Deep Learning MCQ Rapid Notes — What to Know

Use this as a high-yield cram sheet. Each node lists crisp facts, formulas, and classic MCQ traps.

I. Deep Learning Fundamentals & Context

A. Definitions & Taxonomy

- **AI ▶ ML ▶ DL ▶ (LLMs/GenAI)**: DL ⊂ ML ⊂ AI. LLMs are DL models; many LLMs are **generative**.
- **Generative AI**: Models that synthesize content (text/images/code/audio) from a learned distribution.
- **Shallow learning**: ML methods without deep stacks (e.g., linear/logistic regression, SVM, trees).
- **DL vs ML**: DL learns features automatically; shallow ML relies on hand-engineered features.

MCQ cues: "Which is a subset of which?" → **LLMs ⊂ DL ⊂ ML ⊂ AI**.

B. Core Concepts

- **Feature extraction**: DL does it end-to-end; shallow ML often requires manual features (SIFT, MFCCs, etc.).
- **Black box**: Hard to interpret internals; mitigations: saliency maps, SHAP, attention viz.
- **Scaling laws**: Performance improves predictably with more model size, data, and compute.
- **Emergent abilities**: Qualitatively new behaviors at scale (e.g., multi-step reasoning suddenly improves).

MCQ cues: "What mainly differentiates DL from classic ML?" → **Automatic representation learning**.

C. History & Milestones (high-level)

- **McCulloch–Pitts (1943)** neurons; **Hebbian rule (1949)**; **Turing Test (1950)**; **McCarthy coins "AI" (1956)**.
- **Perceptron (1957)**; **Limits (1969)**: can't solve non-linearly separable (XOR) without hidden layers.
- **Expert systems (1980s)**: rule bases + inference engines; brittle, knowledge engineering bottleneck.
- **Deep Learning era**: term in 1980s; **Hinton et al. 2006** deep belief nets kick-start modern DL.
- **Milestones**: **ImageNet (2009)**; **AlexNet (2012)** deep CNN; **AlphaGo (mid-2010s)**; **TensorFlow (2015)**; **ChatGPT (2022)**.
- **2018 Turing Award**: Bengio, Hinton, LeCun.

MCQ cues: "Perceptron's classic failure case?" → **XOR/non-linear separability**.

II. Neural Networks & Activation Functions

A. Network structure

- Stack of layers: **Input → hidden (weights + bias + activation) → output**.
- **No activations (all linear)** ⇒ equivalent to **one linear layer**.

B. Bio vs. artificial

- **Biological**: dendrites, soma, axon, spikes.
- **Artificial**: units/neurons; **weights** are parameters learned from data.

C. Activations (purpose & quick pros/cons)

- **Purpose**: Inject **non-linearity**, enabling complex mappings.
- **Step**: binary output; **not differentiable** → no gradient; rarely used.
- **Sigmoid**: (0,1); **vanishing gradients**, **not zero-centered**, slower.
- **Tanh**: (−1,1); zero-centered but still can vanish.
- **ReLU**: `max(0,x)`; fast, sparse; issues: **dying ReLU**, not zero-centered.
- **Leaky ReLU**: small negative slope; mitigates dying ReLU.
- **Swish/SiLU**: `x·sigmoid(x)`; smooth; strong in vision nets (e.g., EfficientNet).
- **Mish**: self-regularized, non-monotonic; works well in detection/NLP.
- **GELU**: used in **Transformers** (BERT/GPT); smooth, stochastic gating interpretation.

MCQ cues: “Which is zero-centered?” → **tanh** (not sigmoid/ReLU). “Which activation in GPT/BERT?” → **GELU**.

III. Training, Loss Functions, Backpropagation

A. Optimization basics

- **Gradient Descent** minimizes loss via gradients.
- **SGD (minibatch)**: noisy but efficient; batch size trades variance vs. throughput.
- **Learning rate (η)**: most critical hyperparameter; often **decay** over time.
- **Backprop**: reverse-mode auto-diff using **chain rule**; enables efficient gradient computation.
- **Jacobian**: matrix of first-order partials; used conceptually for vector-valued mappings.

MCQ cues: “Why backprop?” → Efficient gradients for all parameters in one backward pass.

B. Losses — Regression

- **MSE/L2**: penalizes large errors heavily; outlier-sensitive; assumes Gaussian noise.
- **MAE/L1**: robust to outliers; constant gradient; non-smooth at 0; assumes Laplace noise.
- **Smooth L1 (Huber)**: quadratic near 0, linear for large residuals; combats exploding gradients.

C. Losses — Classification

- **Logistic regression**: sigmoid → Bernoulli prob.
- **Cross-entropy**: measures divergence between true P and model Q; penalizes **confident wrong** predictions strongly.

- **Softmax**: multi-class probs sum to 1.
- **Softmax + Cross-Entropy**: numerically stable; equivalent to multinomial logistic regression.

MCQ cues: "Why choose cross-entropy over MSE for classification?" → **Better gradients; avoids saturation issues**.

IV. Advanced Optimization & Regularization

A. Optimizers (what distinguishes)

- **Momentum**: adds velocity to damp zig-zag, speed plateaus.
- **Nesterov (NAG)**: look-ahead gradient at the **approx. future** position.
- **Adagrad**: per-parameter LR, accumulates squared grads → **ever-decreasing LR**.
- **RMSProp**: moving avg of squared grads (no indefinite decay); stabilizes Adagrad.
- **Adam**: momentum (1st moment) + RMSProp-like scaling (2nd moment). Typical **β_1=0.9, β_2=0.999, $\varepsilon \approx$1e-8**.
- **AdamW**: **decouples weight decay** from gradient update → better regularization than L2 in Adam.
- **Nadam**: Adam with Nesterov-style momentum.

MCQ cues: "Which optimizer decouples weight decay?" → **AdamW**. "Which has vanishing LR over time?" → **Adagrad**.

B. Regularization (reduce overfitting)

- **Overfitting**: high train acc, low val acc; **Underfitting**: low both.
- **Bias-variance trade-off**: regularization reduces variance (can raise bias).
- **L2/weight decay**: shrinks weights; improves generalization.
- **L1**: sparsity; feature selection.
- **Elastic Net**: L1 + L2.
- **Early stopping**: stop when val error rises.
- **Label smoothing**: soften targets (e.g., ε=0.1) to prevent over-confidence.
- **Dropout**: randomly drop activations during **training**; **inverted dropout** scales at train time so test time is identity.
- **Variants**: Gaussian dropout, DropConnect (drop weights), variational dropout, attention/embedding dropout, DropBlock.
- **Stochastic depth**: skip entire residual blocks during training (deep ResNets).

MCQ cues: "Dropout used at test time?" → **No** (only inference with full net; scaling handled by training variant).

V. PyTorch Framework & Implementation

A. Concepts

- **Dynamic graph (define-by-run)**: graphs built as Python executes (flexible control flow).
- **Autograd**: set `requires_grad=True`; `.backward()` computes grads.
- **Freezing**: set `requires_grad=False` for layers when fine-tuning.

B. Workflow

1) **Dataset/DataLoader** → 2) **nn.Module** model → 3) **Train loop** (forward, loss, backward, step) → 4) **Eval** (no grad) → 5) **Save/Load** (`state_dict`). - **Optimizers**: `SGD`, `Adam`, `RMSprop`, `Adagrad`, `Adadelta`, `AdamW`, `LBFGS`. - **LR schedulers**: `StepLR`, `MultiStepLR`, `ExponentialLR`, `CosineAnnealingLR`, etc.

C. Loss APIs (shape gotchas)

- `nn.CrossEntropyLoss`: **expects logits** of shape `(N, C)` and **targets** as `LongTensor` class indices `(N,)`. Internally = `LogSoftmax + NLLLoss`.
- `nn.NLLLoss`: expects **log-probs**.
- `nn.BCELoss`: expects **probabilities** `(0..1)`.
- `nn.BCEWithLogitsLoss`: expects **logits**; internally applies sigmoid.
- `nn.MSELoss`**,** `nn.L1Loss`: regression losses.

MCQ cues: "Do you pass softmax before CrossEntropyLoss?" → **No** (pass raw logits).

VI. Convolutional Neural Networks (CNNs)

A. Why CNNs

- **FC layers** on images: parameter explosion, lose spatial structure, no translation invariance.
- **CNNs**: local receptive fields + **weight sharing** → fewer params + translation-equivariant features.

B. Convolution mechanics

- **Filter/kernel** slides spatially to detect patterns.
- **Output size (no padding)**: `(N − F)/stride + 1` (per dimension, assume integer result).
- **Padding**: adds borders; common "same size" pad: `(F−1)/2` when `stride=1` and odd `F`.
- **Parameter count Conv2d**: `(kH·kW·in_ch·out_ch) + out_ch` (if bias).
- **Typical block**: `Conv → Norm (optional) → Activation` (e.g., ReLU/SiLU).

C. Pooling

- **Max pooling**: emphasizes strong responses; **Avg pooling**: smooths.
- **Adaptive pooling**: specify output size directly.
- **Global average pooling**: replaces large dense layer; **0 weights**.

D. Transfer learning

- **Feature extractor**: freeze backbone; train head (fast, less data).
- **Fine-tuning**: unfreeze some/all layers with small LR; better if you have data.

MCQ cues: "Conv2d params for 3×3, in=64, out=128, bias?" → `3*3*64*128 + 128 = 73,856`.

Quick-Solve MCQ Patterns (with answers)

1) **Which activation is NOT zero-centered?** Sigmoid/ReLU. *(tanh is zero-centered)* 2) **Why Softmax+Cross-Entropy over MSE for classification?** Better gradients + numerical stability. 3) **Perceptron fails on?** XOR / non-linearly separable. 4) **Which optimizer decouples weight decay?** AdamW. 5) **Label smoothing does what?** Reduces over-confidence by assigning small mass to non-targets. 6) **Backprop relies on which calculus rule?** Chain rule. 7) **CNN advantage vs FC on images?** Far fewer params + translation equivariance. 8) **Dropout train vs test?** Active in training only (inverted scaling); off at test. 9) **CrossEntropyLoss expects?** Raw logits + integer class targets. 10) **Emergent ability definition?** New capabilities appearing abruptly as scale increases.

Micro-Formulas to Memorize

- **ReLU**: `max(0, x)`; **Leaky ReLU**: `max(αx, x)` (α≈0.01).
- **Swish/SiLU**: `x·σ(x)`; **GELU** ≈ `0.5 x (1 + tanh(√(2/π)(x + 0.044715x^3)))`.
- **Softmax**: `softmax(z_i) = e^{z_i} / Σ_j e^{z_j}`.
- **Cross-entropy (multi-class)**: `L = −Σ_i y_i log p_i`.
- **Conv output (1D)**: `⌊(N + 2P − F)/S⌋ + 1`; (apply per spatial dim for 2D/3D).

Last-Minute Drill Prompts

- Explain **why** `softmax + CE` is preferred over `sigmoid + MSE`.
- Compute conv output for `N=32, F=5, S=1, P=2` ⇒ **32**.
- Contrast **Adam vs AdamW** in one sentence.
- Two causes of **vanishing gradients** (sigmoid/tanh saturation; deep chains w/ poor init) and two fixes (ReLU/SiLU; residuals/norms; better init; LR schedules).

Exam Strategy Tips

- Watch for **shape/expectation** mismatches in PyTorch loss functions.
- When in doubt on taxonomy, think **subset ladder**: AI ⊃ ML ⊃ DL ⊃ LLM/GenAI.
- Prefer options that describe **numerical stability** and **gradient quality** for training-related questions.