Pertemuan 11 — Melatih *Neural network* secara Stabil

Dari *initialization weight* sampai eksperimen yang dapat diulang

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Peta masalah *training*: *optimization* atau *generalization*
2. *Initialization weight*: Xavier dan He
3. *Optimizer*: *momentum*, RMSProp, Adam, AdamW
4. *Normalization*: *input*, *batch norm*, *layer norm*
5. *Regularization*: *weight decay* dan *dropout*
6. *Early stopping* dan *data augmentation*
7. Membaca kurva *loss* dan *accuracy*
8. *Tuning hyperparameter*
9. Reproduktibilitas eksperimen
10. Resep lengkap dengan PyTorch
11. Kesalahan yang sering terjadi
12. Praktikum: MLP dan tabel pembanding

Sasaran pertemuan ini

Anda dapat memilih *initialization*, *optimizer*, dan *learning rate* dengan alasan yang jelas, menerapkan *dropout*, *early stopping*, dan *normalization* dengan benar, membaca kurva *training* untuk mendeteksi *overfitting*, dan mencatat eksperimen yang dapat diulang.

Dari mesin ke kemudi

Menyambung dari Pertemuan 10

Pertemuan 10 memberi kita mesinnya: *forward pass* menghitung *prediction*, *backpropagation* menghitung *gradient* lewat aturan rantai, dan *gradient descent* memperbaiki *weight*.

Dua mahasiswa, satu MLP

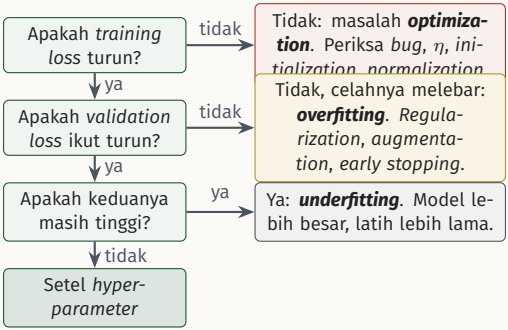
Keduanya melatih MLP yang sama pada Fashion-MNIST selama sepuluh *epoch*. Yang pertama (He + AdamW) mendapat *validation accuracy* 88,4% dalam 27 detik. Yang kedua ($N(0;\ 0{,}01^2)$ + SGD) melihat *loss*-nya macet di $2{,}3026 = \ln 10$; saat beralih ke $\eta = 1$, *loss*-nya menjadi nan.

Perbedaannya ada pada hal yang sering dianggap detail: *initialization*, *optimizer* dan *learning rate*, *normalization*, dan kapan *training* dihentikan.

Ringkasan

Tiga sasaran: *training* yang **stabil**, model yang **menggeneralisasi**, dan hasil yang **dapat dipercaya** karena dapat diulang.

Peta masalah: perbaiki *optimization* dulu, baru *generalization*



***Optimization* gagal**: *training loss* sendiri tidak turun. Sinyal mengecil atau meledak antar-*layer*, η salah, atau ada *bug*.

***Generalization* gagal**: *training loss* bagus, *validation loss* tidak. Ini *overfitting* dari Pertemuan 9.

Urutannya penting. Pastikan model **mampu** menghafal satu *mini-batch* kecil. Bila itu pun gagal, menambah *regularization* hanya memperparah keadaan.

Initialization weight: Xavier dan He

Menjaga *variance* sinyal antar-*layer*

Ambil satu neuron dengan n_{in} *input*, $z = \sum_{k=1}^{n_{\text{in}}} w_k x_k$. Bila *weight* dan *input* saling bebas, berpusat di nol, dan seragam *variance*-nya, maka

$$\text{Var}(z) = n_{\text{in}} \, \text{Var}(w) \, \text{Var}(x).$$

Agar sinyal tidak mengecil atau membesar dari *layer* ke *layer*, kita ingin $\text{Var}(z) \approx \text{Var}(x)$, artinya $n_{\text{in}} \text{Var}(w) \approx 1$. *Gradient* yang mengalir mundur tunduk pada hitungan serupa dengan n_{out}.

Xavier (Glorot, 2010)

$$\text{Var}(w) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

Kompromi arah maju dan mundur. Cocok untuk aktivasi yang hampir linier di sekitar nol: tanh, sigmoid.

He dkk., 2015 (*initialization* Kaiming)

$$\text{Var}(w) = \frac{2}{n_{\text{in}}}$$

ReLU membuang separuh sinyal (semua nilai negatif menjadi nol), sehingga *variance weight* dilipatduakan untuk menggantinya.

Ambil dari normal dengan *variance* itu, atau seragam $U(-a, a)$ dengan $a = \sqrt{3 \, \text{Var}(w)}$.

Contoh terhitung: *layer* pertama MLP MNIST (784–256)

Tiga pilihan skala

- Xavier:
  $\sigma_w = \sqrt{2/(784+256)} = \sqrt{0{,}00192} = 0{,}0439$
- He: $\sigma_w = \sqrt{2/784} = \sqrt{0{,}00255} = 0{,}0505$
- “Acak kecil” $\sigma_w = 0{,}01$:

$$\mathrm{Var}(z) = 784 \times 10^{-4}\,\mathrm{Var}(x) = 0{,}078\,\mathrm{Var}(x)$$

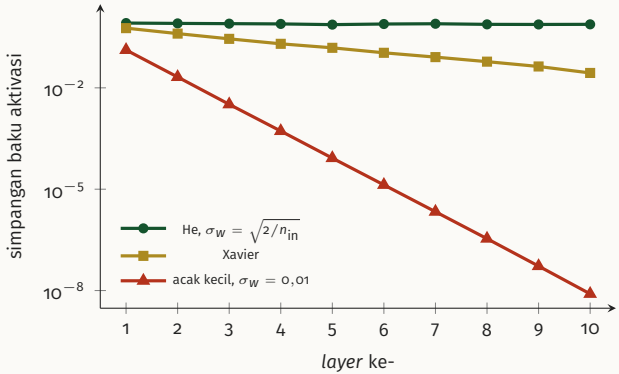
Simpangan baku menyusut $1/\sqrt{0{,}0784} \approx 3{,}6$ kali di setiap *layer*, belum termasuk separuh yang dibuang ReLU.

Batas seragam Xavier: $a = \sqrt{6/1040} = 0{,}076$.

Aktivasi	Skema	PyTorch
tanh, sigmoid	Xavier	`xavier_normal_`
ReLU, Leaky ReLU	He	`kaiming_normal_`
GELU, SiLU	He atau bawaan	bawaan `nn.Linear`
Bias	nol	`zeros_`

Bawaan `nn.Linear` di PyTorch sudah memakai skala yang bergantung pada n_{in}, sehingga jaringan dangkal biasanya aman. Atur sendiri ketika jaringannya dalam, aktivasinya tidak biasa, atau Anda mereplikasi makalah yang menyebut skema tertentu.

Akibatnya pada sepuluh *layer* ReLU



Lebar 512, *input* normal baku, 1000 contoh. Sumbu tegak berskala logaritma.

- **Acak kecil**: menyusut sekitar enam kali per *layer*, tinggal 10^{-8} di *layer* kesepuluh. Sinyalnya hilang, *gradient*-nya juga.
- **Xavier**: jauh lebih baik, tetapi masih menyusut karena tidak memperhitungkan ReLU.
- **He**: stabil di sekitar 0,8 sampai *layer* terakhir.

Ketika jaringan memakai *batch normalization* atau *residual connection* (Pertemuan 12), kepekaan terhadap *initialization* berkurang banyak.

Kode: eksperimen skala *initialization*

```python
import numpy as np

rng = np.random.default_rng(0)
x = rng.normal(size=(1000, 512))
skema = {
    "kecil":  lambda i, o: 0.01,
    "Xavier": lambda i, o: np.sqrt(2.0 / (i + o)),
    "He":     lambda i, o: np.sqrt(2.0 / i),
}
for nama, std in skema.items():
    a, hasil = x, []
    for lapis in range(10):
        W = rng.normal(0, std(512, 512), (512, 512))
        a = np.maximum(a @ W, 0.0)        # ReLU
        hasil.append(a.std())
    print(nama, ["%.2e" % s for s in hasil])
```

Enam belas baris inilah yang menghasilkan grafik di slide sebelumnya.

Coba sendiri:

- Ganti ReLU dengan `np.tanh`. Skema mana yang sekarang paling stabil?
- Naikkan jumlah *layer* menjadi 50. Apa yang terjadi pada Xavier?
- Ganti σ_w He dengan $\sqrt{4/n_{\text{in}}}$. Sinyalnya meledak secepat apa?

Sumber: `sample-code-buku`, `bab13-melatih-jaringan-saraf/init_eksperimen.py`.

Optimizer: dari SGD ke Adam

Dua kelemahan SGD biasa, dan *momentum*

Pertemuan 9 sudah memperlihatkan dua kelemahan *gradient descent*:

1. Pada lembah sempit (*condition number* besar) *gradient* menunjuk ke dinding lembah, sehingga lintasannya memantul dan maju lambat di sepanjang dasar.
2. Satu η dipakai untuk semua parameter, padahal skala *gradient*-nya berbeda-beda. Pada *neural network* dengan jutaan parameter, perbedaan itu besar sekali.

Momentum (Polyak, *heavy ball*)

$$\mathbf{v}_t = \beta\,\mathbf{v}_{t-1} + \mathbf{g}_t, \qquad \theta_{t+1} = \theta_t - \eta\,\mathbf{v}_t$$

$\mathbf{g}_t$ *gradient mini-batch*, β biasanya 0,9.

Seperti bola menggelinding: komponen yang bolak-balik saling meniadakan, komponen yang searah menumpuk. Bila *gradient*-nya konsisten, $\mathbf{v} \to \mathbf{g}/(1-\beta) = 10\,\mathbf{g}$, jadi langkah efektifnya sepuluh kali lebih panjang.

Nesterov momentum menghitung *gradient* di titik "sesudah melangkah dengan momentum", sehingga dapat mengerem lebih awal.

RMSProp dan Adam: *learning rate* per parameter

RMSProp

$$\mathbf{s}_t = \rho\,\mathbf{s}_{t-1} + (1-\rho)\,\mathbf{g}_t^2$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\,\frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \epsilon}$$

Semua operasi per elemen. Parameter bergradien besar mendapat langkah kecil, dan sebaliknya. $\epsilon \approx 10^{-8}$ mencegah pembagian dengan nol.

Adam = *momentum* + RMSProp + koreksi bias

Bawaan: $\eta = 10^{-3}$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\epsilon = 10^{-8}$; $\mathbf{m}_0 = \mathbf{s}_0 = \mathbf{0}$. Untuk $t = 1, 2, \ldots$:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \qquad \textit{first moment}$$

$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \qquad \textit{second moment}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \quad \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1-\beta_2^t} \qquad \textit{bias correction}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}$$

Mengapa perlu koreksi bias? **m** dan **s** dimulai dari nol, sehingga pada iterasi awal keduanya jauh lebih kecil dari nilai sebenarnya. Pembagian dengan $1 - \beta^t$ mengembalikan skalanya; efek koreksi ini lenyap setelah beberapa ratus iterasi.

Sumber: Kingma dan Ba, *Adam: A Method for Stochastic Optimization*, ICLR 2015.

Contoh terhitung: dua langkah Adam pada satu parameter

Ambil $\theta_0 = 1$, $\eta = 0{,}1$, nilai bawaan lainnya. *Gradient* $g_1 = 0{,}5$ lalu $g_2 = 0{,}3$.

Langkah 1

$m_1 = 0{,}1 \times 0{,}5 = 0{,}05$
$s_1 = 0{,}001 \times 0{,}25 = 0{,}00025$

Tanpa koreksi: $0{,}1 \times 0{,}05/\sqrt{0{,}00025} = 0{,}316$. Terlalu besar, karena s_1 jauh terlalu kecil.

Dengan koreksi: $\hat{m}_1 = 0{,}05/0{,}1 = 0{,}5$,
$\hat{s}_1 = 0{,}00025/0{,}001 = 0{,}25$
Langkah $= 0{,}1 \times 0{,}5/\sqrt{0{,}25} = 0{,}1$

$\theta_1 = 1 - 0{,}1 = \mathbf{0{,}900}$

Langkah 2

$m_2 = 0{,}9(0{,}05) + 0{,}1(0{,}3) = 0{,}075$
$s_2 = 0{,}999(0{,}00025) + 0{,}001(0{,}09) = 0{,}00034$

$\hat{m}_2 = 0{,}075/(1 - 0{,}81) = 0{,}075/0{,}19 = 0{,}395$
$\hat{s}_2 = 0{,}00034/(1 - 0{,}998001) = 0{,}170$

Langkah $= 0{,}1 \times 0{,}395/\sqrt{0{,}170} = 0{,}096$

$\theta_2 = 0{,}900 - 0{,}096 = \mathbf{0{,}804}$

Bandingkan dengan *momentum* ($\eta = 0{,}1$, $\beta = 0{,}9$) pada *gradient* yang sama: $v_1 = 0{,}5$, $\theta_1 = 0{,}95$; $v_2 = 0{,}9(0{,}5) + 0{,}3 = 0{,}75$, $\theta_2 = 0{,}875$.

Ringkasan

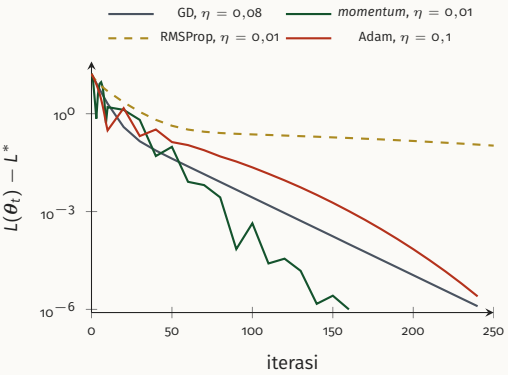
Langkah pertama Adam tepat η, berapa pun *gradient*-nya (0,5 atau 500). Karena itu Adam tidak peka terhadap skala *loss* dan cocok sebagai pilihan awal.

AdamW, dan tidak ada *optimizer* yang selalu menang

Pada SGD, *regularization* L2 dan *weight decay* setara. Pada Adam tidak: suku $\lambda\boldsymbol{\theta}$ yang ditambahkan ke *gradient* ikut dibagi $\sqrt{\hat{\mathbf{s}}}$, sehingga parameter bergradien besar hampir tidak dihukum. **AdamW** memisahkan keduanya:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta\left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t}+\epsilon} + \lambda\,\boldsymbol{\theta}_{t-1}\right)$$

Di PyTorch pakai `torch.optim.AdamW`; jangan `Adam` dengan argumen `weight_decay`.



Grafik kanan: MSE data jam belajar dari Pertemuan 9 (*condition number* ≈ 70). Iterasi untuk mencapai selisih MSE 10^{-4}: *momentum* 84, GD terbaik 161, Adam 196, RMSProp lebih dari 600.

Pada masalah kecil yang konveks dengan η yang disetel rapi, GD dan *momentum* sangat kompetitif. Keunggulan Adam muncul pada jaringan besar, ketika menyetel η per parameter secara manual mustahil.

Memilih *optimizer* dan *learning rate*

Optimizer	Status	η awal	Catatan
SGD	–	0,01–0,1	sederhana; perlu jadwal η; jarang dipakai tanpa momentum
SGD + *momentum*	**v**	0,01–0,1	*generalization* terbaik pada CNN citra bila disetel rapi; peka η
RMSProp	**s**	10^{-3}	laju adaptif; dulu populer untuk RNN
Adam	**m**, **s**	10^{-3}	pilihan awal yang aman; tidak peka skala *gradient*
AdamW	**m**, **s**	10^{-4}–10^{-3}	bawaan untuk *transformer* dan *fine-tuning*; *weight decay* 0,01–0,1

Resep awal yang aman

AdamW dengan $\eta = 10^{-3}$ untuk model kecil yang dilatih dari nol (10^{-4} sampai 3×10^{-5} untuk *fine-tuning* model besar), jadwal *cosine*, dan *warmup* singkat.

Ringkasan

Learning rate tetap *hyperparameter* terpenting; setel dalam skala logaritma. Adam menyimpan dua vektor seukuran parameter, jadi memori *optimizer*-nya dua kali ukuran model.

Kode: *momentum* dan Adam dari nol dengan NumPy

```python
def langkah_momentum(b, g, st, eta, t, beta=0.9):
    st["v"] = beta * st.get("v", 0) + g      # kecepatan
    return b - eta * st["v"]

def langkah_adam(b, g, st, eta, t,
                 b1=0.9, b2=0.999, eps=1e-8):
    st["m"] = b1 * st.get("m", 0) + (1 - b1) * g     # momen 1
    st["s"] = b2 * st.get("s", 0) + (1 - b2) * g**2  # momen 2
    m_hat = st["m"] / (1 - b1 ** t)       # koreksi bias
    s_hat = st["s"] / (1 - b2 ** t)
    return b - eta * m_hat / (np.sqrt(s_hat) + eps)

def jalankan(langkah, eta, T=5000):
    b, st = np.zeros(2), {}
    for t in range(1, T + 1):
        b = langkah(b, grad(b), st, eta, t)
        if mse(b) < 0.4801:          # MSE minimum 0.48
            return t, b
    return None, b
```

`grad` dan `mse` adalah fungsi Pertemuan 9 untuk data jam belajar, dengan $\mathbf{b} = (\beta_0, \beta_1)$.

Kamus `st` menyimpan status *optimizer*, persis seperti `torch.optim` menyimpan *state* per parameter.

Periksa implementasi Anda dengan contoh terhitung tadi: dengan $g = 0{,}5$ lalu $0{,}3$ dan $\eta = 0{,}1$, `langkah_adam` harus memberi $0{,}900$ lalu $0{,}804$.

Sumber: `sample-code-buku`, `bab13-melatih-jaringan-saraf/optimizer_numpy.py`.

Normalization

Dari *normalization input* ke *batch normalization*

Input. *Feature* berbeda skala membuat permukaan *loss* pipih (Pertemuan 9). Standarkan setiap *feature* dengan statistik *training data*. Untuk citra: bagi 255, lalu *normalization* per kanal.

Hidden layer. *Input* ke *layer* berikutnya adalah aktivasi *layer* sebelumnya, dan distribusinya berubah terus karena *weight*-nya berubah. Ioffe dan Szegedy (2015) mengusulkan ***batch normalization*** (BN): normalisasikan *pre-activation* setiap neuron dengan statistik *mini-batch*.

BN untuk satu neuron dengan nilai $z_1, \ldots, z_B$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i, \quad \sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2, \quad \hat{z}_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad y_i = \gamma\,\hat{z}_i + \beta$$

γ (skala) dan β (geser) dipelajari seperti *weight* lain; tanpa keduanya setiap *layer* dipaksa normal baku. Dengan $\gamma = \sigma_B$, $\beta = \mu_B$, BN dapat membatalkan dirinya sendiri.

Contoh terhitung: BN pada empat nilai, dan mode latih–inferensi

Satu neuron, *mini-batch* berisi empat

$z = (2, 4, 6, 8)$, sehingga $\mu_B = 5$ dan

$$\sigma_B^2 = \tfrac{1}{4}(9 + 1 + 1 + 9) = 5.$$

Dengan ϵ diabaikan:

$$\hat{z} = \frac{z - 5}{\sqrt{5}} = (-1{,}342;\ -0{,}447;\ 0{,}447;\ 1{,}342)$$

Bila $\gamma = 2$, $\beta = 1$:

$$y = 2\hat{z} + 1 = (-1{,}683;\ 0{,}106;\ 1{,}894;\ 3{,}683)$$

Rata-rata *output* 1, simpangan baku 2. Keduanya kini ditentukan oleh β dan γ, terlepas dari *weight* pada *layer* sebelumnya.

Saat inferensi kita sering memproses satu contoh saja, sehingga tidak ada rata-rata *batch*. BN menyimpan ***running mean*** dan *running variance* selama *training* dan memakainya saat inferensi.

Lupa `model.eval()`

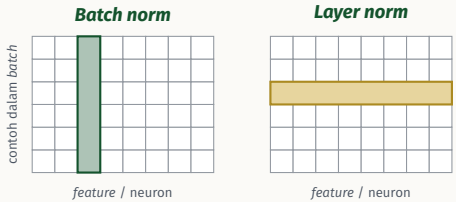
Dalam mode `train()`, BN memakai statistik *batch* uji dan *dropout* tetap aktif. Hasilnya berubah-ubah bergantung isi *batch*, dan biasanya lebih buruk. Panggil `model.eval()` sebelum validasi atau inferensi, dan `model.train()` sebelum melanjutkan *training*. BN juga bermasalah pada *batch* sangat kecil (misalnya 2), karena μ_B dan σ_B^2 terlalu berisik.

Mengapa BN membantu, dan *layer normalization*

Penjelasan awalnya adalah pengurangan *internal covariate shift*. Penelitian lanjutan (Santurkar dkk., 2018) memberi penjelasan yang lebih meyakinkan: BN membuat permukaan *loss* lebih *smooth*, sehingga *gradient* lebih dapat diandalkan dan η yang lebih besar dapat dipakai.

Efek praktisnya: *training* lebih cepat, kurang peka *initialization*, dan sedikit efek *regularization* dari *noise* statistik *batch*.

Layer normalization (LN) menghitung rata-rata dan *variance* di sepanjang ***feature*** untuk setiap contoh. Ia berperilaku sama saat latih dan inferensi, tahan *batch* kecil, dan menjadi bawaan *transformer* (Pertemuan 14). BN tetap umum di CNN (Pertemuan 13).



Daerah yang dipakai menghitung rata-rata dan *variance*. BN: satu *feature* di seluruh contoh *batch* (kolom hijau). LN: seluruh *feature* dalam satu contoh (baris emas).

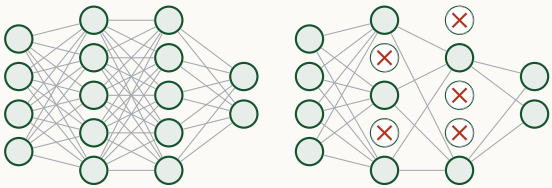
Regularization dan *generalization*

Weight decay dan *dropout*

Weight decay adalah *regularization* L2 dari Pertemuan 9 pada *weight* jaringan. *Weight* kecil membuat fungsi jaringan lebih *smooth*. Pada AdamW, λ lazimnya 10^{-4} sampai 10^{-1}; bias dan parameter *normalization* biasanya dikecualikan.

Dropout (Srivastava dkk., 2014): pada setiap langkah *training*, setiap neuron dimatikan secara acak dengan peluang p. Jaringan tidak dapat bergantung pada satu neuron atau kerja sama rapuh antarneuron (*co-adaptation*), sehingga representasinya tersebar.

Tafsiran lain: 2^N sub-jaringan berbagi *weight*; inferensi dengan seluruh neuron menghampiri rata-rata *ensemble* itu, mirip *bagging* (Pertemuan 4).



jaringan penuh (inferensi) — satu *training step*, $p = 0{,}5$

Setiap langkah melatih sub-jaringan yang berbeda. Nilai p lazim 0,1–0,5 untuk *layer* terhubung penuh; pada CNN yang memakai BN, *dropout* sering dikurangi.

Contoh terhitung: *inverted dropout*

Bila separuh neuron mati saat latih tetapi semua aktif saat inferensi, sinyal ke *layer* berikutnya berlipat dua saat inferensi. ***Inverted dropout*** (`nn.Dropout`) mengalikan neuron yang bertahan dengan $1/(1-p)$ saat latih, dan tidak mengubah apa pun saat inferensi.

$p = 0{,}5$, aktivasi $\mathbf{h} = (0{,}8;\ 0{,}2;\ 1{,}0;\ 0{,}4)$

Topeng acak yang terambil: $\mathbf{r} = (1, 0, 1, 0)$. *Output* saat latih:

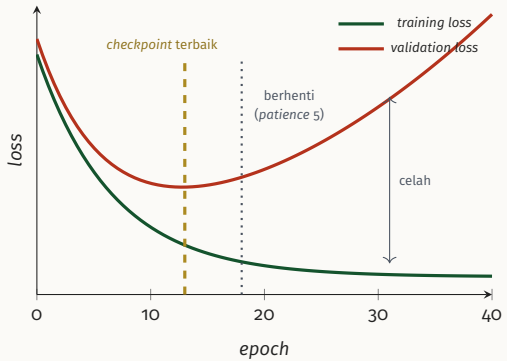
$$\frac{\mathbf{h} \odot \mathbf{r}}{1 - 0{,}5} = (1{,}6;\ 0;\ 2{,}0;\ 0)$$

Rata-rata aktivasi asli 0,6. Untuk topeng ini rata-rata *output*-nya 0,9, tetapi dirata-ratakan atas semua $2^4 = 16$ topeng yang mungkin, nilai harapannya tepat 0,6. Saat inferensi *output*-nya **h** apa adanya.

Ringkasan

Penskalaan saat latih membuat kode inferensi sederhana: cukup panggil `model.eval()`.

Early stopping



Training loss hampir selalu terus turun. *Validation loss* turun, lalu mulai naik ketika model mulai menghafal.

Early stopping memantau *validation loss* setiap *epoch*, menyimpan model terbaik sejauh ini (*checkpoint*), dan berhenti bila tidak ada perbaikan selama ***patience*** *epoch*. *Weight* yang dipakai adalah *checkpoint* terbaik.

Murah, hampir selalu membantu, dan bekerja sebagai *regularization*: jumlah *epoch* membatasi seberapa jauh *weight* bergerak dari titik awal.

Syarat mutlak

Keputusan berhenti diambil dari *validation data*. *Test data* tidak boleh ikut.

Data augmentation

Obat paling ampuh melawan *overfitting* adalah data yang lebih banyak. Bila data baru mahal, buat variasi dari data yang ada dengan *transformation* yang **tidak mengubah label**.



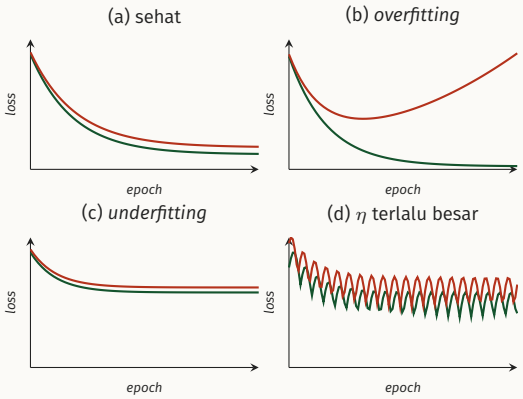
Untuk citra: pencerminan horizontal, *random crop*, rotasi kecil, *color jitter*, *random erasing*. *Mixup* mencampur dua citra beserta labelnya secara linier. Semua contoh di atas tetap berlabel "sepatu".

Augmentation yang mengubah label

Rotasi 180° mengubah "6" menjadi "9" pada MNIST. Mencerminkan citra rontgen dada memindahkan jantung ke kanan. Untuk teks dan data tabular, *augmentation* jauh lebih sulit dan sering tidak dianjurkan. *Augmentation* hanya untuk *training data*; *validation* dan *test data* dibiarkan apa adanya.

Membaca kurva *training*

Empat pola kurva yang paling sering muncul



Hijau: *training loss*. Merah: *validation loss*. Ilustrasi.

(a) Keduanya turun berdampingan, celahnya kecil. Lanjutkan, atau perbesar model.

(b) Celahnya melebar dan *validation loss* naik. *Regularization* atau berhenti lebih awal.

(c) Keduanya macet di nilai tinggi. Model terlalu kecil, η terlalu kecil, atau *regularization* berlebihan.

(d) Bergerigi besar dan tidak turun mantap. Turunkan η atau pakai jadwal.

Biasakan selalu menggambar *training curve* dan validasi. Angka akhir saja menyembunyikan cerita bagaimana model sampai ke sana.

Dari gejala ke tindakan

Gejala	Kemungkinan penyebab	Tindakan
Loss `nan` atau `inf`	η terlalu besar; log 0; data rusak	turunkan η; pakai *loss* yang stabil (`...WithLogits`); periksa data; *gradient clipping*
Loss awal jauh dari $\ln K$	*initialization* atau label salah	periksa label, aktivasi *output*, skala *input*
Latih turun, validasi naik	*overfitting*	*early stopping*, *dropout*, *weight decay*, *augmentation*, data lebih banyak
Keduanya tinggi dan datar	*underfitting*; η terlalu kecil	model lebih besar, η lebih besar, latih lebih lama, kurangi *regularization*
Kurva bergerigi besar	η terlalu besar; *batch* terlalu kecil	turunkan η, pakai jadwal, besarkan *batch*
Validasi lebih baik dari latih	*dropout* dan *augmentation* hanya aktif saat latih	wajar bila celahnya kecil
Validasi melonjak acak	*validation set* kecil; kebocoran	besarkan *validation set*; periksa *split*

Dua pemeriksaan kewarasan sebelum *training* panjang

1. *Loss* awal sekitar $\ln K$

Model yang belum belajar memberi peluang kira-kira seragam $1/K$ untuk setiap kelas. *Cross-entropy*-nya

$$-\ln \tfrac{1}{K} = \ln K.$$

Untuk Fashion-MNIST, $K = 10$ sehingga $\ln 10 = 2{,}30$: angka yang macet pada mahasiswa kedua tadi. Modelnya tidak belajar sama sekali.

2. Hafalkan satu *mini-batch*

Ambil beberapa puluh contoh, matikan *regularization*, lalu latih sampai *loss*-nya hampir nol. Model dengan ratusan ribu parameter pasti sanggup menghafal 32 contoh. Bila tidak bisa, ada *bug*: label teracak, `zero_grad` terlupa, atau aktivasi *output* ganda.

Ringkasan

Dua pemeriksaan lima menit ini menghemat berjam-jam *training* sia-sia. Baca *accuracy* bersama *loss*: bila keduanya naik, model makin yakin pada *prediction* yang salah.

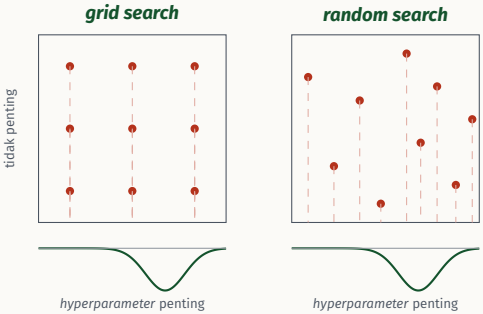
Tuning hyperparameter

Apa yang disetel, dan mengapa *random search*

Hyperparameter tidak dipelajari oleh *gradient descent*. Urutan prioritas yang lazim:

1. *learning rate* dan jadwalnya;
2. kapasitas model: kedalaman dan lebar;
3. *regularization*: *weight decay*, *dropout*, *augmentation*;
4. ukuran *batch*, yang berinteraksi dengan η;
5. β_1, β_2, yang jarang perlu diubah.

Bergstra dan Bengio (2012): *random search* biasanya jauh lebih efisien daripada *grid search*. Bila hanya satu dari dua *hyperparameter* yang berpengaruh, *grid* 3×3 hanya mencicipi tiga nilai penting, sedangkan sembilan titik acak mencicipi sembilan.



Kurva hijau di bawah: kinerja terhadap *hyperparameter* penting, puncaknya sekitar 2,7. *Random search* lebih mungkin mendekati puncak itu.

Metode berbasis model memakai hasil percobaan sebelumnya: *Bayesian optimization* dan TPE. *Hyperband* dan *median pruning* menghentikan dini percobaan yang tertinggal.

Kode: *tuning* dengan Optuna (TPE dan *pruning*)

```
import optuna
def objektif(trial):
    lr = trial.suggest_float("lr", 1e-4, 1e-2, log=True)
    p_drop = trial.suggest_float("dropout", 0.0, 0.5)
    lebar = trial.suggest_categorical("lebar", [256, 512])
    wd = trial.suggest_float("wd", 1e-6, 1e-2, log=True)
    model = buat_model(lebar, p_drop)
    opt = torch.optim.AdamW(model.parameters(), lr=lr,
                            weight_decay=wd)
    for epoch in range(8):
        latih_satu_epoch(model, opt, dl_latih)
        akurasi = evaluasi(model, dl_val)  # data validasi
        trial.report(akurasi, epoch)
        if trial.should_prune():           # tertinggal: stop
            raise optuna.TrialPruned()
    return akurasi
studi = optuna.create_study(direction="maximize",
            sampler=optuna.samplers.TPESampler(seed=0),
            pruner=optuna.pruners.MedianPruner())
studi.optimize(objektif, n_trials=30)
```

Perhatikan `log=True`: η dan *weight decay* dicari dalam skala logaritma, karena selisih 10^{-4} ke 10^{-3} sama pentingnya dengan 10^{-3} ke 10^{-2}.

Kebocoran lewat *tuning*

Setiap kali Anda memilih *hyperparameter* berdasarkan sebuah data, data itu ikut "dipakai belajar". Setel dengan *validation set* atau *cross-validation*, dan sentuh *test set* sekali saja di akhir (Pertemuan 9).

Sumber: `sample-code-buku`, `bab13-melatih-jaringan-saraf/tuning_optuna.py`.

Reproduktibilitas eksperimen

Mengunci sumber keacakan

```python
import random
import numpy as np
import torch

def set_seed(seed):
    random.seed(seed)                    # Python
    np.random.seed(seed)                 # NumPy
    torch.manual_seed(seed)              # PyTorch CPU
    torch.cuda.manual_seed_all(seed)     # semua GPU
    torch.backends.cudnn.benchmark = False
    torch.use_deterministic_algorithms(True,
                                       warn_only=True)

set_seed(42)
g = torch.Generator().manual_seed(42)  # urutan DataLoader
dl = DataLoader(latih, batch_size=128,
                shuffle=True, generator=g)
```

Sumber keacakan dalam *training neural network*: *initialization weight*, urutan data, *dropout*, *augmentation*, pembagian *train*/*validation*, dan algoritma GPU tertentu yang hasil penjumlahannya bergantung urutan eksekusi.

Dokumentasi PyTorch mencatat bahwa hasil identik bit demi bit tidak dijamin antarversi, antarplatform, atau antara CPU dan GPU. Algoritma deterministik juga bisa lebih lambat. *Seed* saja belum cukup.

Reproduktibilitas yang realistis, dan tabel pembanding

1. **Kunci versi**: `requirements.txt` atau `uv.lock`; catat versi CUDA dan jenis GPU.
2. **Versikan data dan kode**: *commit* Git untuk kode, pengenal versi atau *hash* untuk data.
3. **Catat setiap eksperimen**: *hyperparameter*, *metric* per *epoch*, *seed*, lama latih, lokasi *checkpoint*. MLflow atau Weights & Biases mencatat otomatis; CSV sederhana pun jauh lebih baik daripada tidak ada.
4. **Laporkan variasi**: beberapa *seed* (misalnya lima), rata-rata $\pm$ simpangan baku.

Format tabel pembanding: satu baris satu *configuration*, satu kolom satu *hyperparameter* atau *metric*. Simbol $a_i \pm s_i$ diisi hasil Anda sendiri.

ID	Opt.	η	*Drop*	BN	*Accuracy* val.	*Ep.*
E01	SGD	0,1	0	tidak	$a_1 \pm s_1$	e_1
E02	AdamW	10^{-3}	0	tidak	$a_2 \pm s_2$	e_2
E03	AdamW	10^{-3}	0	ya	$a_3 \pm s_3$	e_3
E04	AdamW	10^{-3}	0,3	ya	$a_4 \pm s_4$	e_4

Ringkasan

Selisih 0,3% antara dua *configuration* tidak bermakna bila variasi antar-*seed* sendiri 0,5%. Ingat uji perbandingan dua algoritma dari Pertemuan 9.

Menyatukan semuanya dengan PyTorch

Model: MLP dengan BN, *dropout*, dan *initialization* He

```python
import torch
from torch import nn

def buat_model(lebar=512, p_drop=0.3):
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, lebar), nn.BatchNorm1d(lebar),
        nn.ReLU(), nn.Dropout(p_drop),
        nn.Linear(lebar, lebar // 2),
        nn.BatchNorm1d(lebar // 2),
        nn.ReLU(), nn.Dropout(p_drop),
        nn.Linear(lebar // 2, 10),   # logit, tanpa softmax
    )
    for m in model.modules():        # inisialisasi He
        if isinstance(m, nn.Linear):
            nn.init.kaiming_normal_(m.weight,
                                    nonlinearity="relu")
            nn.init.zeros_(m.bias)
    return model

loss_fn = nn.CrossEntropyLoss()
```

Data: Fashion-MNIST, 70.000 citra pakaian 28×28 dalam sepuluh kelas. Ia dirancang sebagai pengganti MNIST yang lebih menantang.

Urutan per blok: `Linear` $\rightarrow$ `BatchNorm1d` $\rightarrow$ `ReLU` $\rightarrow$ `Dropout`. BN dipasang pada *pre-activation*, sesuai rumusnya.

Layer terakhir mengeluarkan logit. `nn.CrossEntropyLoss` sudah memuat *softmax* di dalamnya; menambah `Softmax` sendiri adalah kesalahan klasik.

Lingkar latih: AdamW, jadwal *cosine*, dan *early stopping*

```python
from torch.optim.lr_scheduler import CosineAnnealingLR
opt = torch.optim.AdamW(model.parameters(), lr=1e-3,
                        weight_decay=1e-4)
sched = CosineAnnealingLR(opt, T_max=n_epoch)
terbaik, sabar = float("inf"), 0
for epoch in range(1, n_epoch + 1):
    model.train()                # dropout dan BN mode latih
    for xb, yb in dl_latih:
        opt.zero_grad()
        loss_fn(model(xb), yb).backward()
        opt.step()
    loss_val, akurasi_val = evaluasi(model, dl_val)
    sched.step()
    if loss_val < terbaik - 1e-4:    # ada perbaikan
        terbaik, sabar = loss_val, 0
        torch.save(model.state_dict(), "terbaik.pt")
    else:
        sabar += 1
        if sabar >= 5:           # patience = 5 epoch
            break
model.load_state_dict(torch.load("terbaik.pt"))
```

Tiga hal yang sering terlupa:

- `opt.zero_grad()` di setiap langkah, karena PyTorch menjumlahkan *gradient*.
- `evaluasi` memanggil `model.eval()` dan `torch.no_grad()`.
- Setelah berhenti, muat ulang *checkpoint* terbaik. *Weight* pada *epoch* terakhir sudah lima *epoch* melewati titik terbaik.

Sumber: `sample-code-buku`, `bab13-melatih-jaringan-saraf/latih_fashion.py`.

Fungsi evaluasi, dan di mana semuanya berjalan

```python
@torch.no_grad()
def evaluasi(model, dl):
    model.eval()              # BN: running stats
    total, benar, n = 0.0, 0, 0
    for xb, yb in dl:
        logit = model(xb)
        total += loss_fn(logit, yb).item() * len(yb)
        benar += (logit.argmax(1) == yb).sum().item()
        n += len(yb)
    return total / n, benar / n
```

Komponen	Pilihan	Alternatif
Framework	PyTorch 2.x	JAX, Keras 3
Lingkar latih	Lightning	manual
Mixed precision	`torch.amp`	fp32
Pencatat	MLflow	W&B
Tuning	Optuna	Ray Tune
Versi data	DVC	lakeFS
Configuration	Hydra	`argparse`

Untuk praktikum, lingkar manual dan *file* CSV sudah cukup. Alat di kanan dipakai ketika eksperimen berjumlah puluhan dan dijalankan di mesin GPU bersama.

Kesalahan yang sering terjadi

Kesalahan yang sering terjadi

Pada kode

- Lupa `model.eval()` saat validasi, sehingga *dropout* masih aktif dan BN memakai statistik *batch* uji.
- Lupa `opt.zero_grad()`: *gradient* menumpuk dari langkah ke langkah.
- Menambah `Softmax` sebelum `CrossEntropyLoss`, atau `Sigmoid` sebelum `BCEWithLogitsLoss`.
- Memakai `Adam(weight_decay=...)` dan mengira itu AdamW.
- Menormalkan dengan rata-rata seluruh data, termasuk validasi dan uji.

Pada eksperimen

- Menambah *regularization* ketika *training loss* sendiri belum turun.
- Memilih *epoch* atau *hyperparameter* berdasarkan *test set*.
- Mengaugmentasi *validation* dan *test data*, atau memakai *augmentation* yang mengubah label.
- Mencari η dalam skala linier (0,001; 0,002; . . .) alih-alih logaritma.
- Menyimpulkan "*configuration* B lebih baik" dari satu *seed*.
- Memakai *weight epoch* terakhir setelah *early stopping*.

Hampir semua kesalahan di kiri tertangkap oleh dua pemeriksaan kewarasan tadi; hampir semua di kanan tertangkap oleh tabel pembanding yang jujur.

Praktikum dan penutup

Praktikum: melatih MLP dan mencatat eksperimen

Data. Fashion-MNIST (`torchvision.datasets`) atau satu data tabular pilihan Anda, misalnya *Adult Income* atau *Covertype*. Bagi menjadi *training*, *validation*, dan *test*.

Langkah.

1. Pasang `set_seed` dan *normalization input* dari *training data*.
2. Jalankan dua pemeriksaan kewarasan: *loss* awal $\approx \ln K$, dan hafalkan satu *batch* berisi 32 contoh.
3. Latih empat *configuration*, masing-masing tiga *seed*:
   - E01: SGD, tanpa BN dan *dropout*;
   - E02: AdamW, tanpa BN dan *dropout*;
   - E03: AdamW dengan BN;
   - E04: AdamW, BN, dan *dropout* 0,3.

   Semuanya memakai *early stopping* dengan *patience* 5.
4. Catat ke CSV: *hyperparameter*, *validation accuracy*, *epoch* berhenti, lama latih.
5. Gambar kurva *loss* latih dan validasi untuk keempatnya.

Yang dikumpulkan

- *Notebook* yang dapat dijalankan ulang dari awal sampai akhir.
- Tabel pembanding dengan rata-rata $\pm$ simpangan baku atas tiga *seed*.
- Empat kurva *loss* dan diagnosis singkat tiap kurva (sehat, *overfitting*, *underfitting*, atau η salah).
- Satu paragraf: *configuration* mana yang dipilih, dengan alasan dari tabel dan kurva. *Test accuracy* dilaporkan sekali saja, untuk *configuration* terpilih.

Penilaian

Kualitas eksperimen (protokol, *seed*, tanpa kebocoran) dan kualitas dokumentasi. *Accuracy* tertinggi bukan komponen penilaian.

Rekap

- **Diagnosis dahulu**: kegagalan *optimization* (*training loss* tidak turun) berbeda dari kegagalan *generalization* (celah latih–validasi melebar).
- ***Initialization*** menjaga *variance* sinyal: Xavier $2/(n_{\text{in}} + n_{\text{out}})$ untuk tanh dan sigmoid, He $2/n_{\text{in}}$ untuk ReLU.
- ***Optimizer***: *momentum* meredam pantulan, RMSProp menyesuaikan langkah per parameter, Adam menggabungkan keduanya dengan koreksi bias, AdamW memisahkan *weight decay*.
- ***Normalization***: BN per *feature* di sepanjang *batch*, memakai statistik berjalan saat inferensi; LN per contoh.
- ***Regularization***: *dropout* dengan penskalaan $1/(1-p)$, *weight decay*, *early stopping* dengan *checkpoint* terbaik, *augmentation* yang menjaga label.
- **Kurva** latih dan validasi adalah alat diagnosis utama; *loss* awal $\approx \ln K$.
- ***Tuning***: *random search* atau TPE pada *validation set*, η dalam skala log.
- **Reproduktibilitas**: *seed*, versi *library* dan data, catatan eksperimen, beberapa *seed* per *configuration*.

Ringkasan

Jaringan yang sama dapat berhasil atau gagal karena detail *training*, dan detail itu dapat dipelajari, diperiksa, serta dicatat.

Tugas untuk pertemuan berikutnya

1. Baca Prince, *Understanding Deep Learning*, Bab 6–9, dan bab 13 buku ajar.
2. Latihan tangan, tanpa komputer:
   - Hitung σ_w Xavier dan He untuk *layer* 512–256 dan 3072–1024, serta batas seragam a-nya.
   - Lanjutkan contoh Adam sampai langkah keempat dengan $g_3 = -0{,}1$ dan $g_4 = 0{,}2$. Bandingkan dengan SGD biasa $\eta = 0{,}1$.
   - Hitung *output* BN untuk $z = (1, 1, 1, 5)$ dengan $\gamma = 1$, $\beta = 0$. Apa yang terjadi bila ukuran *batch* hanya 1?
3. Selesaikan praktikum hari ini, lalu jalankan Optuna 30 *trial* dan *grid search* berukuran sama. Bandingkan *validation accuracy* terbaik dan waktu totalnya.

Pertemuan depan

Semua teknik hari ini kita pakai pada jaringan yang jauh lebih dalam. Mengapa kedalaman membantu, mengapa *gradient* dapat lenyap atau meledak ketika *layer*-nya puluhan, bagaimana *residual connection* mengatasinya, dan bagaimana *transfer learning* menghemat data: *deep neural network* di Pertemuan 12.

Terima kasih

Pertanyaan?