Deep Learning: folha de consulta

Capítulos 1 a 23 de 2026.2, um fio por página: 1 dados, 2 métricas, 3 aprendizado e otimização, 4 regularização, profundidade e convoluções, 5 atenção e Transformers, 6 adaptação de modelos e LLMs, 7 e 8 modelos generativos, 9 capítulo a capítulo. Os números são os medidos nas aulas; o número pequeno em cada barra é o capítulo.



A regra única 2

Nada que aprende um número dos dados pode ser ajustado antes da divisão.

ajustado, então dentro do `Pipeline`	não ajustado: antes pode
imputador, scaler, vocabulário de categorias, target encoding, seleção dos k melhores, PCA, oversampling (só no fold de treino)	gráficos, `describe()`, tirar duplicatas exatas, corrigir uma unidade ou um sentinela

▲ **Armadilha:** um `scaler.fit(X_train)` escrito à mão é ajustado uma vez enquanto os folds mudam cinco vezes. Só um `Pipeline` sobrevive à validação cruzada.

Treino, validação, teste 2

conjunto	para	olhar
treino	ajustar os pesos	sempre
validação	toda escolha: arquitetura, η, quando parar, features	à vontade, ela se gasta
teste	um número em que nenhuma escolha se baseou	**uma vez**

Divida pelo que o modelo precisa generalizar

`stratify=y`	classificação, sempre; essencial com uma classe rara
`GroupKFold`	um paciente, usuário ou aparelho gera várias linhas
`TimeSeriesSplit`	linhas ordenadas: o teste é o futuro

- Um intervalo estreita como $1/\sqrt{n}$: 200 linhas de teste não separam 0,90 de 0,94 (Wilson: 0,851–0,934 contra 0,898–0,965).
- Tamanhos: $<$10 mil linhas, validação cruzada mais um teste separado; 10 mil a 1 milhão, 80/10/10; $>$1 milhão, 98/1/1.
- Uma divisão 80/20 do mesmo modelo variou de 0,717 a 0,875 em 120 sementes; 5 folds levam o desvio de 0,032 a 0,006.
- O melhor de 40 modelos equivalentes em 200 linhas: +1,4 ponto de puro ruído. Validação cruzada aninhada ou um teste final.

Vazamento de dados 2

Eu teria *exatamente isto* no momento da previsão?

entra por	medido no laboratório	honesto
uma coluna	uma consequência da resposta; uma feature com $\lvert r\rvert > 0{,}95$ é vazamento até prova em contrário	
seleção antes da divisão	5 000 colunas de ruído, ficam 20: 0,870	0,500
a divisão (grupos)	`StratifiedKFold` 0,986	0,708
o tempo	`rolling(3, center=True)`: 0,937	0,716
o pipeline	padronizar antes $+0{,}004$ de AUC; oversampling antes $+0{,}173$	

▲ **Armadilha:** o vazamento barato é o famoso. Seleção, grupos e reamostragem não aparecem no código nem levantam erro.

Tipos de feature: uma codificação é uma afirmação 2

tipo	ordem	distância	codificação segura
contínua, contagem	sim	sim	escalar (contagens: `log1p` antes)
ordinal (P/M/G)	sim	não	inteiros, com `categories=` explícito
cíclica (hora)	sim	circular	$\sin(2\pi h/24)$ e $\cos(2\pi h/24)$
nominal (cidade)	não	não	one-hot ou um embedding
identificador	não	não	**não é feature**: define os grupos

▲ **Armadilha:** label encoding numa coluna nominal: R^2 de 0,163 com 40 categorias contra 0,94 com one-hot.

Escala 2

scaler	fórmula	ajustado de, comportamento
Standard	$(x-\mu)/\sigma$	média, desvio; sem limites
MinMax	$(x-\min)/(\max-\min)$	extremos; um erro de digitação esmaga o resto
Robust	$(x-\text{med})/\text{IQR}$	quartis; ignora o erro
MaxAbs	$x/\max\lvert x\rvert$	mantém os zeros: dados esparsos
Quantile	posto, depois mapeia	só a ordem sobrevive

- Escale toda entrada numérica de qualquer coisa treinada por gradiente: sem escala 0,605; com qualquer scaler 0,89–0,91. Árvores não precisam.
- Sigmoide ou tanh na primeira camada: MinMax para $[0,1]$ ou $[-1,1]$. Esparso: nunca centralize.

▲ **Armadilha:** `fit_transform` no teste. O teste é transformado, nunca ajustado.

Qualidade dos dados 2

faltante	quem some e o custo
MCAR	ninguém em especial: só o tamanho da amostra
MAR	um grupo que outras colunas identificam: dá para corrigir com elas
MNAR	definido pelo próprio valor que falta: um viés que os dados não revelam

- Imputar pela média não corrige um viés e encolhe o espalhamento (a 0,30 do real sob MNAR). Guarde uma coluna `faltava`.
- Sentinelas se passam por dado: `-999`, `0`, `"N/A"`, `1970-01-01`.
- Outliers: o z-score usa as estatísticas que os próprios outliers corromperam (com 10% de contaminação achou 0 de 100); o IQR e o z-score modificado (mediana, MAD) resistem.
- Duplicatas dos dois lados da divisão são um gabarito: tire as duplicatas antes de dividir.
- Uma rede ajusta 40% de rótulos errados até acurácia de treino 1,000 enquanto a nota honesta cai de 0,951 para 0,639; parar cedo devolve 0,825.

Desbalanceamento de classes 2, 8

- **Conserte a métrica**: de 30% para 0,5% de positivos, a acurácia trivial sobe de 0,700 para 0,995, o ROC-AUC fica em 0,968→0,967 e a precisão média cai de 0,934 para 0,457.
- **Conserte o limiar**: pelos custos, $p^\star = C_{FP}/(C_{FP}+C_{FN})$, escolhido na validação.
- Pesos de classe, oversampling, SMOTE e undersampling não mexeram no ranking (AP 0,758–0,761) e estragaram as probabilidades (Brier 0,0088→0,0360).
- Em deep learning: `pos_weight` $\approx N_{neg}/N_{pos}$ ou focal loss quando os negativos dominam o gradiente; batches balanceados só quando os batches chegam sem positivos.

▲ **Armadilha:** reamostrar antes da divisão; e usar as probabilidades de um modelo reamostrado sem a correção de Elkan para a prior.

Alta dimensão 2

casca	uma fração $1-(1-\varepsilon)^d$ do volume; raio mediano $(1/2)^{1/d}$
bola/cubo	$\pi^{d/2}/(\Gamma(d/2{+}1)\,2^d) \to 0$
distâncias	concentram quando as coordenadas extras são ruído i.i.d.: k-NN, k-means e t-SNE perdem contraste
ângulos	vetores aleatórios ficam ortogonais, desvio do cosseno $\approx 1/\sqrt{d}$

Quem manda é a dimensão **intrínseca**, não o número de colunas. Selecione features ou reduza mantendo o sinal.

Redução de dimensionalidade 2

PCA: $\max_{\|w\|=1} w^\top \Sigma w \;=\; \min_{\|w\|=1} \frac{1}{n}\sum_i \|x_i - (x_i^\top w)w\|^2$

	PCA	t-SNE	UMAP
preserva	variância global	vizinhanças	local, um pouco global
`transform` novo	sim	não	aproximado
parâmetro	k	perplexidade	`n_neighbors`
uso	comprimir, pré-processar	olhar	olhar em escala

- `PCA(n_components=0.95)` guarda 95% da variância; é reajustado em cada fold.
- Centenas de colunas: PCA para umas 50 antes do t-SNE ou do UMAP.

▲ **Armadilha:** ler um mapa de t-SNE: o tamanho de um grupo e a distância entre grupos não querem dizer nada, os eixos também não. As coordenadas do t-SNE não são features.

Lendo uma distribuição 2

- Duas classes ao longo de uma feature: $d' = |\mu_B - \mu_A|/\sqrt{(\sigma_A^2+\sigma_B^2)/2}$; a melhor acurácia possível é $\Phi(d'/2)$.
- Assimétrica (renda, contagens, durações): `log1p`. Um teto no histograma é uma propriedade dos dados (784 distritos em 500 001).
- Uma feature por vez não é o mesmo que duas juntas: salmão e robalo se separam no plano, não em cada eixo.

Antes de acreditar num número 2

- Quanto tira o modelo trivial? Reporte ao lado do seu.
- Existe um grupo (paciente, usuário, sessão) com mais de uma linha?
- Algo foi ajustado, selecionado ou reamostrado antes da divisão?
- Cada feature existiria, com este valor, no momento da previsão?
- Um salto repentino é um incidente a investigar: *o que este modelo sabe que não deveria saber?*

Métricas e avaliação

Capítulo 8: classificação, regressão, modelos generativos e LLMs. Uma métrica é uma decisão sobre como somar o erro.

A matriz de confusão 8.1

	de fato positivo	de fato negativo
previsto positivo	TP: acerto	FP: alarme falso (tipo I)
previsto negativo	FN: perda (tipo II)	TN: rejeição correta

$$\text{Precisão} = \frac{TP}{TP+FP} \qquad \text{Revocação} = \frac{TP}{TP+FN}$$

$$\text{Especificidade} = \frac{TN}{TN+FP}$$

$$F_1 = \frac{2PR}{P+R} \qquad F_\beta = (1+\beta^2)\frac{PR}{\beta^2 P+R}$$

$$\text{MCC} = \frac{TP\cdot TN - FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

- Leia o denominador: a precisão divide pelo que o *modelo* afirmou, a revocação (recall) pelo que a *realidade* tem.
- Acurácia balanceada = (revocação + especificidade)/2. Prevalência = $(TP+FN)/N$.
- F_2: a revocação pesa o dobro (triagem). $F_{0,5}$: a precisão pesa o dobro (busca). O F1 nunca vê o TN; o MCC usa as quatro células.

Armadilha: 99% de acurácia numa doença que atinge 1 em 100 é a nota de `return 0`. Pergunte primeiro quanto tira o classificador trivial.

Escores, limiares e curvas 8.1

$$\text{AUC} = P\big(s(x^+) > s(x^-)\big) \qquad \text{AP} = \textstyle\sum_n (R_n - R_{n-1})P_n$$

- A ROC põe a revocação contra a FPR $= FP/(FP+TN)$: cega à prevalência. Com positivos raros a ROC elogia e a PR conta a verdade; a linha de base da AP é a própria prevalência.
- O limiar não faz parte do modelo. Os custos esperados dão $p > p^\star = C_{FP}/(C_{FP}+C_{FN})$: se uma perda custa $9\times$ um alarme falso, marque acima de 10%.

Armadilha: ajustar o limiar no teste. Escolha na validação, reporte no teste.

Qual métrica 8.1

situação	use
balanceado, erros custam o mesmo	acurácia
perder um positivo é caro (doença, fraude)	revocação, F_2
um alarme falso é caro (spam)	precisão, $F_{0,5}$
positivos raros, comparar modelos	PR-AUC (AP)
comparar rankings, mais ou menos balanceado	ROC-AUC
a probabilidade é usada como número	log loss, Brier, calibração
multiclasse desbalanceado	F1 macro, acurácia balanceada

Multiclasse: a média *é* a métrica

F1 por classe de 0,970, 0,675 e 0,261 (suporte 900, 90 e 10): micro 0,938 (= acurácia), ponderada 0,937, **macro 0,636**, a única que mostra a classe rara falhando.

Probabilidades 8.1

$$\text{LogLoss} = -\tfrac{1}{N}\textstyle\sum\big[y\log\hat{p} + (1-y)\log(1-\hat{p})\big]$$

$$\text{Brier} = \tfrac{1}{N}\textstyle\sum(\hat{p}-y)^2, \text{ limitado a } [0,1]$$

$$\text{ECE} = \textstyle\sum_m \frac{|B_m|}{N}\big|\text{acc}(B_m) - \text{conf}(B_m)\big|$$

- Um erro confiante: $-\log 0{,}01 = 4{,}6$ contra $0{,}92$ para quem previu 0,4.
- Redes costumam ser confiantes demais. O **temperature scaling** divide os logits por um único T ajustado na validação: AUC e acurácia não se mexem.

Armadilha: métrica e perda não são a mesma coisa: uma perda também precisa ser diferenciável.

Regressão 8.2

$$\text{MAE} = \tfrac{1}{N}\textstyle\sum|e_i| \quad \text{MSE} = \tfrac{1}{N}\textstyle\sum e_i^2 \quad \text{RMSE} = \sqrt{\text{MSE}} \;\geq\; \text{MAE}$$

$$R^2 = 1 - \frac{\sum(y_i-\hat{y}_i)^2}{\sum(y_i-\bar{y})^2} \qquad R^2_{aj} = 1-(1-R^2)\frac{N-1}{N-k-1}$$

- O MSE é minimizado pela **média**, o MAE pela **mediana**: a métrica que você otimiza é o modelo que você ganha. O MedAE ignora qualquer outlier isolado.
- RMSE/MAE $\approx 1{,}25$ para erros gaussianos; bem acima disso, poucos erros enormes. O alarme de fumaça mais barato.
- O R^2 compara com prever $\bar{y}$: fica negativo quando você faz pior (comum no teste) e não se compara entre conjuntos de dados.
- Huber: $\frac{1}{2}e^2$ para $|e| \leq \delta$, $\delta(|e| - \frac{1}{2}\delta)$ além disso. Perda quantílica (pinball): τe se $e \geq 0$, $(\tau-1)e$ se não; $\tau = 0{,}1;\ 0{,}5;\ 0{,}9$ dão um intervalo.

Armadilha: MAPE: indefinido em $y = 0$ e assimétrico, então empurra o modelo a prever baixo. Prefira MASE (< 1 ganha da previsão ingênua) ou WAPE $= \sum|e| / \sum|y|$.

Armadilha: sem linha de base. Reporte a média, a mediana ou, com tempo, o último valor ao lado do modelo.

Modelos generativos 8.3

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}\big(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\big)$$

$$\text{CLIPScore} = w\cdot\max\big(0, \cos(e_{\text{img}}, e_{\text{texto}})\big)$$

- BLEU (precisão de n-gramas, penalidade de brevidade, média geométrica) e ROUGE (revocação de n-gramas) comparam strings: uma paráfrase perfeita tira BLEU 0,00; inserir um "não" mantém o ROUGE-1 em 1,00.
- O BERTScore compara embeddings contextuais e importa os vieses do codificador.
- O FID ajusta **uma gaussiana** por nuvem: um modelo de um modo imitando uma realidade de dois modos tirou 0,009. Ele cai com o número de amostras e depende do extrator de features.
- Precisão e revocação de distribuições separam o número do FID: **perder modos** derruba a revocação (GANs), o **borrão** derruba a precisão (VAEs).
- FID para realismo, CLIP score para aderência ao prompt; o CLIP é fraco em composição. Depois, preferência humana aos pares.

Armadilha: reportar só um número. Olhe vinte amostras.

Modelos de linguagem 8.4

$$\text{PPL} = \exp\Big(-\tfrac{1}{N}\textstyle\sum_i \log p_\theta(w_i \mid w_{<i})\Big)$$

$$\text{pass@}k = \mathbb{E}_{\text{problemas}}\Big[1 - \tbinom{n-c}{k}\Big/\tbinom{n}{k}\Big], \; c \text{ de } n \text{ amostras passam}$$

$$P(A \text{ vence } B) = \frac{1}{1+10^{(R_B-R_A)/400}} \qquad R_A \leftarrow R_A + K\,(S_A - \mathbb{E}[S_A])$$

- Um chute uniforme sobre V tokens tem perplexidade V. Menor é melhor, mas só com o **mesmo tokenizador**; senão compare bits por byte.
- Benchmarks (MMLU, HumanEval, GSM8K…) medem tarefas; uma nota só é evidência se o benchmark for mais novo que o corte do treino (**contaminação**).
- Codex: 28,8% em pass@1 e 70,2% em pass@100. pass@k sem o seu k não diz nada.
- Juízes LLM concordam com pessoas em cerca de 80% das vezes. Vieses: posição (julgue nas duas ordens), verbosidade, autopreferência, notas amontoadas em 7–9 (prefira pares).
- RAG: recuperação (precisão e revocação do contexto) e geração (fidelidade, relevância da resposta), medidas em separado.

Armadilha: uma taxa de segurança sozinha. A recusa precisa da recusa excessiva ao lado: um modelo que recusa tudo tira nota perfeita.

Escolhendo pela pergunta 8.2–8.4

pergunta	use
erros simétricos, sem outliers absurdos	RMSE
uma cauda real que não se deve perseguir	MAE, MedAE
ganha de uma linha de base que não faz nada?	R^2, MASE
um intervalo, não um ponto	perda quantílica
alvo de cauda pesada, treinando uma rede	Huber / SmoothL1
tradução	chrF ou COMET; BLEU por continuidade
resumo	ROUGE mais uma checagem de fatos
texto para imagem	FID + CLIP score + pessoas
imagens incondicionais	FID + precisão/revocação, com o n
o corpus está sendo aprendido?	perplexidade, mesmo tokenizador
código que funciona	pass@1; pass@k só com verificador
a resposta A é melhor que a B?	juiz aos pares, nas duas ordens
os usuários preferem?	A/B com usuários, Elo da Arena

O que toda métrica é 8

- Uma **aproximação**: saiba no lugar de quê ela está e você sabe onde ela quebra.
- Uma **especificação**: minimizar o MSE pede a média condicional; o F1 diz que precisão e revocação valem o mesmo. Se você sabe a razão dos custos, use-a.
- Reportada **com a sua linha de base** e, para tudo que é amostrado, com a variação entre sementes ou folds.

Neurônios, backpropagation e otimização

Capítulos 1 e 3 a 6: de um neurônio ao otimizador que treina uma rede de milhões de pesos.

Onde fica 1, 3

- Paradigmas de IA: **simbólico** (regras e lógica), **conexionista** (redes neurais: aprendem com dados), **neuro-simbólico** (os dois).
- IA ⊃ aprendizado de máquina ⊃ redes neurais ⊃ deep learning. Supervisionado: entradas com rótulos. Não supervisionado: estrutura sem rótulos. Semi e autossupervisionado no meio.
- 1943 neurônio de McCulloch–Pitts; 1958 perceptron; 1986 backpropagation; 1989 aproximação universal (Cybenko: uma camada oculta aproxima qualquer função contínua num conjunto compacto).

Perceptron 4

$\hat{y} = H(w \cdot x + b)$, com $H(z) = 1$ se $z \geq 0$, senão 0

$w \leftarrow w + \eta\,(y - \hat{y})\,x \qquad b \leftarrow b + \eta\,(y - \hat{y})$

- Só atualiza quando erra. Uma passada pelos dados é uma **época**.
- Converge em um número finito de passos **se e somente se** os dados forem linearmente separáveis; senão oscila: limite as épocas.
- Sem probabilidades, só binário (um-contra-todos para mais classes), sensível a η.

▲ **Armadilha:** XOR. Nenhuma reta o separa, então nenhum perceptron o aprende: precisa de uma camada oculta.

Perceptron multicamadas 5

$h = \sigma(Wx + b^h) \qquad \hat{y} = \sigma(Vh + b^y)$

Sem a ativação, duas camadas viram uma: $V(Wx) = (VW)x$. É a não linearidade que compra a profundidade.

Ativações e suas derivadas

nome	$f(z)$	$f'(z)$
sigmoide	$1/(1+e^{-z})$, em $(0,1)$	$\sigma(1-\sigma) \leq 0{,}25$
tanh	$(e^{2z}-1)/(e^{2z}+1)$, em $(-1,1)$	$1-\tanh^2$
ReLU	$\max(0,z)$	1 se $z>0$, senão 0
Leaky ReLU	$\max(\alpha z, z)$	1 ou α
GELU	$\frac{1}{2}z(1+\tanh(0{,}798(z+0{,}0447z^3)))$	ReLU suave
SiLU / Swish	$z\,\sigma(z)$	usada na SwiGLU

Cada sigmoide que o gradiente atravessa o divide por pelo menos 4; a ReLU o passa inteiro onde a unidade está ativa (e uma ReLU morta não passa nada).

Perdas 5, 8, 15

regressão	MSE $\frac{1}{N}\sum(y-\hat{y})^2$; MAE; Huber
binária	BCE $-[y\log\hat{p} + (1-y)\log(1-\hat{p})]$ sobre uma sigmoide
K classes	entropia cruzada $-\log p_y$, $p = \text{softmax}(z)$; $\partial L/\partial z = p - y$

No PyTorch passe **logits**: `BCEWithLogitsLoss`, `CrossEntropyLoss` (elas aplicam a sigmoide ou a softmax sozinhas, de forma estável).

Backpropagation 5

A regra da cadeia, da perda de volta a cada peso ($u = Vh + b^y$, $z = Wx + b^h$):

$$\delta_y = \frac{\partial L}{\partial u} = \frac{\partial L}{\partial \hat{y}}\,\sigma'(u) \qquad \frac{\partial L}{\partial V} = \delta_y\,h^\top \qquad \frac{\partial L}{\partial b^y} = \delta_y$$

$$\delta_h = (V^\top \delta_y) \odot \sigma'(z) \qquad \frac{\partial L}{\partial W} = \delta_h\,x^\top \qquad \frac{\partial L}{\partial b^h} = \delta_h$$

$$p \leftarrow p - \eta\,\partial L/\partial p$$

Um passo, na mão (capítulo 5)

$W = \begin{bmatrix} 0{,}2 & 0{,}4 \\ 0{,}6 & 0{,}8 \end{bmatrix}$, $b^h = [0{,}1;\ 0{,}2]$, $V = [0{,}3;\ 0{,}5]$, $b^y = 0{,}4$, $x = [0{,}5;\ 0{,}8]$, $y = 0$, $\eta = 0{,}7$, MSE: $z = [0{,}52;\ 1{,}14]$, $h = [0{,}627;\ 0{,}758]$, $u = 0{,}967$, $\hat{y} = 0{,}725$, $L = 0{,}5249$; $\delta_y = 0{,}289$, $\delta_h = [0{,}020;\ 0{,}027]$; a primeira linha de W vira $[0{,}193;\ 0{,}389]$.

Gradiente descendente 6

$\theta_{t+1} = \theta_t - \eta\,\nabla J(\theta_t)$ (∇J aponta para cima)

- Numa quadrática de curvatura L só converge se $\eta < 2/L$: monótono abaixo de $1/L$, em zigue-zague entre $1/L$ e $2/L$, explode acima. Divergir é aritmética, não azar.
- Ravinas: $\kappa = L/\mu$ (número de condição) dá o custo, $O(\kappa \log 1/\varepsilon)$ passos; com $\kappa = 100$, 115 passos por ordem de grandeza.
- O inimigo é o plano (selas, platôs), não os mínimos locais ruins.

Quantos dados por passo

Batch: todos os N (exato, uma atualização por época). Estocástico: uma amostra. Mini-batch: B de 32 a 512. O ruído do gradiente cai como $\sigma/\sqrt{B}$, então o SGD termina numa **nuvem** em volta do mínimo, cujo raio cresce com $\sqrt{\eta}$ e encolhe com $\sqrt{B}$: reduza η no fim. Batch $\times k$: $\eta \times k$ (regra linear) ou $\times\sqrt{k}$.

Momentum 6

$V_{t+1} = \beta V_t + (1-\beta)\nabla J(\theta_t) \qquad \theta_{t+1} = \theta_t - \eta V_{t+1}$

- Atravessando a ravina o gradiente troca de sinal e se cancela; ao longo dela, se soma. $1/(1-\beta)$ é a memória: 10 passos com $\beta = 0{,}9$.
- Nesterov: o gradiente no ponto adiante $\theta_t - \eta\beta V_t$.

▲ **Armadilha:** o `SGD(momentum=0.9)` do PyTorch usa $v \leftarrow \beta v + g$, sem o $(1-\beta)$: a mesma direção, um passo $10\times$ maior.

Métodos adaptativos 6

AdaGrad: $G_t = G_{t-1} + g_t^2$, passo $\eta\,g_t/(\sqrt{G_t} + \epsilon)$

RMSProp: $V_{t+1} = \beta V_t + (1-\beta)g_t^2$, passo $\eta\,g_t/(\sqrt{V_{t+1}} + \epsilon)$

Adam: $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$, $v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$

$\hat{m}_t = m_t/(1-\beta_1^t)$, $\hat{v}_t = v_t/(1-\beta_2^t)$, $\theta \leftarrow \theta - \eta\,\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$

AdamW: $\theta \leftarrow \theta - \eta\left(\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon) + \lambda\theta\right)$

- A soma do AdaGrad nunca diminui, então o passo morre; o RMSProp usa uma média.
- Padrões: $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\epsilon = 10^{-8}$. Sem a correção de viés o primeiro passo seria $3{,}16\,\eta$.
- O AdamW decai os pesos diretamente; no Adam um termo L2 também é dividido por $\sqrt{\hat{v}}$. Não decaia vieses nem ganhos de normalização.

Agendamento e clipping 6

$\eta_t = \eta_{\max}\,s(t)$: **warmup** a partir de quase 0 nos primeiros passos, depois decaimento **cosseno** até 0. **Clipping**: $g \leftarrow g \cdot \min(1, c/\|g\|)$, com $c = 1$: a direção fica, o comprimento é limitado.

Na prática 6

situação	otimizador	comece com
Transformer do zero	AdamW	3e-4, $\beta = (0{,}9;\ 0{,}95)$, wd 0,1, warmup + cosseno
fine-tuning	AdamW	2e-5 a 5e-5, wd 0,01
CNN, treino longo	SGD + Nesterov	0,1 (batch 256), 0,9, wd 5e-4, cosseno
MLP tabular; primeira tentativa	Adam	1e-3

Varra η em 1e-1, 1e-2, 1e-3 e 1e-4 por algumas centenas de passos; fique com o maior que não explode e divida por dois.

sintoma	tente
perda `NaN`/`inf`	$\eta \div 10$, clipping, procure um $\log 0$
plana desde o passo 0	$\eta \times 10$; ela decora 10 amostras?
cai e estaciona alta	momentum, reduzir η, olhar a inicialização
treme muito	batch maior, η menor
diverge na época k	agendamento, warmup
treino bem, validação subindo	não é o otimizador: regularize (página 4)

O laço de treino 6

```
opt = torch.optim.AdamW(params, lr=3e-4, weight_decay=0.1)
for x, y in loader:
    opt.zero_grad()                # gradientes se acumulam
    loss = loss_fn(model(x), y)    # logits, não probabilidades
    loss.backward()
    nn.utils.clip_grad_norm_(params, 1.0)
    opt.step(); sched.step()
```

```
def adam(th, g, m, v, t, lr=1e-3, b1=.9, b2=.999, e=1e-8):
    m = b1 * m + (1 - b1) * g          # direção
    v = b2 * v + (1 - b2) * g ** 2     # escala
    mh, vh = m / (1 - b1 ** t), v / (1 - b2 ** t)
    return th - lr * mh / (np.sqrt(vh) + e), m, v
```

Regularização, profundidade e convoluções

Capítulos 7, 9 e 10: ajustar pior de propósito, manter o sinal vivo por muitas camadas e a primeira arquitetura que é uma prior.

Generalização 7

lacuna $= R(\theta) - \hat{R}(\theta)$: risco verdadeiro menos risco de treino

$$\mathbb{E}\left[(y-\hat{f})^2\right] = \underbrace{(\mathbb{E}\hat{f}-f)^2}_{\text{viés}^2} + \underbrace{\mathbb{E}\left[(\hat{f}-\mathbb{E}\hat{f})^2\right]}_{\text{variância}} + \underbrace{\sigma^2}_{\text{ruído}}$$

- **Underfitting**: erros de treino e validação altos e próximos. **Overfitting**: treino baixo, validação bem mais alta; a lacuna é o sintoma.
- Com a quantidade de dados fixa, pesos enormes de sinais alternados são a assinatura de quem decora ($\|w\|^2$: 6 no grau 3, 7×10^6 no grau 14).
- **Double descent**: passado o ponto de interpolação, o erro de teste pode voltar a cair. "Diminua o modelo" é um mau conselho para redes grandes.

L2 e L1 7

L2: $\tilde{J} = J + \frac{\lambda}{2}\sum w_i^2 \;\Rightarrow\; w \leftarrow (1-\eta\lambda)\,w - \eta\nabla J$ (weight decay)

L1: $\tilde{J} = J + \lambda \sum |w_i|$, gradiente λ sinal(w)

- A L2 empurra em proporção e nunca chega a zero (um círculo); a L1 empurra com força constante até exatamente zero (um losango, só quinas): seleção de features. As duas: elastic net.
- Penalize só as matrizes de pesos, não os vieses nem os ganhos de normalização. λ: 10^{-5} a 10^{-2} com SGD, 0,01–0,1 com AdamW.

▲ **Armadilha:** com Adam, um termo L2 não é weight decay. Use AdamW.

Dropout 7

$$m_j \sim \text{Bernoulli}(1-p), \qquad \tilde{h}_j = \frac{m_j}{1-p}\,h_j \quad \text{(invertido: inferência comum)}$$

- Quebra a co-adaptação; treina um ensemble de 2^n sub-redes que dividem os pesos.
- Camadas densas $p = 0{,}2$–$0{,}5$; Transformers $p = 0{,}1$; raramente em convoluções; nunca na saída.
- Coloque depois da BatchNorm, não antes: as duas discordam sobre a variância.

▲ **Armadilha:** esquecer o `model.eval()`: o dropout continua ligado, a BatchNorm continua atualizando e nada quebra.

Normalização 7, 9, 12

$$\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}, \qquad y = \gamma\hat{x} + \beta \quad (\epsilon \text{ dentro da raiz})$$

	BatchNorm	LayerNorm
estatísticas sobre	o batch, por feature	as features de uma amostra
treino	média e variância do batch	igual na inferência
inferência	médias móveis, congeladas	
regulariza?	sim, pelo ruído do batch	não
onde	CNNs	Transformers, qualquer batch

O backward da BatchNorm: mexer num x_i move μ e σ^2, então a saída de toda amostra se move; $\sum_i \partial L/\partial x_i = 0$. O ganho dela é sobretudo de otimização, não de "internal covariate shift".

Parar cedo e mais dados 7

- Avalie a cada época; **paciência**; **restaure o melhor checkpoint**. Num modelo linear equivale ao ridge com $\lambda \approx 1/(\eta t)$.
- Label smoothing: $y_k^{LS} = (1-\varepsilon)y_k + \varepsilon/K$, $\varepsilon = 0{,}1$.
- Mixup: $\tilde{x} = \lambda x_i + (1-\lambda)x_j$, $\tilde{y} = \lambda y_i + (1-\lambda)y_j$, $\lambda \sim \text{Beta}(\alpha,\alpha)$, $\alpha = 0{,}1$–$0{,}4$.
- Mais dados rotulados ganham de todas as técnicas daqui; data augmentation vem logo depois.

as duas curvas	querem dizer
treino ↓, validação ↑	overfitting: pare cedo, regularize, mais dados
as duas altas, próximas	underfitting: capacidade, mais treino, menos regularização
validação *melhor* que treino	dropout/BN ligados, ou um vazamento
validação perfeita	um vazamento, sempre

▲ **Armadilha:** regularizar um modelo que não consegue decorar 10 amostras. Isso é um bug, não variância.

Redes profundas: sinal e gradiente 9

$$\mathbb{E}[a_{l+1}^2] = \underbrace{n\,\sigma^2\,\rho}_{\text{fator}}\;\mathbb{E}[a_l^2]$$, que depois de L camadas vai à potência L

n = fan-in, σ^2 = variância dos pesos, ρ = a parte que a ativação deixa passar ($\frac{1}{2}$ na ReLU). Um fator de 0,95 é inofensivo em 5 camadas e vale 10^{-7} em 300.

ativação	inicialização	por quê
tanh, sigmoide	Xavier $\sigma^2 = 1/n$	ou $2/(n_{in}+n_{out})$
ReLU	He $\sigma^2 = 2/n$	zera metade; o 2 paga
fim de bloco residual	zero	começa como identidade

ReLU com 30 camadas: a ingênua $\mathcal{N}(0,1)$ cresce $10^{39}\times$ (`inf`); a Xavier encolhe a 10^{-9}; a He fica perto de 1. Dá para ver na inicialização, antes de qualquer passo.

residual: $a_{l+1} = a_l + f(a_l)$ $\qquad \frac{\partial a_{l+1}}{\partial a_l} = 1 + \frac{\partial f}{\partial a_l}$

O 1 é um piso, não um teto: um atalho sozinho faz sinal e gradiente explodirem, por isso vem sempre com uma normalização. Pós-LN Norm$(a+f(a))$ prende o sinal; pré-LN $a + f(\text{Norm}(a))$ mantém o caminho identidade limpo.

▲ **Armadilha:** a inicialização padrão do `nn.Linear` tem variância $1/(3n)$, um sexto da He: uma pilha funda de ReLU feita com os padrões apaga o sinal.

Convolução: a conta 10

$$H_{out} = \left\lfloor \frac{H_{in}+2p-k}{s} \right\rfloor + 1 \qquad \#\text{params} = C_{out}\,(C_{in}k^2+1)$$

- Padding "same": $p = (k-1)/2$ com k ímpar. Stride 2 corta a resolução pela metade.
- Não há H nem W na conta: `Conv2d(3, 64, 3)` tem 1 792 pesos; uma camada densa entre os mesmos tensores, ≈201 milhões.
- Uma convolução é uma camada densa com **localidade** e **compartilhamento de pesos** impostos: uma prior, não capacidade.

Campo receptivo

Stride 1: cada camada $k \times k$ soma $k-1$ (crescimento linear). Cada camada de stride 2 dobra o passo de todas as seguintes (geométrico). A dilatação alcança mais longe sem pesos. Duas 3×3 alcançam 5 pixels como uma 5×5, com 18 pesos contra 25 e uma não linearidade no meio (VGG).

Equivariância e invariância

O compartilhamento dá **equivariância** à translação. Um classificador quer **invariância**: o global average pooling e o data augmentation a compram. Rotação e escala só se compram com augmentation e dados.

Convolução: o backward 10

$\partial L/\partial b$	soma do gradiente de saída no batch e no espaço
$\partial L/\partial W$	correlação cruzada da entrada com o gradiente de saída
$\partial L/\partial X$	convolução *full* do gradiente de saída com o kernel (padding $k-1$ de cada lado)

O max pooling é um **roteador**: só o vencedor da janela recebe gradiente (27 de 36 células ficam sem nada em 2×2, stride 2); o average pooling dá $1/k^2$ a cada uma. Usado para a frente, o $\partial L/\partial X$ é a `ConvTranspose2d`.

▲ **Armadilha:** convolução transposta com um stride que não divide o kernel: artefatos de tabuleiro. Faça upsample e depois convolução.

Uma CNN inteira 10

camada	saída	params	CR
`Conv2d(1,16,3,p=1)`+BN+ReLU	$16 \times 28 \times 28$	192	3
`MaxPool2d(2)`	$16 \times 14 \times 14$	0	4
`Conv2d(16,32,3,p=1)`+BN+ReLU	$32 \times 14 \times 14$	4 704	8
`MaxPool2d(2)`	$32 \times 7 \times 7$	0	10
`Conv2d(32,64,3,p=1)`+BN+ReLU	$64 \times 7 \times 7$	18 624	18
`AdaptiveAvgPool2d(1)`+`Flatten`	64	0	tudo
`Linear(64,10)`	10	650	

A resolução desce, os canais sobem; os parâmetros moram nas camadas finais. Com `Flatten` no lugar do pooling global, só a cabeça custaria 31 370.

O que sobreviveu

Pilhas de 3×3, residuais, normalização, global average pooling, convoluções 1×1 e separáveis em profundidade (custo $\approx 1/C_{out} + 1/k^2$). Sumiram: cabeças densas enormes, LRN, stems grandes. ConvNeXt: boa parte da vantagem do ViT era a receita. As convoluções seguem nos autoencoders latentes, na segmentação médica (nnU-Net), na visão embarcada e no stem do ViT.

Regularização, tudo junto 7

mecanismo	técnica	onde vale
restringir os pesos	weight decay	quase sempre: 1e-4 (SGD), 0,01–0,1 (AdamW)
injetar ruído	dropout, BN, mixup	camadas densas, Transformers; visão e áudio
limitar a busca	parar cedo	de graça, sempre
suavizar o alvo	label smoothing	muitas classes, $\varepsilon = 0{,}1$
somar dados	mais rótulos	ganha de todas as linhas acima

Ordem: primeiro faça o modelo decorar, depois some regularização uma peça por vez, medindo cada uma.

Atenção, Transformers e Vision Transformers

Capítulos 11 a 13: "o gato subiu no" → " telhado", um passo por vez; depois, o mesmo bloco sobre patches de imagem.

A Do texto à próxima palavra 11, 12

1 tokens	texto → pedaços → ids inteiros
2 embeddings	um vetor por id, mais a sua posição
3 atenção	cada token relê os outros
4 máscara, cabeças	sem espiar o futuro; várias cabeças lado a lado
5 bloco	atenção + MLP + residuais, $n \times d \to n \times d$
6 saída	último vetor → um logit por token do vocabulário → softmax
7 geração	escolha um token, acrescente, repita

	brinquedo das aulas	GPT-2 small
vocabulário V	12	50 257
largura d; cabeças $h \times d_k$	4; 2×2	768; 12×64
blocos L; largura do MLP	1; 8	12; 3 072
pesos	256	124 milhões

Tokens 11

- **BPE**: comece pelos bytes, junte o par mais frequente, de novo e de novo; para tokenizar texto novo, refaça as junções na mesma ordem. GPT-2: 50 000 junções + 256 bytes + 1 token especial.
- O espaço pertence à palavra seguinte: `o` no começo e `␣o` depois de um espaço são tokens diferentes.
- Português custa mais: 1,92 token do GPT-2 por palavra contra 1,06 em inglês (mais latência, menos contexto, conta maior).

Embeddings e posição 11

$$y_t = E[i_t] = \text{onehot}(i_t)^\top E,\ E \in \mathbb{R}^{V \times d} \qquad x_t = E[i_t] + P_t \text{ (GPT-2)}$$

- A tabela do GPT-2: 38,6 milhões de números, 31% do modelo. O gradiente dela é esparso: só as linhas dos ids do batch se mexem, então tokens raros aprendem devagar.
- Atenção sem posição é equivariante a permutações: não distingue "cachorro morde homem" de "homem morde cachorro".

senoidal	$\sin/\cos$ fixos somados à entrada; histórica
aprendida	um vetor por posição (BERT, GPT-2); não passa do tamanho treinado
ALiBi	uma penalidade de distância nos escores
RoPE	gira q e k de um ângulo $\propto$ posição: $\langle R_m q, R_n k\rangle = \langle q, R_{n-m} k\rangle$, só a distância conta. Universal.

Atenção por produto escalar 11

$$\text{Attention}(Q,K,V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)V$$

$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$: três projeções do mesmo X

Um **dicionário suave**: a query é comparada com todas as chaves e a saída é a média dos valores ponderada. Uma camada densa cujos pesos são calculados a partir da entrada.

X	$n \times d$	um vetor por token
Q, K	$n \times d_k$	perguntas e rótulos
V	$n \times d_v$	conteúdos
$S = QK^\top/\sqrt{d_k}$	$n \times n$	todos os pares: o $O(n^2)$
$A = \text{softmax}(S)$	$n \times n$	cada linha soma 1
$O = AV$	$n \times d_v$	um vetor com contexto por token

Na mão: "banco" ao lado de "praça"

$q = [1;\ 2]$, chaves $[1;\ 0]$, $[1;\ 1]$, $[0;\ 2]$: escores 1, 3 e 4; $\div\sqrt{2}$: 0,71; 2,12; 2,83; softmax $[0{,}07;\ 0{,}31;\ 0{,}62]$; com os valores $[1;\ 0]$, $[0{,}5;\ 0{,}5]$, $[1;\ 0]$ a saída é $[0{,}85;\ 0{,}15]$: o banco de *sentar*.

Por que $\sqrt{d_k}$

$\text{Var}(q \cdot k) = d_k$ para componentes de variância 1. Com $d_k = 128$ os escores diferem por $\sim$20, a softmax vira um argmax duro e o gradiente some. Dividir por d_k achata tudo numa média simples.

▲ **Armadilha:** a entropia da atenção desabando cedo parece bug de treino. Registre-a.

Máscaras e cabeças 11

$M_{ij} = 0$ se $j \le i$, $-\infty$ se $j > i$, somada a S antes da softmax

$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O$

- A máscara causal dá n exemplos de treino por passada. Outras: padding, janela deslizante, fronteira entre documentos empacotados.
- h cabeças de largura d/h custam o mesmo que uma de largura d: paralelismo de graça. Parâmetros: W_Q, W_K, W_V, W^O, quatro matrizes $d \times d$, qualquer que seja h.
- Formas: $B \times n \times d \to B \times h \times n \times d/h$; escores $B \times h \times n \times n$.

O cache KV e o $O(n^2)$ 11

bytes do cache $= 2 \times L \times h_{kv} \times d_{head} \times n \times B \times$ bytes por número

- Llama 3 8B ($L = 32$, $h_{kv} = 8$, $d_{head} = 128$, 16 bits): 128 KiB por token, 16 GiB com 128 mil tokens, tanto quanto os pesos.
- h_{kv} é o termo livre: MHA (h), MQA (1, pior), **GQA** (8, o padrão), MLA (guarda um latente de posto baixo).
- Decodificar lê o cache inteiro a cada token: limitado por banda, não por FLOPs.
- FlashAttention: em blocos, sem guardar a matriz $n \times n$, exata. Janela deslizante, atenção linear e SSMs trocam recuperação por custo; os híbridos vencem.

O que veio antes

A RNN multiplica o gradiente por uma matriz a cada passo: em T passos, um produto de T fatores ($0{,}9^{64} \approx 10^{-3}$). A LSTM freia o decaimento; a atenção alcança qualquer token num passo só e em paralelo.

O bloco Transformer 12

$h = x + \text{Attn}(\text{Norm}(x)) \qquad y = h + \text{FFN}(\text{Norm}(h))$

pilha pré-LN: $x_L = x_0 + \sum_\ell f_\ell(\cdot) \Rightarrow \partial x_L/\partial x_0 = I + \ldots$

- A atenção **conversa** (a única troca entre posições); o MLP **pensa** (cada token sozinho, $d \to 4d \to d$). Os residuais somam correções a um único **fluxo residual**.
- O MLP tem de dois terços a três quartos dos parâmetros de um bloco.

peça	2017	2026
lugar da norma	pós-LN (pede warmup)	**pré-LN**
normalizador	LayerNorm	RMSNorm $x/\sqrt{\frac{1}{d}\sum x_i^2 + \epsilon} \odot g$
MLP	ReLU, $4d$, 2 matrizes	SwiGLU $(\text{Swish}(xW_1) \odot xW_3)W_2$, $\frac{8}{3}d$
atenção	MHA	GQA
posição	senoidal, na entrada	RoPE, em q e k
viés	em toda parte	nenhum

Saída e decodificação 12

logits $z = f\, W_U$ (weight tying: $W_U = E^\top$) $\qquad p_j = \dfrac{e^{z_j/T}}{\sum_k e^{z_k/T}}$

perda: $\mathcal{L} = -\frac{1}{n}\sum_t \ln p(\text{próximo token verdadeiro}_t)$, a entropia cruzada

- Brinquedo: perda média de 0,979 nat contra $\ln 12 = 2{,}485$ de um chute uniforme. Somar 10 aos logits não muda nada.
- A temperatura nunca muda o argmax: logits $[2;\ 1;\ 0]$ dão $[0{,}67;\ 0{,}24;\ 0{,}09]$ com $T = 1$ e $[0{,}87;\ 0{,}12;\ 0{,}02]$ com $T = 0{,}5$. $T \to 0$ é o guloso.
- Top-k fica com k tokens; top-p com o menor conjunto que soma p; min-p com os tokens acima de uma fração do primeiro.

Famílias e escala 12

- Só encoder (BERT, bidirecional): embeddings, busca, rerankers. Encoder–decoder (T5): modalidades diferentes. **Só decoder venceu**: todo token é sinal de treino, um objetivo cobre todas as tarefas e o cache KV funciona.
- Treino: $C \approx 6ND$ (N parâmetros, D tokens). Chinchilla: uns 20 tokens por parâmetro. Modelos em produção treinam bem além (Llama 3 8B: uns 1 900), porque servir custa em proporção a N.

Vision Transformer 13

$x \in \mathbb{R}^{H \times W \times C} \to N = HW/P^2$ patches de P^2C números $\xrightarrow{E} \mathbb{R}^{N \times d}$

custo por imagem: $12LNd^2 + 2LN^2d$ multiplicações-somas

- 224 px, $P = 16$: 196 tokens (+`[CLS]`). ViT-B/16: $d = 768$, 12 blocos, 12 cabeças, ≈86,6 mi de parâmetros. O patch embedding *é* uma `Conv2d(C, d, P, stride=P)`.
- Cortar P pela metade quadruplica N; a atenção só domina com $N > 6d$ (4 608 tokens no ViT-B).
- A prior convolucional é piso em pequena escala e teto em grande: só ImageNet-1k, o ViT perde; 14 mi de imagens, empata; 300 mi, ganha. O DeiT mostrou que boa parte disso era a receita.
- Hoje: pooling no lugar do `[CLS]`, RoPE 2-D, register tokens. O ViT venceu porque emite **tokens**: é o codificador de imagem dos VLMs, do CLIP/SigLIP, do DINOv2 e a base do DiT.

Adaptando modelos e grandes modelos de linguagem

Capítulos 14 e 15: se vale adaptar um modelo pré-treinado, como fazer isso barato e o que transforma um continuador de texto num assistente.

Antes do fine-tuning: não faça 14

tente antes	resolve	não resolve
um prompt melhor	formato, tom, enquadramento	conhecimento que falta
recuperação (RAG)	fatos que faltam ou mudam, citações	comportamento, raciocínio
ferramentas, function calls	contas, dados ao vivo, ações	o que não tem API
um modelo maior	quase tudo, já	o seu custo e latência
fine-tuning	comportamento constante e barato; um modelo pequeno na *sua* tarefa	fatos que mudam

▲ **Armadilha:** fine-tuning para ensinar fatos. Eles se espalham pelos pesos, sem citação nem atualização. Faça fine-tuning para **comportamento**, recupere para **conhecimento**.

Quanto treinar 14

regime	treina	DistilBERT
sonda linear	só a cabeça, sobre features em cache	1 538
parcial	a cabeça e os últimos blocos	7,1 mi/bloco
completo	tudo, com η pequeno	66,4 mi

O fine-tuning completo é caro por causa do **estado do otimizador**: o AdamW guarda uma cópia em fp32 e dois momentos, 12 bytes por parâmetro treinável (seis vezes os pesos em bf16); com os gradientes, uns 16.

LoRA e QLoRA 14

$$W = W_0 + \frac{\alpha}{r} BA, \qquad B \in \mathbb{R}^{d\times r},\ A \in \mathbb{R}^{r\times k},\ r \ll \min(d,k)$$

treináveis: $r(d+k)$ no lugar de dk $\quad(4096\times 4096,\ r=8\text{: }256\times$ menos)

- A hipótese: a mudança que a tarefa pede tem posto baixo. W_0 congelada; A aleatória e $B=0$, então o treino começa do modelo pré-treinado ($\partial L/\partial A = 0$ no primeiro passo). As duas em zero nunca sairiam do lugar.
- Aplique em **todas** as camadas lineares (o MLP também), $r=16$, $\alpha=32$, $\eta \approx$ 1e-4 (10× o do fine-tuning completo). Funda para o deploy: a inferência não fica mais cara.
- **QLoRA**: a base congelada em NF4 de 4 bits; o adaptador treina em volta do erro de quantização. **DoRA**: magnitude e direção. Adapters, prefix tuning, BitFit: históricos.

▲ **Armadilha:** ensinar uma língua nova com LoRA: essa mudança não tem posto baixo e nenhum r acessível resolve.

Pós-treino 14, 15

SFT em demonstrações (formato) → otimização de preferências (qual resposta é melhor) → RL com verificador (acertar).

$$\mathcal{L}_{\text{DPO}} = -\log\sigma\Big(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\Big)$$

GRPO: $\hat{A}_i = \dfrac{r_i - \text{média}(r_{1..G})}{\text{desvio}(r_{1..G})}$ sobre G tentativas no mesmo prompt

- RLHF: um modelo de recompensa treinado em comparações humanas, depois PPO. O DPO treina direto nas comparações: dois modelos em vez de quatro, sem laço de amostragem.
- O GRPO dispensa a rede de valor do PPO; com um verificador (testes unitários, resposta conferível) ele produziu os modelos de raciocínio (DeepSeek-R1).

▲ **Armadilha:** esquecimento catastrófico: mantenha η baixo, misture uns 10% de dados gerais e avalie em tarefas que você não está treinando.

Quando usar o quê 14

formato ou tom errados	prompt
fatos que faltam ou mudam	RAG
qualidade de modelo grande a 1% do custo	destilar: SFT de um pequeno nas saídas dele
1 mil a 100 mil exemplos, uma GPU	LoRA, todas as lineares, $r=16$
memória curta	QLoRA
muitos clientes, uma base	adaptadores LoRA sem fundir
"prefira esta resposta"	SFT → DPO
respostas que um programa confere	SFT → GRPO

Pré-treino 15

$$\mathcal{L} = -\sum_{t=1}^{T}\log p_\theta(x_t \mid x_{<t}) \quad \text{(nats; 1 nat} \approx \text{1,44 bit)}$$

perplexidade $= e^{\mathcal{L}/T}$

- Autossupervisionado: os rótulos são os próximos tokens, então qualquer texto é dado de treino. Comprimir bem um texto exige modelar o que o produziu.
- Uma perda de 3 nats por token é uma perplexidade de $e^3 \approx 20$. Pythia: $L \propto N^{-0{,}10}$, uma reta em escala log–log.

sintoma	causa na tokenização
os r de "strawberry"	o modelo nunca vê letras
contas frágeis	`1234` vira `12|34`, `12345` vira `123|45`
português custa mais	junções ajustadas em texto quase todo em inglês
espaço no fim estraga	`"Paris"` e `"Paris"` são tokens diferentes

▲ **Armadilha:** esperar respostas de um modelo base: ele continua o texto e pode responder com mais três perguntas de prova.

Computação no teste 15

pass@$k = 1-(1-p)^k$ num problema resolvido com probabilidade p

melhor de n com verificador, com n grande: $\dfrac{p\,\text{TPR}}{p\,\text{TPR} + (1-p)\,\text{FPR}}$

- pass@k é um teto; o voto da maioria estaciona; um verificador converte a diferença até onde os falsos positivos dele deixam ($p=0{,}25$, TPR 0,80, FPR 0,35: 0,43 para qualquer n; FPR 0: 1).
- Por isso o raciocínio melhorou primeiro em código e matemática: verificadores baratos, quase sem falsos positivos.

▲ **Armadilha:** reward hacking: otimizar contra um verificador otimiza o verificador, não o acerto.

Mistura de especialistas 15

$$\text{MoE}(x) = \sum_{i\in\text{Top-}k(g(x))} g(x)_i\, E_i(x), \qquad g(x) = \text{softmax}(W_g x)$$

Memória para os parâmetros **totais**, computação para os **ativos** (DeepSeek-V3: 671 bi no total, 37 bi ativos). O difícil é balancear a carga, não rotear.

Decodificar e servir 15

- Padrões: fatos, código e chamadas de ferramenta com $T = 0$–$0{,}2$; prosa com $0{,}7$–$1{,}0$ e top-p 0,9 ou min-p 0,05.
- O **prefill** lê o prompt em paralelo (limitado por computação); o **decode** faz um token por passada (limitado por memória). O cache KV troca $O(n^2)$ passadas de token por $O(n)$.
- Quantização (8 ou 4 bits) lê menos bytes por token. Decodificação especulativa: um rascunho propõe, o modelo grande aceita cada token com probabilidade $\min(1, p/q)$: mesma distribuição, mais rápido.

De onde vêm as falhas 15

- A "emergência" muitas vezes é uma métrica de limiar; a perda melhora suave.
- A alucinação é treinada: nota só por acerto premia o chute, não o "não sei".
- Prompt injection não tem correção robusta quando o modelo tem ferramentas.
- Agentes são laços e os erros se acumulam: um passo 95% confiável vira 60% em dez.

Do chat aos agentes 15

Um chat devolve texto; um **agente** roda um laço: ler o objetivo, chamar uma ferramenta, ler o retorno, decidir, parar. O trabalho está nas ferramentas, nos erros, no histórico e em quando parar. A janela de contexto é um orçamento: cada retorno disputa espaço com o objetivo. O MCP padroniza o servidor de ferramentas.

O cenário e a escolha 15

fronteira, fechados	GPT, Claude, Gemini: raciocínio, multimodais, ferramentas; alugados
pesos abertos	DeepSeek, Qwen, Kimi, GLM, Llama, Mistral: meses atrás, bem mais baratos
pequenos, abertos	de 1 a 30 bilhões, uma GPU ou um celular: alvo de destilação
especialistas	código, embeddings, rerankers, segurança, OCR

Os preços caem uma ordem de grandeza por ano. Não escolha por leaderboard: escreva 20 a 50 exemplos da *sua* tarefa, rode três ou quatro modelos, leia as falhas.

LoRA em código 14

```
from peft import LoraConfig, get_peft_model
cfg = LoraConfig(r=16, lora_alpha=32,
                 target_modules="all-linear")
model = get_peft_model(model, cfg)   # r(d+k) por matriz
merged = model.merge_and_unload()    # W0 + (alpha/r) BA
```

Modelos generativos: o mapa, VAEs, GANs e CLIP

Capítulos 16 a 19: transformar uma distribuição de que se sabe amostrar numa de que não se sabe e aprender o que uma frase quer dizer para uma imagem.

Quatro jeitos de cortar o problema — 16

família	decomposição	amostragem	verossim.
autorregressiva	$p(x)=\prod_i p(x_i\|x_{<i})$	uma passada por dimensão	exata
latente (VAE)	$p(x)=\int p(x\|z)p(z)\,dz$	uma passada	um limite
adversarial (GAN)	nenhuma: gerador e crítico	uma passada	nenhuma
difusão, fluxo	um caminho no tempo, do ruído aos dados	10 a 100+ passadas	pela EDO

$\mathrm{KL}(p_{\text{dados}}\|p_\theta)$ direta: cobre todos os modos, borra (máxima verossimilhança, VAE)
$\mathrm{KL}(p_\theta\|p_{\text{dados}})$ reversa: escolhe um modo, larga o resto (estilo GAN)

Borrão e colapso de modos são as duas pontas de uma única assimetria. O trilema: qualidade, cobertura, velocidade; a velocidade é a única que dá para comprar depois (destilação).

Do autoencoder ao VAE — 17

- Um autoencoder $o=D(E(x))$, $\mathcal{L}=\|x-o\|^2$, comprime; não gera: os códigos são um conjunto de pontos com buracos, não uma distribuição.
- Um VAE codifica uma **distribuição** por entrada: $q_\phi(z|x)=\mathcal{N}(\mu_\phi(x),\operatorname{diag}\sigma^2_\phi(x))$, puxada para a prior $p(z)=\mathcal{N}(0,I)$.

$$\log p_\theta(x)=\text{ELBO}+\mathrm{KL}\big(q_\phi(z|x)\,\|\,p(z|x)\big)\ \geq\ \text{ELBO}$$
$$\text{ELBO}=\mathbb{E}_q\big[\log p_\theta(x|z)\big]-\mathrm{KL}\big(q_\phi(z|x)\,\|\,p(z)\big)$$
$$\mathcal{L}=\|x-\hat{x}\|^2+\beta\cdot\tfrac{1}{2}\textstyle\sum_i\left(\mu_i^2+\sigma_i^2-1-\log\sigma_i^2\right)$$
reparametrização: $z=\mu+\sigma\odot\epsilon,\quad \epsilon\sim\mathcal{N}(0,I)$
$\partial z/\partial\mu=1,\qquad \partial z/\partial\log\sigma^2=\tfrac{1}{2}\,\sigma\epsilon$: o ruído entra como dado

- A folga é exatamente o quanto o chute do encoder sobre a posterior erra; um objetivo treina as duas redes.
- Na mão: $p(z)=\mathcal{N}(0,1)$, $p(x|z)=\mathcal{N}(2z,1)$, $x=3$: $\log p(x)=-2{,}6237$, posterior $\mathcal{N}(1{,}2;\ 0{,}2)$. O melhor ELBO é o da posterior, não o da melhor reconstrução.
- O encoder devolve $s=\log\sigma^2$: $\sigma=e^{s/2}$.
- $\mathrm{KL}(\mathcal{N}(\mu_q,\sigma_q^2)\|\mathcal{N}(\mu_p,\sigma_p^2))=\log\frac{\sigma_p}{\sigma_q}+\frac{\sigma_q^2+(\mu_q-\mu_p)^2}{2\sigma_p^2}-\frac{1}{2}$; nunca negativa, não simétrica.

O β é a história inteira

$\beta\to 0$: um autoencoder; amostras da prior não decodificam em nada. β grande demais: **colapso da posterior**, KL ≈ 0 e o código ignora a entrada. Remédios: KL annealing, free bits.

⚠ **Armadilha:** olhar só a perda total. Registre reconstrução e KL em separado: KL perto de 0 com reconstrução perto da variância dos dados é colapso.

Para que serve um VAE

O borrão é o objetivo (a KL direta; o ℓ_2 devolve a média condicional). Os autoencoders de produção somam um termo perceptual (LPIPS) e um adversarial, usam $\beta\approx 10^{-6}$ e ficam congelados sob um modelo de difusão: latente do SD 1.5 $64\times64\times4$ (48× menos números), SD3/FLUX $64\times64\times16$ (12×). **VQ-VAE**: $z_q=\text{codebook}[\arg\min_k\|z_e-e_k\|]$, inteiros para modelos autorregressivos.

GANs — 18

$$\min_G\max_D\ \mathbb{E}_x\big[\log D(x)\big]+\mathbb{E}_z\big[\log(1-D(G(z)))\big]$$
$$D^*(x)=\frac{p_{\text{dados}}(x)}{p_{\text{dados}}(x)+p_G(x)}$$
$$V(D^*,G)=-\log 4+2\,\mathrm{JS}(p_{\text{dados}}\|p_G)$$

- Uma perda que não se escreve, aprendida: o discriminador estima a razão entre densidades.
- O gerador usa a perda **não saturante** $-\log D(G(z))$: com $D(G(z))=0{,}01$ o gradiente dela é $-0{,}99$; o da minimax, $-0{,}01$.
- $z\in\mathbb{R}^k$; k é a dimensão latente, um tamanho que você escolhe.

colapso de modos	poucas saídas, qualquer que seja z: nada premia a diversidade
gradiente que some	o D ganha com folga demais
oscilação	os jogadores andam em ciclo; um jogo não precisa convergir
D decorando	ele memoriza o conjunto de treino

O que ficou: WGAN-GP, normalização espectral, R1, ADA, todos fazendo uma coisa só: **impedir que o discriminador fique afiado demais**.

⚠ **Armadilha:** ler as perdas da GAN: elas ficam em $2\log 2\approx 1{,}39$ e $\log 2\approx 0{,}69$ funcionando ou não. Olhe amostras e uma métrica.

A arquitetura perdeu (a difusão tem uma perda de regressão que desce); a **perda adversarial** segue nos decoders dos autoencoders latentes, na super-resolução, nos vocoders e na destilação para 1 a 4 passos: onde o alvo é uma de muitas respostas válidas.

CLIP — 19

Dois codificadores (um ViT e um Transformer sobre até 77 tokens), cada um com uma projeção linear e normalização L2, num mesmo espaço de d dimensões.

$$\mathcal{L}=\tfrac{1}{2}\Big(\mathrm{CE}\big(\tfrac{IT^\top}{\tau},\text{diag}\big)+\mathrm{CE}\big(\tfrac{TI^\top}{\tau},\text{diag}\big)\Big)\qquad\text{(InfoNCE simétrica)}$$
SigLIP: $\frac{1}{N}\sum_{i,j}\log\left(1+e^{-y_{ij}(t\langle I_i,T_j\rangle+b)}\right)$, $y_{ij}=\pm1$, b começa em -10

- A linha i de $IT^\top/\tau$ é um problema de N classes: qual legenda é a da imagem i? Toda outra legenda do batch é um negativo (32 767 deles com $N=32\,768$); um chute cego paga $\log N$.
- O τ é aprendido: começa em 0,07, preso em $1/\tau\leq 100$, termina em 0,01. A legenda certa e a errada diferem por centésimos de cosseno (0,26 contra 0,20): só $\times100$ deixa a softmax confiante. O τ muda a perda, não o ranking.
- Zero-shot: logits $=Wx/\tau$ com linhas w_c = prompts codificados, um classificador linear escrito em frases; "zero-shot" é busca em 400 milhões de pares de treino.
- Vetores unitários: $\|a-b\|^2=2-2\cos(a,b)$. Cossenos aleatórios em d dimensões têm desvio $\approx 1/\sqrt{d}$ (0,044 em 512).

⚠ **Armadilha:** o CLIP é quase um saco de palavras: contagem, relações espaciais e composição são fracas; e há o modality gap (imagens e textos em cones separados).

Difusão — 20

$$q(x_t|x_{t-1})=\mathcal{N}\big(\sqrt{1-\beta_t}\,x_{t-1},\ \beta_t I\big),\ T=1000$$
$$x_t=\sqrt{\bar\alpha_t}\,x_0+\sqrt{1-\bar\alpha_t}\,\epsilon,\qquad \bar\alpha_t=\textstyle\prod_{s\leq t}(1-\beta_s)$$
$$\mathcal{L}=\mathbb{E}_{x_0,\epsilon,t}\big\|\epsilon-\epsilon_\theta(x_t,t,c)\big\|^2$$
$$\hat{x}_0=\big(x_t-\sqrt{1-\bar\alpha_t}\,\epsilon_\theta\big)/\sqrt{\bar\alpha_t}\qquad \text{score}\approx-\epsilon_\theta/\sqrt{1-\bar\alpha_t}$$

- Qualquer nível de ruído em forma fechada: sorteie t, pule para lá, faça regressão do ruído que você mesmo sorteou. Regressão supervisionada comum, estável, cobre os modos.
- Predição de ϵ, de x_0 e de v ($v=\sqrt{\bar\alpha_t}\epsilon-\sqrt{1-\bar\alpha_t}x_0$) são intercambiáveis; elas pesam os níveis de ruído de outro jeito.
- O agendamento decide o que se aprende: o $\bar\alpha_T\approx 0{,}0047\neq 0$ do SD 1.x é por que ele não faz uma imagem preta de verdade.

Stable Diffusion — 20

Difusão + um latente de VAE (48× menor) + um codificador de texto do CLIP + classifier-free guidance.

etapa	forma	params
tokenizador, texto do CLIP	77×768	123 mi
ruído inicial z_T (a semente)	$4\times64\times64$	
U-Net, duas vezes por passo	$2\times4\times64\times64$	860 mi
decoder do VAE, uma vez	$3\times512\times512$	50 mi

O prompt entra por **cross-attention**: queries do latente da imagem, chaves e valores do texto.

CFG: $\tilde\epsilon=\epsilon_\theta(x_t,\varnothing)+w\,\big(\epsilon_\theta(x_t,c)-\epsilon_\theta(x_t,\varnothing)\big)$

- Treine com a legenda descartada 10% das vezes; $w=0$ incondicional, $w=1$ condicional, $w>1$ extrapola. Dobra o custo e troca diversidade por aderência; demais, sai da variedade dos dados.
- Os amostradores são separados do modelo: DDPM 1000 passos; DDIM determinístico e inversível; DPM-Solver++ 15–25. Abaixo de uns 10 passos é preciso destilar (LCM, SDXL-Turbo com discriminador: 1–4).
- ControlNet, IP-Adapter e LoRA congelam a base e somam um caminho pequeno ao lado, iniciado sem efeito.
- Depois do SD 1.5: uma base Transformer (MMDiT), texto com T5 + CLIP, latentes de 16 canais, rectified flow.

⚠ **Armadilha:** a semente fixa só o ruído inicial; com um amostrador estocástico a mesma semente ainda muda a imagem.

Modelos generativos: flow matching, diffusion Transformers e tokens

Capítulos 21 a 23: caminhos mais retos, uma base que escala e imagens como sequências.

Flow matching 21

$$\frac{dx}{dt} = v_\theta(x,t), \quad x_0 \sim \mathcal{N}(0,I) \;\Rightarrow\; x_1 \sim p_{\text{dados}}$$
$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,x_0,x_1}\left\|v_\theta(x_t,t) - (x_1 - x_0)\right\|^2, \qquad x_t = (1-t)x_0 + t x_1$$

- O campo marginal $u_t(x) = \mathbb{E}[x_1 - x_0 \mid x_t = x]$ é intratável; fazer regressão do alvo **condicional** $x_1 - x_0$ tem o mesmo gradiente. O mesmo lance do DDPM, que faz regressão de ϵ no lugar do score.
- Por isso a perda tem um piso acima de zero e os caminhos aprendidos não se cruzam.
- Aqui $t = 0$ é ruído e $t = 1$ é dado: o contrário do capítulo 20.

método	$a(t)$	$b(t)$	caminho
DDPM	$\sqrt{\bar\alpha_t}$	$\sqrt{1-\bar\alpha_t}$	curvo, quarto de círculo
flow matching	t	$1-t$	reto por par
rectified flow	t	$1-t$	reto, re-pareado

Com $x_t = a(t)x_1 + b(t)x_0$: difusão é flow matching com um interpolante curvo. O caminho condicional é reto; o **marginal**, que o modelo aprende, não.

- A vantagem real é o objetivo: uma regressão, sem agendamento, sem escolher entre ϵ e v.
- **Reflow**: integre a EDO treinada e retreine nos próprios pares (z, x); caminhos mais retos, um passo de Euler chega mais perto.
- Um passo: aprender saltos em vez de velocidades (consistency, shortcut, MeanFlow) ou destilar (FLUX.1 schnell).
- O SD3 sorteia t de uma logit-normal (o meio); desloque t com a resolução.
- Usado em imagem, vídeo, fala e ações de robô (π_0: 10 passos de Euler por trecho de 50 ações).

▲ **Armadilha:** três convenções mudam de um código para outro: se $t = 0$ é ruído ou dado; se o modelo prevê velocidade, x_1 ou ϵ; se t vai em $[0,1]$ ou é um índice inteiro.

Diffusion Transformers 22

latente $C \times H \times W$ em $N = \frac{H}{p} \cdot \frac{W}{p}$ patches de p^2C números $\to d$

$$c = \text{MLP}(\text{emb}(t)) + \text{emb}(y); \quad (\beta_1,\gamma_1,\alpha_1,\beta_2,\gamma_2,\alpha_2) = W\,\text{SiLU}(c) + b$$
$$h = x + \alpha_1 \odot \text{Attn}\big((1+\gamma_1)\odot \text{LN}(x) + \beta_1\big)$$
$$x' = h + \alpha_2 \odot \text{MLP}\big((1+\gamma_2)\odot \text{LN}(h) + \beta_2\big)$$

- Uma resolução só do começo ao fim; todo bloco vê todo token. DiT-XL/2 em 256 px: latente $4 \times 32 \times 32$, $p = 2$, 256 tokens.
- **adaLN-Zero**: $W = 0$ e $b = 0$ no começo, então todo portão $\alpha = 0$ e cada bloco é a identidade (como o $B = 0$ da LoRA). Venceu o condicionamento in-context e por cross-attention com o menor custo; $6d^2$ parâmetros por bloco.
- O tamanho do patch é o botão de computação: cortar p pela metade quadruplica N com quase os mesmos parâmetros; o FID melhora.
- O texto entra por cross-attention (só leitura), atenção conjunta com dois fluxos (MMDiT: SD3) ou um fluxo só (os blocos finais do FLUX).
- RoPE por eixo (linha, coluna, tempo): a resolução vira só o comprimento da sequência.
- Custo por bloco $12Nd^2 + 2N^2d$; a parte da atenção é $N/(N+6d)$: 4% no DiT-XL/2, 20% no FLUX em 1024^2, ≈87% num vídeo 720p. Imagens são limitadas pelo MLP, vídeo pela atenção.
- A contribuição é uma **curva de escala**: o FID cai de forma estável com os Gflops.

Imagens como tokens 23

$$\mathcal{L}_{VQ} = \|x - D(z_q)\|^2 + \|\text{sg}[z_e] - e\|^2 + \beta\|z_e - \text{sg}[e]\|^2$$
$$z_q = z_e + \text{sg}[e - z_e] \quad \text{(sg: stop-gradient)}$$
$$p(t_1,\ldots,t_N) = \textstyle\prod_i p(t_i \mid t_{<i}, c)\text{; CFG nos logits: } \ell = \ell_\varnothing + w(\ell_c - \ell_\varnothing)$$

- VQ-GAN: 256×256, redução 16 → 256 tokens; com $K = 16\,384$ códigos, no máximo $256 \times 14 = 3\,584$ bits.
- O último termo de z_q é o **straight-through estimator**. Só o código escolhido recebe gradiente, então códigos morrem (colapso do codebook). O FSQ arredonda poucas dimensões limitadas (níveis $8 \cdot 6 \cdot 5 = 240$ códigos, sem perda extra); o LFQ usa sinais (2^{18}).
- Custo: uma passada sequencial por token (256 numa grade 16×16). MaskGIT: todos os tokens mascarados de uma vez, ficam os confiantes, agenda cosseno, 8 a 12 rodadas. VAR: a próxima *escala*, 10 passadas. MAR: tokens contínuos com uma pequena cabeça de difusão.
- Os eixos de verdade: a ordem de geração (raster, rodadas paralelas, do grosso ao fino, tudo de uma vez) e a distribuição por token (categórica ou contínua).
- Verossimilhança exata, mas dos tokens. Voltou pela **unificação**: uma sequência e uma perda para texto e imagens (Chameleon, Emu3, Janus-Pro; Transfusion e BAGEL com uma perda de difusão dentro).

Qual gerador 16–23

texto para imagem, qualidade antes	difusão / flow, DiT
texto e imagens intercalados	autorregressivo ou híbrido
amostrar rápido, tokens discretos	MaskGIT, VAR
verossimilhança exata, compressão	autorregressivo
fala e áudio	autorregressivo sobre tokens de codec
um passo na inferência	GAN ou difusão destilada
um compressor sob um modelo de difusão	VAE (congelado)

Uma história de gargalos 16

quando	o quê	o gargalo que removeu
2013	VAE	retropropagar por um nó aleatório
2014	GAN	precisar de uma verossimilhança tratável
2015–20	difusão, DDPM	qualidade sem adversário
2021	CLIP, difusão latente	condicionar em linguagem; um espaço 48× menor
2022–23	SD, DALL·E 2	acesso: pesos abertos
2023–24	flow matching, DiT	caminhos mais retos, uma base que escala, de 1000 para <10 passos
2024–26	vídeo, any-to-any	coerência no tempo; um modelo para todas as modalidades

▲ **Armadilha:** avaliação e procedência seguem em aberto: o FID é uma aproximação fraca e as imagens de treino são trabalho de outras pessoas.

Flow matching em código 21, 22

```
# treino: t = 0 é ruído, t = 1 é dado
x1 = batch                          # dados
x0 = torch.randn_like(x1)           # ruído
t = torch.rand(len(x1), 1, 1, 1)
xt = (1 - t) * x0 + t * x1
y = torch.where(torch.rand(len(y)) < 0.1, null_id, y)  # CFG
loss = ((model(xt, t, y) - (x1 - x0)) ** 2).mean()

# amostragem: passos de Euler pelo campo aprendido
x = torch.randn(n, C, H, W)
for i in range(steps):
    t = torch.full((n, 1, 1, 1), i / steps)
    x = x + model(x, t, y) / steps
```

Guiando um modelo de difusão 20

img2img	codifique uma imagem, ponha ruído até um t (a força), remova a partir daí
inpainting	a cada passo, devolva o original com o ruído certo fora da máscara
ControlNet	uma cópia treinável do encoder, ligada por convoluções zeradas: pose, profundidade, bordas
IP-Adapter	cross-attention extra sobre embeddings de imagem do CLIP: estilo ou identidade
LoRA	um estilo, um personagem, um conceito, nas camadas de atenção

amostrador ou destilação	passos
DDPM (estocástico)	1000
DDIM (determinístico, inversível)	50
Euler, Heun	20–30
DPM-Solver++ (o padrão prático)	15–25
UniPC, DEIS	10–20
destilação progressiva	8192 → 4
LCM / LCM-LoRA (consistência)	2–8
SDXL-Turbo, ADD (com discriminador)	1–4

Capítulo a capítulo

A pergunta que cada capítulo responde, o que levar dele e a armadilha que pega mais gente.

1 · Conceitos — aprendizado

Onde ficam as redes neurais entre os jeitos de construir IA?
IA simbólica, conexionista e neuro-simbólica; IA ⊃ ML ⊃ redes neurais ⊃ deep learning; supervisionado, não supervisionado, autossupervisionado.
⚠ **Armadilha:** chamar todo modelo aprendido de "deep learning".

2 · Dados — dados

O que pode quebrar antes de existir qualquer peso?
A divisão é o experimento. Nada se ajusta antes dela; divida pela unidade a generalizar; codifique pelas comparações reais; escale dentro do `Pipeline`.
⚠ **Armadilha:** um filtro, um scaler ou oversampling rodando antes da divisão.

3 · Redes neurais — aprendizado

Como se foi de um modelo de neurônio ao deep learning?
1943 McCulloch–Pitts, 1958 perceptron, 1969 o inverno do XOR, 1986 backpropagation, 1989 aproximação universal, 2012 AlexNet, 2017 Transformer.
⚠ **Armadilha:** "uma camada oculta aproxima qualquer coisa" não diz que é fácil de aprender.

4 · Perceptron — aprendizado

O que um neurônio aprende corrigindo os próprios erros?
$w \leftarrow w + \eta(y - \hat{y})x$ converge se e somente se as classes forem linearmente separáveis.
⚠ **Armadilha:** esperar que ele resolva o XOR.

5 · Perceptron multicamadas — aprendizado

Como o erro da saída chega ao primeiro peso?
Forward, perda, backward pela regra da cadeia ($\delta_h = (V^\top \delta_y) \odot \sigma'$), atualização. É a não linearidade que faz duas camadas valerem mais que uma.
⚠ **Armadilha:** sigmoide em toda camada: cada uma divide o gradiente por 4.

6 · Otimização — aprendizado

Que tamanho de passo, em que direção, com quantos dados?
$\eta < 2/L$; ruído $\propto 1/\sqrt{B}$; o momentum faz média, os adaptativos reescalam; AdamW + warmup + cosseno; ajuste η primeiro.
⚠ **Armadilha:** o momentum do PyTorch dá um passo $10\times$ o do livro com $\beta = 0{,}9$.

7 · Regularização — aprendizado

Como fazer uma rede parar de decorar?
Ajustar pior de propósito: weight decay, dropout $(1/(1-p)$ no treino), normalização, parar cedo, augmentation; mais dados ganham de todos.
⚠ **Armadilha:** esquecer o `model.eval()`.

8 · Métricas e avaliação — dados

Que número decide e quanto custa um erro?
Precisão sobre o que se afirmou, revocação sobre a verdade; PR-AUC com desbalanceamento; limiar pelos custos; RMSE com linha de base; FID + CLIP score; pass@k com o seu k.
⚠ **Armadilha:** acurácia com positivos raros; perplexidade entre tokenizadores.

9 · Redes profundas — arquiteturas

O que quebra ao empilhar muitas camadas e o que consertou?
Sinal e gradiente são produtos de L fatores $n\sigma^2\rho$: inicialização He ou Xavier, residuais $(1 + \partial f/\partial a)$, normalização.
⚠ **Armadilha:** um atalho sem normalização: tudo explode.

10 · Redes convolucionais — arquiteturas

O que se ganha supondo localidade e pesos compartilhados?
$C_{out}(C_{in}k^2 + 1)$ pesos, independente da imagem; o campo receptivo é o orçamento; equivariância pelo compartilhamento, invariância pelo pooling e pelo augmentation.
⚠ **Armadilha:** erros de forma: calcule $\lfloor (H + 2p - k)/s \rfloor + 1$.

11 · Atenção — linguagem

Como cada token relê todos os outros?
$\text{softmax}(QK^\top/\sqrt{d_k})V$: uma camada densa com pesos vindos da entrada. Tokens, embeddings, RoPE, máscaras, cabeças, cache KV.
⚠ **Armadilha:** tirar o $\sqrt{d_k}$: a softmax satura e o gradiente morre.

12 · Transformers — linguagem

Do último vetor à próxima palavra e o que mudou desde 2017?
Embedding → L blocos (atenção conversa, MLP pensa, fluxo residual) → logits → amostragem. Pré-LN, RMSNorm, SwiGLU, GQA, RoPE; $C \approx 6ND$.
⚠ **Armadilha:** achar que a temperatura muda o token mais provável.

13 · Vision Transformers — arquiteturas

E se uma imagem for só uma sequência de patches?
$N = HW/P^2$ tokens; uma prior fraca comprada com dados ou com receita; os ViTs venceram porque emitem tokens.
⚠ **Armadilha:** "ViT precisa de 300 milhões de imagens": boa parte disso era a receita de treino.

14 · Transfer learning — linguagem

Vale adaptar um modelo pré-treinado e como fazer isso barato?
Prompt, RAG e ferramentas antes; LoRA $W_0 + \frac{\alpha}{r}BA$ em todas as lineares; QLoRA com pouca memória; SFT → DPO → GRPO.
⚠ **Armadilha:** fine-tuning para ensinar fatos.

15 · Grandes modelos de linguagem — linguagem

O que faz de um preditor do próximo token um assistente e quanto custa um token?
O pré-treino comprime texto; o pós-treino faz o assistente; computação no teste e verificadores; MoE; decodificar é limitado por memória.
⚠ **Armadilha:** confiar em pass@k sem verificador ou numa taxa de segurança sozinha.

16 · Modelos generativos — generativos

Como transformar ruído que se sabe amostrar em dados que não se sabe?
Quatro decomposições (AR, latente, adversarial, tempo); a KL direta borra, a reversa colapsa.
⚠ **Armadilha:** tratar borrão e colapso de modos como bugs sem relação.

17 · VAEs — generativos

Como codificar uma distribuição e amostrar dela?
ELBO = reconstrução − KL para a prior; $z = \mu + \sigma\epsilon$; o β decide; hoje, o compressor congelado sob a difusão.
⚠ **Armadilha:** colapso da posterior escondido numa perda total que cai.

18 · GANs — generativos

E se a própria perda for aprendida?
$D^* = p_{\text{dados}}/(p_{\text{dados}} + p_G)$; perda não saturante; impeça o D de ficar afiado demais; a perda adversarial segue viva.
⚠ **Armadilha:** ler as perdas como progresso.

19 · CLIP — generativos

Como imagens e frases acabam num mesmo espaço?
InfoNCE simétrica sobre o batch, τ aprendido, vetores unitários; zero-shot é um classificador linear feito de frases; o SigLIP o substituiu.
⚠ **Armadilha:** confiar nele para contar ou compor.

20 · Stable Diffusion — generativos

Como o ruído vira a imagem que uma frase pediu?
Regressão do ruído; rodar num latente de VAE; condicionar por cross-attention; CFG com w; amostradores e destilação definem os passos.
⚠ **Armadilha:** w alto demais: imagens fora da variedade, saturadas.

21 · Flow matching — generativos

Gerar pode ser uma regressão sobre uma reta?
Regressão de $x_1 - x_0$ ao longo de $x_t = (1-t)x_0 + tx_1$; difusão é o caso curvo; poucos passos pedem reflow ou destilação.
⚠ **Armadilha:** trocar a convenção de $t = 0$ entre códigos.

22 · Diffusion Transformers — generativos

O que se ganha trocando a U-Net por um Transformer?
Patches do latente, condicionamento adaLN-Zero, atenção conjunta para o texto, RoPE por eixo e uma curva de escala.
⚠ **Armadilha:** esquecer que vídeo é limitado pela atenção: parte $N/(N + 6d)$.

23 · Geração autorregressiva — generativos

E se uma imagem for gerada como texto, token por token?
Um tokenizador (VQ, FSQ, LFQ) mais predição do próximo token; MaskGIT, VAR e MAR mudam a ordem ou o token; o caminho dos modelos unificados.
⚠ **Armadilha:** um codebook morto: a maioria dos códigos nunca é usada.