.ResNet18_Weights.IMAGENET1K_V1
tf = bobot.transforms()     # praproses bawaan
model = models.resnet18(weights=bobot)
model.fc = nn.Linear(model.fc.in_features, 10)

if mode == "fitur":         # feature extraction
    for nama, p in model.named_parameters():
        p.requires_grad = nama.startswith("fc.")
    opt = torch.optim.AdamW(model.fc.parameters(),
                            lr=1e-3)
else:                       # fine-tuning
    badan = [p for n, p in model.named_parameters()
             if not n.startswith("fc.")]
    opt = torch.optim.AdamW([
        {"params": badan, "lr": 1e-4},
        {"params": model.fc.parameters(), "lr": 1e-3}])
```

- `bobot.transforms()` memberi ukuran, potongan, dan *normalization* yang sama dengan saat *pretraining*.
- `model.fc` adalah kepala asli $512 \to 1000$; diganti menjadi $512 \to 10$ untuk CIFAR-10, yaitu 5.130 parameter.
- Mode `fitur` hanya melatih 5.130 dari sekitar 11,2 juta parameter.
- Mode *fine-tuning* memakai dua kelompok parameter dengan η berbeda sepuluh kali lipat.

Sumber: Kode lengkap: `sample-code-buku`, `bab14-deep-neural-network/transfer_learning.py`.

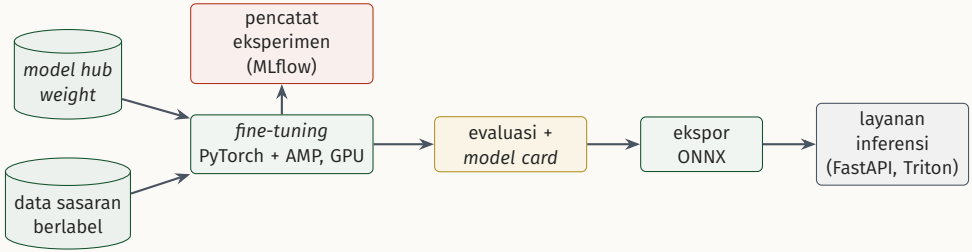
Jebakan *transfer learning*

Empat kesalahan yang paling sering

1. **Praproses berbeda.** Model *pretrained* mengharapkan ukuran, skala, dan *normalization* yang sama dengan saat ia dilatih. Pakai `weights.transforms()` atau *configuration* yang disertakan model.
2. ***Batch normalization* saat dibekukan.** Bila badan dibekukan tetapi model dalam mode `train()`, statistik berjalan BN tetap diperbarui oleh data baru. Jalankan badan yang dibekukan dalam mode `eval()`.
3. ***Learning rate* terlalu besar saat *fine-tuning*.** *Weight pretrained* dapat "terlupakan" dalam beberapa langkah (*catastrophic forgetting*).
4. **Lisensi.** *Weight pretrained* punya lisensi, dan sebagian melarang penggunaan komersial. Periksa sebelum dipakai dalam produk.

Satu pembanding wajib dalam laporan: arsitektur yang sama **tanpa** *weight pretrained* (`weights=None`). Tanpa itu Anda tidak tahu berapa sumbangan *transfer learning* sesungguhnya.

Dari *fine-tuning* sampai layanan: *tech stack* ringkas



- **Sumber model**: `timm`, `torchvision`, Hugging Face Hub.
- ***Training***: PyTorch 2.x dengan `torch.amp` (*mixed precision* menghemat memori GPU); PEFT (LoRA) untuk model besar.
- **Pencatatan**: MLflow atau Weights & Biases untuk *metric*, artefak, dan *model registry*.
- **Ekspor dan layanan**: ONNX Runtime, `torch.export`, TensorRT; FastAPI untuk API sederhana, Triton atau BentoML untuk *batching* dinamis.

Etika penggunaan data

Persetujuan, tujuan, dan data pribadi

Jaringan dalam menyerap pola dari data dalam jumlah besar, termasuk pola yang tidak kita inginkan. Pertanyaan etis tentang data karenanya termasuk desain sejak hari pertama proyek.

UU No. 27 Tahun 2022 (Pelindungan Data Pribadi)

Wajah, suara, nomor identitas, lokasi, dan catatan kesehatan adalah data pribadi. UU PDP antara lain mensyaratkan dasar pemrosesan yang sah (misalnya persetujuan yang jelas), pemrosesan terbatas pada tujuan yang dinyatakan, dan perlakuan lebih ketat untuk data spesifik seperti kesehatan dan biometrik.

Prinsip praktis bagi pengembang model

- Kumpulkan hanya yang dibutuhkan (*data minimization*).
- Data layanan pelanggan tidak otomatis boleh melatih model pemasaran.
- Samarkan pengenal langsung (*pseudonymization*); data “anonim” sering dapat dikenali ulang bila digabung data lain.
- Batasi akses. Model pun dapat menghafal *training example* dan membocorkannya.

Bias, hak atas data, dan dokumentasi

Bias dan keadilan. Ketimpangan dalam data menjadi ketimpangan dalam keputusan. Buolamwini dan Gebru (2018) menunjukkan beberapa sistem *classification* gender komersial jauh lebih sering salah pada perempuan berkulit gelap, karena *training data* dan evaluasinya didominasi kelompok tertentu.

Bayangkan model penilaian kredit yang dilatih dari nasabah perkotaan di Jawa, lalu dipakai untuk calon nasabah di daerah dengan pola pendapatan berbeda. Evaluasi harus dipecah per kelompok yang relevan; satu angka *accuracy* rata-rata menyembunyikan masalah ini.

Hak atas data. Data yang dapat diunduh dari internet belum tentu boleh dipakai melatih model. Periksa lisensi *dataset*, syarat layanan situs, dan hak cipta isinya.

Dokumentasi.

- ***Datasheet for datasets***: motivasi, komposisi, proses pengumpulan, dan batas penggunaan data.
- ***Model card***: tujuan model, data evaluasi, kinerja per kelompok, dan keterbatasannya.

Sumber: Buolamwini dan Gebru 2018; Gebru dkk. 2021; Mitchell dkk. 2019.

Daftar periksa etika data untuk proyek *deep learning*

Aspek	Pertanyaan yang harus dapat dijawab
Dasar pemrosesan	Ada persetujuan atau dasar sah lain untuk melatih model dengan data ini?
Tujuan	Sesuaikah penggunaan ini dengan tujuan saat data dikumpulkan?
Minimisasi	Semua kolom dan contoh memang diperlukan? Pengenal langsung dapat dihapus?
Representasi	Kelompok mana yang kurang terwakili? Bagaimana kinerja per kelompok?
Lisensi	Lisensi data dan *weight pretrained* mengizinkan penggunaan ini?
Keamanan	Siapa yang dapat mengakses data dan model? Bagaimana bila bocor?
Dokumentasi	Ada *datasheet* untuk data dan *model card* untuk model?
Lingkungan	Ukuran model dan jumlah eksperimen sepadan dengan manfaatnya?

Daftar ini menjadi bagian wajib laporan tugas besar UAS.

Etika sebagai bagian dari evaluasi

Laporkan kinerja per kelompok (wilayah, gender, rentang usia, bila relevan dan sah diproses). Uji model pada data yang berbeda dari *training data* sebelum dipakai. Sediakan jalur keberatan bagi orang yang terdampak keputusan model. Bila sistem berisiko tinggi tidak dapat dijelaskan atau diuji dengan memadai, pertimbangkan untuk tidak membangunnya.

Kesalahan umum, praktikum, dan rekap

Kesalahan yang sering terjadi

Saat membangun jaringan dalam

- Menumpuk *layer* sigmoid tanpa *normalization* atau residual, lalu heran *layer* awal tidak berubah.
- Menyimpulkan "model dalam *overfitting*" padahal *training loss*-nya sendiri tinggi. Itu degradasi atau masalah *optimization*; periksa *gradient norm* per *layer*.
- Lupa *projection shortcut* ketika lebar blok berubah, sehingga `x + f(x)` gagal karena dimensi tidak cocok.
- Mengabaikan *loss* yang menjadi `nan` alih-alih memasang *gradient clipping* dan menurunkan η.

Saat membandingkan dan memakai ulang

- Membandingkan model dangkal dan dalam dengan satu η, satu *seed*, dan anggaran *epoch* berbeda.
- Tidak melaporkan jumlah parameter, sehingga efek kedalaman tercampur efek ukuran.
- Memakai *normalization* sendiri untuk model *pretrained*, atau membiarkan BN badan yang dibekukan tetap dalam mode `train()`.
- Memakai *weight* atau *dataset* tanpa memeriksa lisensi dan dasar pemrosesan datanya.

Praktikum: jaringan dangkal, dalam, dan *transfer learning*

Bagian A — dangkal lawan dalam. Pada Fashion-MNIST:

1. Jalankan `residual_mlp.py` untuk tiga varian: dangkal 1 blok, polos 20 blok, residual 20 blok. Tambahkan varian polos dan residual 40 blok.
2. Pakai tiga *seed* dan 10 *epoch*; setel η untuk tiap varian dari $\{3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}\}$ pada *validation data*.
3. Catat *gradient norm layer* pertama di awal *training* untuk setiap varian.

Bagian B — *transfer learning*. Pada CIFAR-10 yang dikurangi:

4. Jalankan `transfer_learning.py` mode `fitur` dan `finetune` dengan 1.000, 5.000, dan 20.000 citra latih.
5. Tambahkan ResNet-18 yang sama dengan `weights=None` sebagai pembanding.
6. Gambarkan *test accuracy* terhadap jumlah data untuk ketiga varian.

Tabel yang dikumpulkan (rata-rata $\pm$ simpangan baku):

Varian	Param.	η	*Train loss*	Uji
dangkal 1				
polos 20 / 40				
residual 20 / 40				

Tulis satu halaman tafsiran: apakah kedalaman membantu pada data ini, apa peran jalur identitas, dan berapa banyak data yang dihemat oleh *weight pretrained*.

Progres tugas besar UAS. Setiap kelompok menyerahkan: rumusan masalah, sumber dan lisensi data, daftar periksa etika yang sudah diisi, model patokan yang sudah berjalan, dan rencana arsitektur (dari nol atau *transfer learning*) beserta alasannya.

Dinilai: kebenaran protokol pembanding, kelengkapan tabel, dan kualitas tafsiran.

Rekap pertemuan ini

- Kedalaman memungkinkan komposisi: *k layer* dengan $2k$ neuron membangun 2^k potongan linier, yang memerlukan sekitar 2^k neuron pada jaringan dangkal. Jaringan dalam juga mempelajari *feature* bertingkat dari umum ke spesifik.
- *Gradient layer* awal adalah hasil kali banyak faktor. Sigmoid memberi $0{,}25^{20} \approx 9 \times 10^{-13}$ dalam kasus terbaik; faktor besar memberi ledakan.
- Penanganannya: ReLU, *initialization* He, *normalization*, residual, *gradient clipping*, dan *warmup*.
- Blok residual $\mathbf{x} + \mathcal{F}(\mathbf{x})$ punya *derivative* $I + \partial\mathcal{F}/\partial\mathbf{x}$. Pada contoh tiga *layer*, *gradient* polos 0,0096 dan residual 1,438.
- Arsitektur modern berpola *stem*, badan, kepala. *Cost* sekitar $6P$ FLOP per contoh dan 16 byte per parameter dengan Adam fp32.
- *Transfer learning*: *feature extraction* untuk data sangat sedikit, *fine-tuning* dengan η kecil untuk data lebih banyak atau domain berbeda.
- Etika data: dasar pemrosesan yang sah (UU PDP), tujuan, minimisasi, bias per kelompok, lisensi, *datasheet*, dan *model card*.

Tugas untuk pertemuan berikutnya

1. Baca Prince, *Understanding Deep Learning*, Bab 4 (jaringan dalam) dan Bab 11 (*residual network*), serta makalah ResNet He dkk. (2016).
2. Latihan tangan: ulangi contoh tiga *layer* hari ini dengan tanh sebagai pengganti sigmoid ($a_0 = 1$, $w = 1$). Hitung $\partial a_3 / \partial a_0$ untuk rantai polos dan residual, lalu bandingkan dengan hasil sigmoid.
3. Hitung jumlah parameter MLP residual dengan 40 blok dan lebar 512, lalu taksir memori *training*-nya dengan Adam fp32.
4. Selesaikan praktikum Bagian A dan B, dan serahkan progres tugas besar UAS.

Pertemuan depan

Semua jaringan hari ini meratakan citra menjadi vektor 784 angka dan membuang struktur ruangnya. Pertemuan 13 membahas *convolutional neural network*: konvolusi, *stride*, *padding*, dan *pooling* yang memanfaatkan struktur citra, arsitektur klasik dari LeNet sampai ResNet, dan praktikum membandingkan CNN dengan MLP pada MNIST atau CIFAR-10.

Terima kasih

Pertanyaan?