EASYDEEPLEARN

Deep Learning

Interview cheat sheet

214 questions · Neural networks, training tricks, CNNs, RNNs, transformers and everything in between.

#training #transformers #architectures #computer-vision #losses #fundamentals #regularization #optimizers #attention #distributed

Activations & fundamentals

ReLU vs sigmoid vs GELU — when do you use each?

Sigmoid/tanh saturate and cause vanishing gradients — avoid in hidden layers of deep nets, keep sigmoid for a binary output. ReLU is the default hidden activation: fast, sparse, but suffers from 'dying ReLU' (permanently zero neurons).

In one sentence, what is backpropagation?

Backpropagation is the reverse-mode automatic differentiation algorithm that efficiently computes gradients of a scalar loss with respect to every parameter of a neural network by traversing the computation graph backwards, applying the chain rule at each node.

State the universal approximation theorem in one sentence and its practical caveat.

A feedforward network with a single hidden layer of finite width can approximate any continuous function on a compact domain to arbitrary accuracy — but the theorem says nothing about width required, training feasibility, or generalization. In practice depth is much more parameter-efficient than width, which is why deep networks dominate.

Why does stacking linear layers without nonlinearity collapse to a single linear layer?

The composition of linear maps is a linear map: $W_2 (W_1 x + b_1) + b_2 = (W_2 W_1) x + (W_2 b_1 + b_2)$. No matter how many layers you stack, the effective function is W x + b.

Depth vs width — which do you scale first?

Depth compounds representational capacity exponentially in the number of piecewise-linear regions for ReLU networks, while width only grows it polynomially. So depth is usually the better initial investment for representation.

Why can't you initialize a neural net with all zeros?

Symmetric weights mean every neuron in a layer computes the same output and receives the same gradient — the layer effectively has one neuron. Training never breaks the symmetry.

What is a computation graph and why does autograd care?

A DAG whose nodes are operations and edges track data + gradient dependencies. During the forward pass, each op records its inputs so its backward function can be called.

What is 'dying ReLU' and how do you fix it?

A ReLU neuron that always outputs zero has a zero gradient everywhere and stops learning permanently. Causes: large negative bias, extreme learning rate, poor init.

What are SiLU (Swish) and GELU, and why do transformers prefer them over ReLU?

$\mathrm{SiLU}(x) = x \cdot \mathrm{sigmoid}(x)$. $\mathrm{GELU}(x) \approx$ x * Phi(x) — smooth version of ReLU using a Gaussian CDF.

What does temperature do in a softmax?

$\mathrm{softmax}(z / T)$: T > 1 flattens the distribution (softer, higher entropy — used in distillation for teacher outputs), T < 1 sharpens toward one-hot (used in decoding to make choices more deterministic), T = 0 is argmax. In LLM decoding, temperature is a knob for creativity vs determinism.

Why compute softmax + cross-entropy jointly via log-sum-exp?

Naive $\mathrm{softmax} = \exp(z_i) / \mathrm{sum}(\exp(z_j))$ overflows for large logits. Compute $\log - \mathrm{softmax}(z_i) = z_i - \log_{\mathrm{sum}} \exp(z)$ where $\log_{\mathrm{sum}} \exp(z) = \max(z) + \log(\mathrm{sum}(\exp(z - \max(z))))$.

How should the output head be designed for a regression task with a strictly positive target?

Options: (1) predict $\log(y)$ and exponentiate — implicit positivity, natural for right-skewed targets; (2) apply $\exp()$ or softplus at the output to guarantee positivity; (3) predict a mean of a Gamma/log-normal distribution via a GLM-style head. Never use identity output with MSE if y can only be positive — the model will predict negatives during training which then contribute noise.

How is the output head different for multi-label vs multi-class classification?

Multi-class: one softmax head, one loss (categorical cross-entropy) — labels are mutually exclusive. Multi-label: K independent sigmoid heads, one binary cross-entropy loss per label, summed — a document can have multiple tags simultaneously.

Do you need bias terms in every layer of a modern neural network?

Often no. When a layer is followed by BatchNorm or LayerNorm with learned shift (beta), the bias is redundant — the norm's beta term absorbs it.

How do you compute the parameter count of a linear layer and a convolution?

Linear(in, out): in * out + out (weights + bias).
$\mathrm{Conv2d}(C_{\mathrm{in}}, C_{\mathrm{out}}, \mathrm{kernel} = k, \mathrm{stride} = s, \mathrm{pad} = p)$:
$C_{\mathrm{in}} \cdot C_{\mathrm{out}} \cdot k \cdot k + C_{\mathrm{out}}$.

How do FLOPs scale for a forward pass through an MLP and a transformer?

MLP layer(in, out) on batch B: 2 * B * in * out FLOPs. Transformer forward (per token, per layer) ≈ $2 \cdot n_{\text{layers}} \cdot d \cdot d_{\text{ffn}}$ $(\text{MLP}) + 2 \cdot n_{\text{layers}} \cdot d \cdot (d + d_{\text{head}})$ $(\text{attention linear projections}) + 2 \cdot n_{\text{layers}} \cdot \text{seq} \cdot d$ (attention softmax + matmuls).

Why does tanh cause vanishing gradients in deep nets?

$\tanh'(x) = 1 - \tanh^2(x) \leq 1$, and is close to 1 only near x = 0. Deep nets keep multiplying such derivatives during backprop — the product shrinks toward 0.

What is a forward hook in PyTorch and when is it useful?

A callback registered on a module that fires during the forward pass, receiving (module, input, output). Useful for extracting intermediate features (e.g., pen-ultimate embeddings for downstream models), debugging shape / value issues, visualizing activations, running feature-map probes, or computing custom regularizers on hidden activations without modifying the model code.

In one sentence, what is broadcasting and why does it matter?

Broadcasting extends tensor operations across dimensions of unequal shape by implicit repetition, with rules: align shapes from the right, dimensions of size 1 are broadcast to match, otherwise dimensions must match. It lets you write add, multiply, and mask operations across batches and features without explicit tile/repeat — critical for efficient GPU code.

Compute the output shape of a 2D convolution with input (H, W), kernel k, stride s, padding p, dilation d.

$H_{\text{out}} = \text{floor}((H + 2*p - d*(k-1) - 1) / s) + 1$, same for W. Special cases: 'same' padding for stride 1 uses p = (k-1)/2 * d.

Write the vanilla RNN update equation and explain why it struggles with long sequences.

$h_t = \tanh(W_{\text{hh}} \cdot h_{t-1} + W_{\text{xh}} \cdot x_t + b)$. Repeatedly multiplying by W_{hh} during BPTT causes gradients to vanish (spectral norm < 1) or explode (> 1) over long sequences.

MSE vs MAE for regression — when is each preferred?

MSE (L2): differentiable everywhere, penalizes large errors quadratically → sensitive to outliers, gradients are proportional to error size. MAE (L1): penalizes linearly → robust to outliers but not differentiable at 0.

Why is 'overfit a single batch' the first test you should run on a new model?

Because it isolates implementation bugs from learning problems. A correctly wired model with enough capacity can drive the loss on a handful of examples to nearly zero, since it can simply memorize them.

The same training script gives different results on two runs. How much of that can you remove?

Less than people expect, and it is worth knowing which sources you control. Seeding the framework, Python and NumPy fixes weight initialization, dropout masks and shuffling order, which removes most of the variance.

What do residual connections actually fix?

They make optimization tractable at depth. Without them, a deep stack must learn an identity mapping through many nonlinear layers just to preserve information, which gradient descent does badly, and the observed symptom is that a deeper network trains to a worse training error than a shallower one.

Backpropagation & autograd

Why does the backward pass require more memory than the forward pass?

Reverse-mode autodiff needs the intermediate activations from the forward pass to compute gradients (the chain rule uses them). So all activations are stored until backward finishes.

What is backpropagation through time (BPTT)?

Unroll the RNN over the sequence length T, treat it as a very deep feed-forward network with shared weights, and apply standard backprop. Memory scales linearly with T.

Initialization & gradients

What are vanishing and exploding gradients, and how do you fix them?

In deep networks, chain-rule products can shrink toward zero (vanishing) or blow up (exploding), stalling learning. Fixes: better initializations (He for ReLU, Xavier for tanh), normalization (BatchNorm, LayerNorm), skip connections (ResNet), non-saturating activations (ReLU family), gradient clipping (for RNNs and large models), and appropriate learning rates.

What is Xavier (Glorot) initialization and why?

Sample weights from a distribution with variance $2 / (\text{fan}_{\text{in}} + \text{fan}_{\text{out}})$ — designed so that activations and gradients have roughly the same variance layer to layer for tanh/sigmoid activations. Prevents both vanishing and exploding signals at initialization.

Why does He initialization use $\text{Var} = 2 / \text{fan}_{\text{in}}$ instead of Xavier's $2 / (\text{fan}_{\text{in}} + \text{fan}_{\text{out}})$?

ReLU zeros out half of its inputs on average, halving the effective variance of activations. He init doubles the variance to compensate: $\text{Var}(W) = 2 / \text{fan}_{\text{in}}$.

When is orthogonal weight initialization useful?

Orthogonal init sets weight matrices to random orthogonal matrices (singular values all 1), which preserves the norm of the input under linear map. Especially useful for very deep networks and RNNs where repeated multiplication amplifies or shrinks signals — orthogonal keeps things unit-norm.

What causes exploding gradients and how do you handle them?

Exploding gradients happen when the product of many derivatives with singular values > 1 grows exponentially. Common in RNNs $(\text{repeated multiplication by } W_{\text{hh}})$ and very deep networks without normalization.

Gradient clipping by norm vs by value — which do you prefer?

Clip-by-norm: rescale the whole gradient vector so its L2 norm is at most max_{norm}. Preserves direction, only shrinks magnitude.

Training dynamics

What does Batch Normalization do?

For each mini-batch, BN normalizes activations to zero mean and unit variance per feature, then applies learned scale and shift. Effects: faster convergence, higher learning rates, mild regularization, and reduced internal covariate shift.

How does dropout work and when is it applied?

During training, dropout randomly zeros a fraction p of activations per forward pass, forcing the network to distribute knowledge and not co-adapt. At inference, all units are active and outputs are scaled (or scaling is done at train time — 'inverted dropout').

Why use a learning-rate schedule (warmup + cosine decay)?

Large models with LayerNorm behave badly at high LR early on — warmup ramps LR from 0 to peak over a few thousand steps to avoid divergence. Then cosine (or linear) decay reduces LR smoothly toward zero, which improves final generalization.

Why do residual (skip) connections help training deep networks?

They provide a shortcut so gradients can flow directly back to earlier layers, alleviating vanishing gradients. They also make it easy for a layer to learn the identity function, so adding depth can't hurt.

What is gradient accumulation and when do you use it?

Run several forward-backward passes with small mini-batches without calling optimizer.step(); sum the gradients. Then apply one optimizer step as if you had used a large batch.

How does batch size affect training dynamics?

Larger batch → lower gradient noise, closer to true gradient, allows higher learning rate (roughly LR ∝ sqrt(batch)). But too large batches tend to converge to sharper minima that generalize slightly worse (the 'generalization gap').

What is deep double descent?

Test error as a function of model capacity is not monotonically U-shaped: it goes up as capacity crosses the interpolation threshold (train error → 0), then goes down again for extremely overparameterized models. Documented in Belkin et al. (2019) and Nakkiran et al. (2020).

What is grokking in deep learning?

Phenomenon (Power et al., 2022) where a network trained on a small algorithmic task memorizes the training set quickly (train loss → 0) but validation stays random for a long time — then, after many more epochs of continued training, suddenly generalizes to near-perfect validation accuracy. Suggests the network first memorizes, then slowly discovers a more compressed, generalizable circuit under continued regularization pressure.

In one sentence, what is the Lottery Ticket Hypothesis?

Frankle & Carbin (2019): a randomly initialized dense network contains sparse subnetworks ('winning tickets') that — when trained in isolation from their original initialization — reach comparable accuracy in comparable or fewer iterations. Motivates pruning research: identify and train these tickets to shrink models dramatically without loss.

What is the 'implicit bias' of SGD?

Among the infinite minimizers of the training loss in an over-parameterized net, SGD tends to select ones with certain 'flat' properties — approximately minimum-norm solutions in linear cases, and empirically flat-minimum / max-margin solutions in nonlinear ones. This implicit bias explains why deep nets trained by SGD generalize despite having capacity to fit noise.

Your training loss becomes NaN. Debug it.

(1) Check for divisions by zero and log of non-positive values — most common cause. (2) Reduce learning rate; loss NaN often means gradient exploded.

What problem does LAMB solve?

LAMB (Layer-wise Adaptive Moments for Batch training) enables very large-batch training (32k+ per step) without loss of accuracy. It scales the Adam update per layer by the ratio $||\theta_{\text{layer}}|| \,/\, ||\text{update}_{\text{layer}}||$.

Describe the LR-range finder (Smith, 2015).

Train the model for a few epochs with the learning rate increased exponentially from very small to very large. Plot loss vs LR.

What is the one-cycle policy?

Smith's 'super-convergence' recipe: linearly ramp LR up from $\text{LR}_{\max}/25$ to $\text{LR}_{\max}$ over the first ~30% of training, then linearly decay back down. Simultaneously modulate momentum in the opposite direction (high when LR is low, low when LR is high).

How should learning rate scale with batch size?

Linear rule (Goyal et al., 2017): $\text{LR} = \text{base}_{\text{LR}} \cdot (\text{batch} \,/\, \text{base}_{\text{batch}})$. Works up to ~8k batch in vision with sufficient warmup.

What is gradient noise scale (GNS) and how do you use it?

GNS (McCandlish et al., 2018) measures the ratio between the variance of the mini-batch gradient and its squared mean, roughly telling you how much batch size you can grow before returns diminish. Compute $B_{\text{crit}} = \text{trace}(\text{Cov}(g)) \,/\, ||g||^2$.

What is LARS and when is it used?

Layer-wise Adaptive Rate Scaling (You et al., 2017). Scales the update per layer by $||\theta_{\text{layer}}|| \,/\, ||g_{\text{layer}}||$, keeping the ratio update-to-weight at ~1%.

Why does a model with BatchNorm behave differently at train time vs eval time?

Train: BN uses batch statistics (mean/var of the current mini-batch), providing regularization noise. Eval: uses stored running averages accumulated during training via EMA.

What is stochastic depth / DropPath and where is it used?

During training, randomly drop entire residual branches (set them to 0 and pass identity through the skip connection) with some probability p that increases with depth. Effectively trains an ensemble of networks of varying depth.

What is early stopping and how do you configure it?

Monitor a validation metric each epoch; stop training when it hasn't improved for 'patience' epochs. Restore the best-performing checkpoint.

What is Stochastic Weight Averaging (SWA)?

During the last part of training (with a constant or cyclical LR), keep a running average of the weights: $\theta_{\text{swa}} = \text{mean}$ of θ_t across recent steps. Use θ_{swa} at inference.

How does snapshot ensembling work?

Train with a cyclical LR schedule (SGDR / cosine with restarts). Save a checkpoint at each cycle's minimum LR (one snapshot per cycle).

What is adversarial training (PGD, FGSM)?

Generate small adversarial perturbations that fool the current model (FGSM = one gradient step; PGD = multiple projected gradient steps), then include them in training with the true labels. Improves robustness to bounded input perturbations at ~2-10x compute overhead.

How are input images typically normalized for pretrained CNNs?

Subtract ImageNet channel means [0.485, 0.456, 0.406] and divide by stds [0.229, 0.224, 0.225] (RGB). This matches the distribution the model saw during pretraining — using different stats can hurt accuracy noticeably during transfer.

What is a solid default augmentation recipe for training a modern vision model?

Random resized crop, horizontal flip, color jitter (brightness/contrast/saturation/hue), and one of RandAugment / TrivialAugment for stronger transforms. Add Random Erasing / Cutout.

How should training resolution be chosen for a CNN or ViT?

For CNNs: bigger resolution → higher accuracy up to a point (diminishing returns and OOM). Common choices: 224 for ResNet baselines, 256/288/380/456 for EfficientNet variants (compound scaling), 384/512 for ViT-Large.

When do you use truncated BPTT and what's the trade-off?

For very long sequences (language modeling on 1000+ tokens, time-series, audio). Unroll for k steps, backward, then carry hidden state forward but detach it from the graph.

What is teacher forcing and its main pitfall?

During training, feed the ground-truth previous token as input to the decoder at each step (instead of the model's own previous prediction). Speeds training and avoids compounding errors from incorrect early predictions.

What is scheduled sampling?

Interpolate between teacher forcing and self-generation during training: with probability p feed the true token, with 1-p feed the model's own prediction. Decay p over training (start at 1, end near 0).

Contrast the pretraining objectives of BERT, GPT, and T5.

BERT: masked language modeling — mask 15% of tokens, predict them (bidirectional context). GPT: causal / next-token prediction — predict token t from tokens 1..t-1 (unidirectional).

What is deep supervision / auxiliary loss?

Attach small classification / regression heads to intermediate layers and add their losses to the main loss (with lower weights). Provides gradient signal deeper into the network — helps optimization of very deep nets and enables intermediate feature usefulness.

How does mixed-precision training work with fp16 / bf16?

Store weights in fp32, cast to fp16 (or bf16) for forward and backward passes, accumulate gradients in fp32, then update weights in fp32. Cuts activation memory ~2x and speeds up compute 2-3x on modern GPUs with tensor cores. fp16 has small range (needs loss scaling to prevent gradient underflow); bf16 has fp32's range but less precision — usually plug-and-play, preferred on Ampere+.

Why does fp16 training need loss scaling?

fp16 has a small dynamic range (~5e-8 to 6.5e4). Gradients often live near the underflow boundary — many gradient elements silently become zero.

How does PyTorch DDP work under the hood?

Each GPU has a full replica of the model + its own data slice. Forward and backward happen independently.

Model parallel vs data parallel — when do you need each?

Data parallel: replicate the model, split the batch across GPUs — dominant approach up to ~10B params on a single node. Model parallel: split the model itself across GPUs — needed when the model exceeds one GPU's memory.

Explain the three ZeRO stages.

ZeRO (Zero Redundancy Optimizer) partitions training state across GPUs to save memory. Stage 1: shard the optimizer state (Adam's m and v) across N GPUs → memory / N.

How is FSDP different from DDP?

PyTorch FSDP (Fully Sharded Data Parallel) implements ZeRO-3: each GPU stores only a shard of the parameters, gradients, and optimizer state. When a layer runs forward, FSDP all-gathers its full parameters, computes, then discards them (or shards again).

What is an EMA of weights and why do modern training recipes use it?

Maintain a shadow copy of the weights θ_{ema} updated as $\theta_{\text{ema}} \leftarrow \alpha \cdot \theta_{\text{ema}} + (1-\alpha) \cdot \theta$ every step, with α ~ 0.999-0.9999. Evaluate / deploy with θ_{ema}, not the raw training weights.

State the Chinchilla scaling insight in one sentence.

For a given compute budget C = 6 * N * T (params × tokens), the compute-optimal split has N and T roughly proportional (~20 tokens per parameter), meaning older large models (GPT-3, Gopher) were undertrained: same compute would have given a smaller model trained on more data with much better loss. Reshaped modern LLM training toward smaller-but-longer-trained models (LLaMA, Mistral).

Your validation loss is lower than your training loss. Is something broken?

Usually not, and there are three ordinary explanations before you go looking for a bug. Regularization is active during training but not at evaluation, so dropout and stochastic depth make the training forward pass genuinely harder than the validation one.

Your model trains well at batch size 256 but degrades at batch size 4. Why might batch normalization be the culprit?

Batch norm estimates the mean and variance from the batch itself, so with four samples those statistics are extremely noisy. The noise acts as an uncontrolled regularizer during training and, worse, the running averages accumulated for inference no longer match what the layer saw, so train and eval behaviour diverge.

Your augmentation pipeline made validation accuracy worse. What went wrong?

Almost always a distribution mismatch: the augmentation created inputs unlike anything at test time, so the model spent capacity on a harder problem than the one it is graded on. Classic examples are horizontal flips on digits or text, aggressive colour jitter when colour is the label signal, and rotations on medical scans with a canonical orientation.

How does gradient accumulation let you train with a batch that does not fit in memory, and what does it not fix?

You run several smaller micro-batches, summing or averaging their gradients, and only step the optimizer once at the end. Mathematically the update matches a single large batch, so the loss curve is nearly identical while peak activation memory stays at the micro-batch level.

Optimizers

SGD vs Adam vs AdamW — how do you choose?

SGD with momentum is simple and often generalizes better on vision when tuned carefully with LR schedules. Adam adapts per-parameter learning rates using first and second moment estimates — great default, converges fast, robust to hyperparameters.

Write the vanilla SGD update rule and explain each term.

$\theta_t + 1 = \theta_t - \eta \cdot g_t$, where θ is the parameter vector, η the learning rate, and g_t = ∇_θ $L(\theta_t \text{ batch}_t)$ is the stochastic gradient on the current mini-batch. Each step is a noisy estimate of the true gradient — the noise itself acts as regularizer and helps escape saddle points.

How does classical momentum modify SGD?

Maintain a velocity: $v_t = \beta \cdot v_t - 1 + g_t$; then $\theta_t + 1 = \theta_t - \eta \cdot v_t$. Typical β = 0.9.

What is Nesterov momentum and how is it different from classical momentum?

Look-ahead trick: compute the gradient at the parameters shifted by the current velocity, not at the current parameters. Update: $v_t = \beta \cdot v_t - 1$ + $\nabla L(\theta_t - \eta \cdot \beta \cdot v_t - 1)$; $\theta_t + 1 = \theta_t - \eta \cdot v_t$.

What is AdaGrad and its main drawback?

Per-parameter learning rate: θ ← θ - η * g / (sqrt(sum of squared past gradients) + ε). Parameters with large historical gradients get smaller effective LRs — great for sparse features (NLP with one-hot inputs).

How does RMSProp fix AdaGrad?

Replaces the running sum of squared gradients with an exponential moving average: $v_t = \beta \cdot v_t - 1 + (1 - \beta) \cdot g_t^2$. Update: θ ← $\theta - \eta \cdot g$ / $(\text{sqrt}(v_t) + \varepsilon)$.

Write Adam's full update rule.

$m_t = \beta_1 \cdot m_t - 1 + (1 - \beta_1) \cdot g_t$ (first moment);
$v_t = \beta_2 \cdot v_t - 1 + (1 - \beta_2) \cdot g_t^2$ (second moment). Bias-corrected: $m_{\text{hat}} = m_t$ / $(1 - \beta_1^t)$, $v_{\text{hat}} = v_t$ / $(1 - \beta_2^t)$.

Why does AdamW work better than Adam + L2 weight decay for transformers?

In Adam, adding λ*θ inside the gradient means weight decay gets rescaled by the per-parameter learning rate $1/\text{sqrt}(v_{\text{hat}})$. Parameters with small gradients get almost no decay; those with large ones get too much.

What is the Lion optimizer and why is it interesting?

Lion (Chen et al., 2023, from Google) uses only the sign of the momentum: θ ← $\theta - \eta \cdot \text{sign}(\beta_1 \cdot m + (1 - \beta_1) \cdot g)$, then m ← $\beta_2 \cdot m + (1 - \beta_2) \cdot g$. Half the optimizer state of Adam (no v), often trains 1.5-2x faster in wall-clock at same accuracy on transformers and vision.

Why does Adafactor use less memory than Adam?

Adafactor factorizes the second-moment matrix v: instead of storing a full v_t per parameter (O(N) memory), it stores row-sum and column-sum statistics for each 2D weight matrix (O(sqrt(N))). Massive memory savings — enabled T5 training.

What is Shampoo and when does second-order optimization pay off?

Shampoo (Gupta et al., 2018 / Anil et al., 2020) is a second-order-ish optimizer that maintains a per-parameter matrix preconditioner factorized along tensor axes — computes the inverse Kronecker-factored covariance of gradients. Uses more compute per step than Adam but converges in far fewer steps.

How does weight decay change the objective and the update?

Adds an L2 penalty λ/2 * ||θ||² to the loss. In SGD: θ ← θ - η * (∇L + λ*θ) = (1 - η*λ) * θ - η * ∇L — shrinks weights each step.

How much extra memory does Adam use versus SGD?

SGD with momentum: 1x parameter count in optimizer state (momentum). Adam: 2x parameter count (first and second moment estimates).

What does SAM (Sharpness-Aware Minimization) do?

SAM (Foret et al., 2020) minimizes a surrogate that penalizes sharp minima: for each step, first perturb θ in the direction that maximizes the loss within a small ball ($\theta + \rho \cdot \nabla L$ / $||\nabla L||$), then compute the gradient at that perturbed point and update the original θ. Roughly 2x compute per step, but consistently improves generalization on vision and NLP benchmarks.

Why is weight decay via L2-in-loss different from decoupled weight decay in Adam?

L2 in loss: gradient becomes g + λ*θ, then Adam divides by $\text{sqrt}(v_{\text{hat}})$. Parameters with large gradients get smaller effective weight decay — inverted relative to what you want.

When do you prefer SGD with momentum over Adam?

For convolutional vision models trained from scratch to their best possible accuracy, tuned SGD with momentum still tends to generalize slightly better than Adam, which is why many image classification recipes use it. Adam wins where gradient scales differ wildly across parameters: transformers, embeddings with sparse updates, and anything with attention.

Why does AdamW handle weight decay differently from Adam, and why does it matter?

In Adam, weight decay is added into the gradient, so it passes through the adaptive scaling. Parameters with a large accumulated second moment get their decay divided down, which means the effective regularization differs per parameter and does not match what you asked for.

Learning-rate schedules

What are cyclical learning rates (CLR)?

Instead of monotonic decay, oscillate LR between a low and a high bound in periodic triangular / sinusoidal / exp cycles. Helps escape shallow local minima (high LR periods) and refines fits (low LR periods).

Why do transformers need learning-rate warmup?

At the start of training, weights are random and LayerNorm statistics are meaningless. A high LR causes large gradients that destabilize LayerNorm's running scale and can permanently damage the network (attention heads collapse).

What is cosine annealing and why is it preferred over step decay?

LR(t) = LR_min + 0.5 * (LR_max - LR_min) * (1 + cos(π * t / T)). Starts high, decays smoothly to LR_{min} over T steps.

When is step decay still appropriate?

Multiply LR by a factor (e.g., 0.1) at fixed milestones (e.g., 60/90 epochs on ImageNet). Simple, deterministic, well-understood — classic recipe for training ResNet-style CNNs with SGD+momentum.

How does ReduceLROnPlateau work?

Track a validation metric; if it doesn't improve for 'patience' epochs, multiply LR by a factor (e.g., 0.5). Adaptive to the actual loss curve — doesn't require guessing the schedule in advance.

What is SGDR (warm restarts) and when does it help?

Cosine annealing with periodic restarts back to $\mathrm{LR}_{\max}$: each 'cycle' cools LR to $\mathrm{LR}_{\min}$ then jumps back up. Enables ensembling by saving one model per cycle (Snapshot Ensembles).

Why does transformer training usually need a learning-rate warmup?

At initialization, Adam's second-moment estimate is based on almost no history, so its normalization is unreliable and the first updates can be enormous relative to the weights. Large early steps push a transformer into a region from which it never fully recovers, often visible as an attention collapse or an immediate loss spike.

Normalization

When do you use LayerNorm instead of BatchNorm?

Use LayerNorm when batch size is small or varies (RNNs, transformers, sequence data). LayerNorm computes statistics across features within a single example, so it doesn't depend on batch statistics.

What is GroupNorm and when do you use it?

Divide channels into G groups, normalize each group per example. No dependence on batch size — works with batch=1, useful for detection / segmentation where memory forces small batches, or for very high-resolution training.

What is InstanceNorm and where does it shine?

Normalize each channel independently within one example. Removes per-image style / contrast information — hence its use in style transfer and image-to-image translation (CycleGAN, StyleGAN).

What is Weight Normalization and its trade-off?

Reparameterize weights as w = g * v / ||v||, decoupling magnitude (g, scalar per neuron) from direction (v, vector). Data-independent normalization — no batch statistics.

How does RMSNorm differ from LayerNorm?

RMSNorm(x) = x / RMS(x) * γ, where $\mathrm{RMS}(x) = \mathrm{sqrt}(\mathrm{mean}(x^2) + \varepsilon)$. Skips the mean-subtraction and β (bias) of LayerNorm.

Pre-norm vs post-norm transformers — which is standard and why?

Original Transformer (Vaswani 2017) used post-norm: x + Attn(x), then LayerNorm. Trains poorly at scale (deep post-norm nets need long warmup).

What is Synchronized BatchNorm (SyncBN) and when do you need it?

In data-parallel training, each GPU has a mini-batch — regular BN computes stats per GPU. If per-GPU batch is small (e.g., 2-4 for detection / segmentation), stats are noisy and accuracy suffers.

Why is BatchNorm awkward in RNNs?

RNNs share weights across time; you'd need separate BN statistics for every timestep to be principled, but sequences have variable length. Also, small effective batch (batch × time reshape doesn't help because time is not exchangeable), and running stats depend on sequence length.

Why did transformers switch from post-norm to pre-norm at scale?

Original transformer used post-norm: LayerNorm(x + sublayer(x)). At depth > 12, the sublayer outputs interfere with LN's running scale, causing training instabilities that require very long warmup or careful initialization.

Regularization

What is DropConnect and how is it different from Dropout?

Dropout zeros activations; DropConnect zeros weights (elements of the weight matrix) — i.e., randomly severs individual connections. More granular regularization but harder to implement efficiently on GPUs.

Where is dropout typically inserted in a transformer block?

(1) On attention weights (after softmax), (2) on attention output projections, (3) on MLP hidden and output activations, (4) on residual sums (dropout after each residual add — 'residual dropout'). Rates: ~0.1 for pretraining (BERT), sometimes 0.0 for LLM pretraining (large models overfit less).

What is label smoothing and why does it help?

Replace one-hot target [0, ..., 1, ..., 0] with (1-α)*one-hot + α/K (uniform), typically α=0.1. Prevents the network from becoming over-confident: it can't drive the softmax logit for the true class to infinity because the target isn't exactly 1.

How does Mixup work?

Sample two examples $(x_i,\ y_i)$, $(x_j,\ y_j)$ and a mixing coefficient λ ~ Beta(α, α) with α ~ 0.2. Train on the interpolation:
$x_{\text{new}} = \lambda \cdot x_i + (1-\lambda) \cdot x_j$, $y_{\text{new}} = \lambda \cdot y_i + (1-\lambda) \cdot y_j$.

What is Cutout / Random Erasing?

During training, mask a random rectangular region of the input image with zeros (Cutout) or random noise (Random Erasing). Forces the network to look at the whole image rather than one salient region — improves robustness to occlusion.

Why is data augmentation effectively free regularization?

Every augmentation is a prior about invariances the true function satisfies (translations of a cat are still a cat). Applying augmentations enlarges the effective training distribution without labeling cost.

What is consistency regularization?

Enforce that two augmented views of the same unlabeled input produce similar predictions: $L_{\text{cons}} = D$(f(aug1(x)), f(aug2(x))) with D = KL or MSE. Powers modern semi-supervised methods (FixMatch, Mean Teacher, MixMatch) and self-supervised learning (SimCLR, MoCo, BYOL).

What is Manifold Mixup?

Variant of Mixup (Verma et al., 2019): at each step, pick a random hidden layer k and linearly interpolate the hidden representations of two examples at that layer (rather than at the input). Encourages smoother hidden representations, further improves calibration and robustness.

Why does dropout play a smaller role in large language model training than in older vision models?

Because the regularization pressure comes from the data instead. When a model sees each token roughly once, on a corpus far larger than its parameter count, memorization is not the binding constraint, so the variance reduction dropout provides is not needed and its added gradient noise mostly slows convergence.

Data augmentation & mixing

How does CutMix differ from Mixup?

Cut a rectangular patch from image B and paste it into image A. The label is mixed by the area ratio: $y = \lambda \cdot y_A + (1-\lambda) \cdot y_B$.

What is RandAugment and why is it a nice augmentation policy?

Instead of learning an augmentation policy (AutoAugment), RandAugment randomly picks N transforms from a fixed pool (rotate, shear, color jitter, ...) each with a shared magnitude M. Two hyperparameters (N, M) instead of dozens — much easier to tune, and matches AutoAugment's accuracy on ImageNet.

What is Test-Time Augmentation (TTA)?

At inference, apply K augmentations (crops, flips, color jitter) to the same input, run K forward passes, and average the predictions. Better probability estimates, often 0.5-2% accuracy gain in classification and detection, at K× inference cost.

Convolutional networks

What inductive biases do CNNs have?

(1) Locality: convolutions look at small neighborhoods. (2) Translation equivariance: shifting the input shifts the feature map.

What is the receptive field in a CNN?

The receptive field of a neuron is the region of the input image that influences its value. It grows with depth, kernel size, and stride/dilation.

SAME vs VALID vs REFLECT padding — what are the practical differences?

VALID = no padding (output shrinks each layer). SAME = pad zeros so output has same H, W as input for stride 1.

What is a dilated (atrous) convolution?

Insert (d-1) zeros between kernel elements to enlarge the receptive field without adding parameters or reducing resolution. A 3×3 kernel with dilation 2 has an effective 5×5 view.

What is a transposed convolution ('deconv') and its checkerboard artifact issue?

A convolution that upsamples by inserting zeros between pixels then applying a normal conv — output is larger than input. Used in decoders (U-Net upsampling), GAN generators, and dense prediction.

How does a depthwise separable convolution reduce compute?

Factor a standard conv into (1) depthwise: one k×k filter per input channel independently; (2) pointwise: a 1×1 conv mixing the depthwise outputs across channels. Compute goes from $C_{\text{in}} \cdot C_{\text{out}} \cdot k \cdot k$ to $C_{\text{in}} \cdot k \cdot k \; + \; C_{\text{in}} \cdot C_{\text{out}}$ — for k=3, $C_{\text{in}} = C_{\text{out}} = 512$, that's 8-9x cheaper.

What are 1x1 convolutions used for?

Three roles: (1) channel-mixing / channel-wise projection — a matrix multiply across channels per spatial location, so it's a cheap way to change the number of channels; (2) bottleneck in ResNet blocks (reduce channels → 3×3 conv → expand channels) to save compute; (3) pointwise conv in depthwise-separable architectures. Effectively a per-position linear layer.

Max pooling vs average pooling — when do you pick each?

Max pool: keeps the strongest activation in the window — sharper features, invariance to small translations, dominant in classification CNNs. Average pool: smoother, retains overall energy — better in generative / decoder paths where you don't want to lose detail.

Why does Global Average Pooling replace the FC head in modern CNNs?

Take the mean of each feature map across H, W to get a C-dim vector, then apply a linear classifier. Advantages: (1) drastic parameter reduction — no huge FC on flattened features; (2) intrinsic regularization; (3) enables class activation maps (CAM) since one channel per class is aligned with a class score; (4) works with any input resolution.

What is adaptive pooling and why is it useful?

Given a target output size (H_{out}, W_{out}), computes the kernel and stride needed to produce that output regardless of input size. Standard trick to accept variable input resolutions with a fixed-size classifier head. torch.nn.AdaptiveAvgPool2d((1,1)) is the go-to for global pooling.

What is ConvNeXt's philosophy?

Liu et al. (2022) 'modernize' a ResNet using tricks from Swin: bigger kernels (7×7 depthwise), GELU + LayerNorm, inverted bottlenecks, fewer activations / norms, and modern training recipes (AdamW + Mixup + RandAugment). The result is a pure-conv architecture that matches Swin Transformer on ImageNet, detection, and segmentation.

CNN architectures

What made AlexNet (2012) a breakthrough?

AlexNet: 8 layers, ReLU activations (huge speedup vs sigmoid), dropout (novel regularization), overlapping max pooling, local response normalization (LRN, now obsolete), heavy data augmentation, and GPU training split across 2 GTX 580s. Won ImageNet 2012 by 10+ percentage points — kicked off the deep-learning revolution and made GPUs mandatory for vision.

What is the design principle behind VGG?

Stack 3x3 conv blocks repeatedly (2, 3, or 4 in a row) and downsample with max-pooling. Two stacked 3x3 convs have the same receptive field as one 5x5 but with fewer parameters and one extra nonlinearity — makes deeper nets with more nonlinearity cheap.

What is the Inception module?

Multi-branch block that applies 1x1, 3x3, 5x5 convolutions and 3x3 max pool in parallel, concatenates the outputs along channels. Each branch captures features at a different scale. 1x1 bottlenecks reduce cost.

What is the key insight of ResNet's identity mapping?

A residual block computes y = F(x) + x, where F is a stack of convolutions. If the optimal function is close to identity, the block only needs to learn a small correction F(x) ≈ 0 — easier than learning the identity from scratch.

How is DenseNet different from ResNet?

In a DenseNet block, each layer receives the concatenated feature maps of every preceding layer, not just the previous one. Massive feature reuse — each layer produces only 'growth rate' k channels (typically 12-32), so the whole network has surprisingly few parameters.

What is MobileNet designed for?

Efficient inference on mobile and embedded devices. Uses depthwise-separable convolutions everywhere and a 'width multiplier' α scaling all layer widths, plus a resolution multiplier ρ.

What is compound scaling in EfficientNet?

Scale depth (d), width (w), and resolution (r) together by a single coefficient φ: d = α^φ, w = β^φ, r = γ^φ, with $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ to keep FLOPs proportional to 2^φ. Better than scaling one dimension at a time.

How does a Vision Transformer (ViT) treat images?

Split image into fixed-size patches (e.g., 16×16), linearly embed each patch (like word embeddings), add positional embeddings, prepend a learnable [CLS] token, and feed to a standard transformer encoder. The CLS token's final representation is the image feature for classification.

What does Swin Transformer do differently from ViT?

Swin uses hierarchical (multi-scale) feature maps like CNNs, with attention computed inside local windows (e.g., 7×7 patches). Adjacent transformer blocks shift the windows so information flows across window boundaries — hence 'Swin' (Shifted Windows).

How does a two-stage detector (Faster R-CNN) work?

Stage 1: a Region Proposal Network (RPN) sliding a small conv over the backbone's feature map generates ~1k-2k class-agnostic object proposals (anchor boxes + objectness). Stage 2: RoIAlign extracts fixed-size features from each proposal, then a small head classifies the region and refines the box.

How does YOLO / SSD / RetinaNet differ from two-stage detection?

One-stage detectors predict class and box in a single forward pass over dense anchor / grid locations — no proposal step. YOLO: divide image into an S×S grid, each cell predicts B boxes + class probs.

What is Non-Maximum Suppression (NMS) and its variants?

Post-processing that removes duplicate detections: sort boxes by confidence, keep the top one, remove all others with IoU > threshold with it, repeat. Standard threshold: 0.5.

What are anchor boxes and their downsides?

Predefined reference boxes of various sizes and aspect ratios placed at every location of the feature map. The network predicts offsets and class scores relative to each anchor.

Why is U-Net so popular for segmentation?

Encoder-decoder with skip connections between symmetric layers. Encoder downsamples via convs + pooling; decoder upsamples via transposed convs / bilinear + convs, and concatenates the same-resolution encoder features.

How does Mask R-CNN extend Faster R-CNN for instance segmentation?

Add a parallel FCN mask-prediction head to each RoI, predicting a small binary mask (28×28 typically) per class. Replaces RoIPool with RoIAlign (bilinear interpolation instead of quantization) — critical for precise mask localization.

What is a Feature Pyramid Network (FPN)?

A top-down architecture that fuses features from multiple backbone stages (different resolutions and semantic levels) into a pyramid where each level is high-resolution and semantically rich. Take the deep low-resolution feature, upsample, add to the mid-resolution feature, and so on.

How does DETR reformulate object detection?

DETR (Carion 2020) treats detection as a set-prediction problem: a transformer decoder outputs a fixed set of N (>>true count) predictions in parallel, matched to ground-truth objects via bipartite Hungarian matching during training. No anchors, no NMS.

Describe a single transformer encoder block in detail.

Input x. Sublayer 1: y = x + Dropout(MultiHeadAttention(LayerNorm(x))).

How does a decoder block differ from an encoder block?

Decoder has three sublayers: (1) masked self-attention (each token can only attend to earlier positions — causal mask); (2) cross-attention over encoder outputs (queries from decoder, keys/values from encoder); (3) MLP. Each still wrapped in pre-norm + residual + dropout.

Encoder-only vs decoder-only vs encoder-decoder — when do you pick each?

Encoder-only (BERT, RoBERTa): bidirectional attention over the whole input → best for understanding tasks (classification, NER, retrieval embeddings). Decoder-only (GPT, LLaMA): causal attention → best for open-ended generation and general chat assistants.

Explain focal loss and why it's used in dense detection.

$\mathrm{FL}(p_t)$ = -α · $(1 - p_t)$^γ · $\log(p_t)$, where p_t is the model's probability for the true class. The $(1 - p_t)$^γ term (γ ~ 2) down-weights well-classified examples, keeping the loss focused on hard examples.

Write Dice loss and its role in segmentation.

Dice = 1 - 2 * |P ∩ G| / (| P | + | G |) — one minus the Dice coefficient (F1 on binary masks). Directly optimizes overlap between predicted and ground-truth masks.

What is perceptual loss and why is it better than pixel-wise for image quality?

Loss computed as MSE (or cosine similarity) between features from a pretrained network (e.g., VGG-19 conv layers) applied to prediction and target, rather than on raw pixels. Pretrained features capture perceptual similarity — small changes in features correspond to visually similar images.

What are the main ideas of StyleGAN?

(1) Map input latent z to an intermediate 'style' space W with an MLP — disentangles factors of variation. (2) Inject style at every layer of the generator via adaptive instance normalization (AdaIN) — controls features at multiple scales.

How does Latent Diffusion (Stable Diffusion) reduce compute?

Train a VAE / autoencoder that maps 512×512 images to an 8-16x smaller latent space (e.g., 64×64×4). Run the diffusion process in that latent space instead of pixels.

What are normalizing flows and their trade-off vs GANs / diffusion?

Sequence of invertible transformations from a simple base (N(0, I)) to the data distribution, using the change-of-variables formula for exact log-likelihood. Advantages: exact density evaluation, invertible sampling.

What is Neural Architecture Search (NAS)?

Automated search for the best architecture (layers, connections, widths) given a target task and constraints (FLOPs, latency, memory). Methods: reinforcement learning over architectures (NASNet, MnasNet), evolutionary search (AmoebaNet), gradient-based (DARTS — differentiable), or one-shot supernets (Once-For-All).

Object detection & segmentation

What are IoU / Jaccard and Tversky losses?

IoU (Jaccard) = |P ∩ G| / |P ∪ G|; loss = 1 - IoU. Similar to Dice but stricter.

Recurrent networks

Why did transformers replace RNNs for sequence modeling?

RNNs process tokens sequentially, so they don't parallelize across time and struggle with long-range dependencies due to vanishing gradients (LSTMs help but only partly). Transformers process all tokens in parallel with attention, capture arbitrary-range dependencies directly, and scale much better on GPUs/TPUs.

Explain the four gates of an LSTM.

(1) Forget gate $f = \sigma(W_f \cdot [h_{t-1}, x_t])$: what to remove from cell state. (2) Input gate i = σ(...) and candidate $c_{\text{tilde}} = \tanh()$: what to add.

GRU vs LSTM — how are they different in practice?

GRU merges forget + input gates into an update gate and drops the separate cell state (uses only h). Fewer parameters (~25% less), often trains faster, similar accuracy on most sequence tasks.

How does a bidirectional RNN work and when is it appropriate?

Run two RNNs — one left-to-right, one right-to-left — and concatenate their hidden states at each timestep. Provides access to both past and future context.

What was Bahdanau attention and why was it a big deal?

Bahdanau et al. (2014) introduced attention for seq2seq: at each decoder step, compute alignment scores between the current decoder hidden state and every encoder hidden state, softmax to get weights, take a weighted sum to form a 'context' vector, and use it in the decoder update. Removed the RNN's fixed-length bottleneck between encoder and decoder — enabled machine translation on long sentences.

How does beam search work and what is its main failure mode?

At each decoding step, keep the k highest-probability sequences ('beams') so far, expand each by one token, keep the top k of the resulting k*|V | candidates. Approximate search — better than greedy, cheaper than exhaustive.

Why is length normalization needed in beam search?

Log-probability of a sequence is the sum of log $P(\text{token}_t \mid \text{history})$ — always negative, and longer sequences pile up more negatives. Without normalization, beam search prefers shorter sequences by construction.

What is CTC loss and where is it used?

Connectionist Temporal Classification (Graves 2006): sums the probability over all valid alignments between an input sequence (frames) and an output sequence (labels) that may be shorter. Allows a 'blank' token and repeated labels.

Attention & transformers

How does self-attention work in a transformer?

For each token, produce a Query, Key, and Value vector. The output for token i is a weighted sum of all Values, where the weight is $\text{softmax}(Q_i \cdot K_j \,/\, \text{sqrt}(d_k))$.

Why does a transformer need positional encoding?

Self-attention is permutation-invariant — without position info, a sentence and its shuffle look identical. Positional encoding injects order.

Why is attention scaled by $1/\mathrm{sqrt}(d_k)$?

Assume q, k have entries with mean 0 and variance 1 (post-init or post-normalization). Their dot product has variance d_k.

Why multi-head attention instead of a single big head?

Different heads can attend to different types of relationships (syntactic, semantic, positional) in different subspaces. Multi-head with H heads of dim d/H has similar compute to one head of dim d, but the diversity of learned attention patterns empirically helps a lot.

How do you pick the head dimension d_{head}?

Common choice: $d_{\text{head}} = d / n_{\text{heads}}$ (so total FLOPs match a single-head version with hidden dim d). d_{head} too small (< 32) → heads can't represent complex relationships; too large (> 128) → wastes parameters on redundancy. Typical: $d_{\text{head}} = 64 - 128$.

Why is standard self-attention $O(n^2)$ in sequence length?

For each of n query tokens, compute a dot product with each of n key tokens → n^2 dot products, each of size d. Then a softmax over n weights per query.

How do Performer / Linformer / linear attention reduce $O(n^2)$?

Performer: replace $\exp(q \cdot k \ / \ \mathrm{sqrt}(d))$ with a positive-feature kernel φ(q)·φ(k) that lets you re-associate the matmul as $(K^T\ V)(Q^T)$ — $O(n\ \cdot\ d^2)$ instead of $O(n^2\ \cdot\ d)$. Linformer: project the sequence-length dimension of K, V from n to a small constant k → O(n * k * d).

What is sparse / sliding-window attention?

Restrict each token to attend to a fixed neighborhood (window of size w) rather than all n tokens. Complexity drops from $O(n^2)$ to O(n * w).

How do Longformer / BigBird combine sparse and global attention?

Combine local sliding-window attention (each token sees a window of size w) with a handful of 'global' tokens (e.g., [CLS], question tokens in QA) that attend to every position and are attended by every position. BigBird adds a random-attention pattern for provably close approximation of full attention.

What is Flash Attention?

Dao et al. (2022) IO-aware attention algorithm: tile Q, K, V into blocks that fit in on-chip SRAM, compute softmax incrementally with the online softmax trick, and never materialize the full n×n attention matrix in HBM. Same math as standard attention (exact, not approximate), but 2-4x faster and O(n) memory instead of $O(n^2)$.

What is the KV cache in transformer inference?

During autoregressive generation, tokens are produced one at a time. For each layer, the K and V projections of all previously generated tokens are cached; only the new token's Q, K, V are computed each step.

Explain sinusoidal positional encoding.

$\mathrm{PE}(\mathrm{pos},\ 2i)\ =\ \sin(\mathrm{pos}\ /\ 10000(2i/d))$;
$\mathrm{PE}(\mathrm{pos},\ 2i+1)\ =\ \cos(\mathrm{pos}\ /\ 10000(2i/d))$. Each dimension is a sinusoid of a different frequency.

What is the main limitation of learned absolute positional embeddings?

They can't extrapolate — at inference, positions beyond $\max_{\text{position}}$ embeddings have never been trained and produce garbage. Also, they encode absolute rather than relative positions, so translating the sentence within a longer context changes the embedding.

How does relative positional encoding (Shaw / T5) work?

Instead of adding a positional vector to the token embedding, inject position information into the attention score itself as a learned bias depending on the offset (i - j) between query and key. In T5, this bias is bucketed (log-spaced) so the number of learned parameters stays finite.

How does ALiBi encode position?

Attention with Linear Biases (Press et al., 2021): add a linear penalty proportional to distance to the attention logits: $a_{i,j}$ -= $m_h\ \cdot\ (i\ -\ j)$, where m_h is a fixed per-head slope. No learned embeddings — just a static penalty.

How does Rotary Position Embedding (RoPE) work?

Rotate the Q and K vectors in each 2D subspace of the head dimension by an angle proportional to the position: $R(\theta_{\text{pos}})\ \cdot\ q$, $R(\theta_{\text{pos}})\ \cdot\ k$, with $\theta_{\text{pos}}\ =\ \mathrm{pos}\ /\ \mathrm{base}$^(2i/d). Because of the rotation identity, $(R(\theta_i)q)\cdot(R(\theta_j)k)$ depends only on the difference $\theta_i\ -\ \theta_j$ — encodes relative position multiplicatively.

How does cross-attention differ from self-attention?

Cross-attention: queries come from one sequence (e.g., decoder tokens), keys/values from another (e.g., encoder outputs, retrieved documents, image patches). The decoder 'reads' the encoder representation without needing its own tokens to encode it.

How much can you interpret a model from attention weights?

Attention weights are a noisy signal for interpretability. High weight ≠ 'the model uses this token'.

Briefly, what do BLEU and ROUGE measure?

BLEU: n-gram precision of the generated hypothesis against one or more reference translations, with a brevity penalty for short outputs. Standard for machine translation.

What are NTK-aware RoPE scaling and YaRN?

Techniques to extend a RoPE-based LLM's context window beyond the training length without retraining from scratch. Naive linear interpolation of positions (Position Interpolation) shrinks the effective frequency and hurts short-range attention.

How do you extend a transformer to very long contexts?

Options: (1) sparse/local attention (Longformer, Mistral's sliding window); (2) linear-attention approximations (Performer, RWKV, Mamba SSM); (3) retrieval augmentation (chunk documents, retrieve top-k with a vector store, stuff into the prompt); (4) position-encoding extrapolation (RoPE scaling, NTK-aware interpolation, YaRN); (5) hierarchical models (encode chunks separately then attend at a higher level). Combine multiple techniques in practice.

Attention is quadratic in sequence length. Why is FlashAttention still a major win without changing that?

Because the practical bottleneck is memory traffic, not arithmetic. A naive implementation materializes the full attention matrix in high-bandwidth memory, so the cost is dominated by writing and reading a quadratic-sized intermediate.

Losses for deep learning

Write the Huber loss and explain when to use it.

L_δ(y,) $= 0.5 \cdot (y-)^2$ if $|y-\hat{y}| \le \delta$, else $\delta \cdot (|y-| - 0.5 \cdot \delta)$. Quadratic for small errors (smooth, MSE-like gradients), linear for large errors (robust to outliers). δ ~ 1-2*σ of the noise.

What is log-cosh loss and its advantage over Huber?

L(y, ŷ) = log(cosh(y - ŷ)) ≈ $0.5 \cdot (y-)^2$ for small errors and $|y-\hat{y}| - \log(2)$ for large — same behavior as Huber but smooth everywhere (all derivatives exist). No δ hyperparameter.

Write binary cross-entropy for probability p̂ and label y ∈ {0, 1}.

L(y, p̂) = -[y*log(p̂) + (1-y)*log(1-p̂)]. Comes from the negative log-likelihood of a Bernoulli.

Multi-class cross-entropy vs one-vs-rest — pros and cons?

Cross-entropy on softmax: mutually exclusive classes, single loss, joint probability calibration → standard for classification. One-vs-rest with K sigmoids + BCE: allows multi-label, imbalanced-per-label handling, per-class thresholding at inference.

Write the basic contrastive loss and when it's used.

$L = y \cdot d^2 + (1-y) \cdot \max(\text{margin} - d, 0)^2$, where y=1 for a positive pair (same class) and y=0 for a negative pair, d is a distance in the embedding space. Pulls positives together, pushes negatives apart at least by a margin.

What is triplet loss and its main challenge?

L(a, p, n) = max(d(a, p) - d(a, n) + margin, 0), where a=anchor, p=positive, n=negative. Wants the anchor closer to positives than to negatives by at least the margin.

Write info-NCE / NT-Xent and its role in SSL.

$L(x, x^+, x^-)$ = - $\log[\exp(\text{sim}(x, x^+) / \tau) / (\exp(\text{sim}(x, x^+) / \tau) + \Sigma \exp(\text{sim}(x, x^-) / \tau))]$ sim = cosine similarity, τ = temperature. Treats contrastive learning as a classification problem: 'which of these candidates is the positive?'.

What is the standard reconstruction loss for autoencoders?

MSE (or L1) between input and reconstruction for continuous data. Binary cross-entropy per pixel for images normalized to [0, 1] (treats each pixel as a Bernoulli).

Write the vanilla GAN's minimax objective.

$\min_G \max_D E_x[\log D(x)] + E_z[\log(1 - D(G(z)))]$. D tries to output 1 for real, 0 for fake; G tries to make D output 1 on its fakes.

What does WGAN + gradient penalty change vs vanilla GAN?

WGAN uses the Wasserstein distance approximation: $\max_D$ E[D(x)] - E[D(G(z))] where D (the 'critic') must be 1-Lipschitz. Original WGAN clipped weights (crude).

Write the VAE ELBO and explain each term.

$\text{ELBO}(x) = E_q(z \mid x)[\log p(x \mid z)] - \text{KL}(q(z \mid x) \mid\mid p(z))$. Reconstruction term: how well the decoder reproduces x from a latent sampled by the encoder.

Transfer learning & fine-tuning

What is transfer learning and when is fine-tuning better than feature extraction?

Transfer learning reuses a model pretrained on a large source task on a new target task. Feature extraction freezes the backbone and trains a new head — fast, works when the target is small and similar to pretraining data.

What is LoRA and why is it the standard for parameter-efficient fine-tuning?

LoRA (Hu et al., 2021) freezes the pretrained weights W and adds a low-rank update ΔW = B * A $(\text{with } A \in R(r \times d),\ B \in R(d \times r),\ r << d)$. Only A and B are trained → ~0.1-1% of the total parameters.

With a small labelled dataset and a large pretrained backbone, which parameters do you actually train?

Start with the smallest set that can express the task. Train only a new head first, which is fast, cannot damage the backbone, and gives a baseline within minutes.

Efficiency & distributed training

What is gradient checkpointing and its trade-off?

Store activations at only a few 'checkpoint' layers instead of every layer. During backward, recompute the missing activations from the nearest checkpoint on demand.

How does activation checkpointing interact with gradient accumulation and FSDP?

Activation checkpointing (recompute activations in backward) trades ~30% extra compute for O(√L) memory. Gradient accumulation reduces optimizer-step frequency but doesn't reduce peak activation memory.

Generative models

What is mode collapse in GANs and how do you mitigate it?

Generator maps most latents to a few output modes because those fool the discriminator well — trained distribution becomes much less diverse than the data. Symptoms: repeated outputs, poor coverage of the true distribution.

Describe the forward and reverse processes in diffusion models.

Forward (q): gradually add Gaussian noise to the data over T steps until it becomes ~ N(0, I). Fixed, no learning.

How does DDIM speed up diffusion sampling?

DDIM (Song 2020) formulates a deterministic non-Markovian reverse process that shares the same training objective as DDPM but allows sampling with far fewer steps (e.g., 50 instead of 1000) with minimal quality loss. Also enables deterministic sampling (identical noise → identical output), which is essential for tasks like image-to-image editing, inversion, and reproducibility.

What is classifier-free guidance (CFG) in diffusion?

Train the model to accept both conditional (c) and unconditional (∅, e.g., empty prompt) inputs, by dropping the condition 10-20% of the time. At sampling, form the guided prediction $\mathrm{eps}_{\mathrm{guided}} = \mathrm{eps\_}\theta(x_t,\ t,\ ) + w \cdot (\mathrm{eps}\theta(x_t,\ t,\ c) - \mathrm{eps}\theta(x_t,\ t,\ ))$. w>1 amplifies condition alignment (sharper prompt adherence) at some diversity cost.

Compression & interpretability

How does knowledge distillation work?

Train a small 'student' network to match the outputs (usually soft probabilities at high temperature T) of a large pretrained 'teacher'. $\mathrm{Loss} = \alpha \cdot \mathrm{CE}(\mathrm{student},\ \mathrm{hard}_{\mathrm{labels}}) + (1-\alpha) \cdot T^2 \cdot \mathrm{KL}(\mathrm{softmax}(\mathrm{student}/T) \| \mathrm{softmax}(\mathrm{teacher}/T))$

What are the main pruning techniques for neural networks?

Unstructured pruning: set individual weights to zero based on magnitude ($|\ w\ | < \mathrm{threshold}$) — high compression, but sparse matrices are hard to accelerate without specialized kernels. Structured pruning: remove entire filters / channels / attention heads — smaller matrices, natively fast on GPUs.

Post-training quantization vs Quantization-Aware Training — trade-offs?

Post-training (PTQ): calibrate on a small dataset then quantize weights (and optionally activations) from fp32 to int8 / int4. Fast (a few minutes), zero training overhead, some accuracy drop especially at ≤ 4 bits.

You need to halve inference cost. Do you quantize, prune, or distill?

Quantize first, because it is the cheapest to try and the best understood. Post-training quantization to 8-bit integers typically costs a fraction of a point of accuracy and needs only a small calibration set, and 4-bit weight-only quantization is now routine for language models.