CS-GY 6923 Machine Learning — Final Exam Cheat Sheet

Use on exam: write objective (ERM + reg), state dimensions, use standard gradients/identities, keep steps small; focus on understanding-style prompts.

0. Setup: supervised learning & ERM

Supervised learning pieces Data: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$. Model class f_θ. Loss $\ell(\hat{y}, y)$ (or NLL). Train algorithm finds θ.

Regularized ERM

$$\hat{R}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \lambda\Omega(\theta)$$

Split: train fit; validation tune λ/hyperparams; test report once.

Generalization (practical) Overfitting: low train loss, high test loss. Fix via: more data/augmentation, simpler model, regularization, early stopping, ensembling, better features, better optimization.

Bias–variance (safe heuristics) More flexible model ⇒ lower bias, higher variance. More data/augmentation/ensembles ⇒ lower variance. Strong regularization ⇒ higher bias, lower variance.

1. Linear regression (OLS / WLS / Ridge)

Notation $X \in \mathbb{R}^{n\times d}$ (rows $x_i^\top$), $w \in \mathbb{R}^d$, predictions $\hat{y} = Xw$ (add bias via augmented feature $[x; 1]$).

OLS (MSE)

$$\min_w \frac{1}{n}\|Xw - y\|_2^2 \quad \nabla_w = \frac{2}{n}X^\top(Xw - y)$$

Normal equations: $X^\top Xw = X^\top y$. If invertible: $w^* = (X^\top X)^{-1}X^\top y$. If not: minimum-norm solution $w^* = X^\dagger y$.

Weighted least squares (WLS) Weights $\alpha_i > 0$, $W = \mathrm{diag}(\alpha_1, \ldots, \alpha_n)$:

$$\min_w (Xw - y)^\top W(Xw - y) \quad \Rightarrow \quad w^* = (X^\top WX)^{-1}X^\top Wy$$

Ridge regression (ℓ_2)

$$\min_w \frac{1}{n}\|Xw - y\|_2^2 + \lambda\|w\|_2^2 \quad \Rightarrow \quad w^* = (X^\top X + n\lambda I)^{-1}X^\top y$$

Gradient: $\frac{2}{n}X^\top(Xw - y) + 2\lambda w$. Effect: shrinks weights, improves conditioning, reduces variance.

LASSO / Elastic Net LASSO: $\min \frac{1}{n}\|Xw - y\|^2 + \lambda\|w\|_1$ (no closed form; subgradient at 0; sparse solutions).
Elastic Net: $\lambda_1\|w\|_1 + \lambda_2\|w\|_2^2$.

Feature scaling Standardize features: $x_j \leftarrow (x_j - \mu_j)/s_j$ to make ℓ_1/ℓ_2 comparable; helps SGD and kNN; improves conditioning.

2. Feature maps & nonlinear linear models

General trick Pick $\phi : \mathbb{R}^d \to \mathbb{R}^p$, fit linear model in feature space:

$$f(x) = w^\top\phi(x) \quad \text{(still linear in } w\text{)}$$

Common $\phi(x)$

- Polynomial / interactions: add x_ix_j, x^k, etc.
- Hinge basis (piecewise linear): $(x - a)_+ = \max(0, x - a)$.
- One-hot for categories (drop one column if intercept included).
- Fourier features (random or fixed frequencies): include $\cos(2\pi kx)$, $\sin(2\pi kx)$; increasing frequency capacity reduces train error, can overfit.

Model selection Tune feature degree / number of features / λ by validation (or k-fold CV). Use test once.

3. Logistic regression & Softmax

Binary logistic regression Logit $z = w^\top x + b$, probability $p = \sigma(z) = \frac{1}{1+e^{-z}}$.
Binary cross-entropy (NLL):

$$\ell(y, p) = -y\log p - (1 - y)\log(1 - p)$$

Key derivative: $\frac{\partial\ell}{\partial z} = p - y$.
Gradients: $\nabla_w\ell = (p - y)x$, $\partial\ell/\partial b = (p - y)$.

Multiclass softmax Logits $z \in \mathbb{R}^K$, $p = \mathrm{softmax}(z)$ with

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}, \qquad \ell = -\sum_{k=1}^{K} y_k \log p_k$$

Derivative: $\frac{\partial\ell}{\partial z} = p - y$ (vector form).

Numerical stability Use $a = \max_j z_j$:

$$\log\sum_j e^{z_j} = a + \log\sum_j e^{z_j - a}$$

Metrics Accuracy. Confusion matrix. Precision/Recall/F1 for imbalanced classes. Threshold tradeoffs. ROC/AUC conceptually.

4. Convexity & optimization (exam-level)

Convexity f convex if $f(\theta x + (1 - \theta)y) \le \theta f(x) + (1 - \theta)f(y)$ for all $\theta \in [0, 1]$.
If differentiable: $f(y) \ge f(x) + \nabla f(x)^\top(y - x)$.
Strong convex: $f(y) \ge f(x) + \nabla f(x)^\top(y - x) + \frac{\mu}{2}\|y - x\|^2$.

Gradient descent (GD) Update: $w_{t+1} = w_t - \eta\nabla f(w_t)$.
If f is L-smooth (Lipschitz gradient), stable step: $\eta \le 1/L$ (typical).
Quadratic $f(w) = \frac{1}{2}\|Aw - b\|^2$: $L = \lambda_{\max}(A^\top A)$.

Stochastic GD (SGD) Minibatch gradient:

$$g = \frac{1}{|B|}\sum_{i\in B}\nabla\ell_i(w) \quad , \quad w \leftarrow w - \eta g$$

Noise helps escape some poor regions; needs LR schedule. Typical knobs: learning rate, batch size, momentum, weight decay, early stopping.

Momentum / Adam (rules of thumb) Momentum: $v \leftarrow \beta v + (1 - \beta)g$, $w \leftarrow w - \eta v$.
Adam: momentum + per-coordinate scaling (good default; watch weight decay + LR).

Early stopping Stop when val loss stops improving (patience). Acts like implicit regularization.

5. Matrix calculus: high-yield identities

Quadratic forms If $f(w) = \frac{1}{2}\|Aw - b\|_2^2$, then $\nabla f = A^\top(Aw - b)$.
If $f(w) = \frac{1}{2}w^\top Qw$ with symmetric Q, then $\nabla f = Qw$.

Linear layer backprop shapes If $y = Xw$ with $X \in \mathbb{R}^{n\times d}$, $w \in \mathbb{R}^d$, and upstream gradient $g = \frac{\partial L}{\partial y} \in \mathbb{R}^n$:

$$\frac{\partial L}{\partial w} = X^\top g, \qquad \frac{\partial L}{\partial X} = g\,w^\top$$

Softmax + cross-entropy With one-hot y: $\frac{\partial L}{\partial z} = p - y$ (memorize).

6. Neural networks (MLP) & backprop

MLP Forward: $h^{(0)} = x$,

$$a^{(\ell)} = W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}, \quad h^{(\ell)} = \phi(a^{(\ell)})$$

Output layer: regression (linear), or classification (softmax logits).

Backprop core Chain rule with Jacobian-vector products (avoid explicit Jacobians).
If upstream gradient is $g^{(\ell)} = \frac{\partial L}{\partial a^{(\ell)}}$:

$$\frac{\partial L}{\partial W^{(\ell)}} = g^{(\ell)}(h^{(\ell-1)})^\top, \quad \frac{\partial L}{\partial b^{(\ell)}} = g^{(\ell)}, \quad \frac{\partial L}{\partial h^{(\ell-1)}} = (W^{(\ell)})^\top g^{(\ell)}$$

And $g^{(\ell)} = \frac{\partial L}{\partial h^{(\ell)}} \odot \phi'(a^{(\ell)})$.

Activations + derivatives

- Sigmoid: $\sigma'(u) = \sigma(u)(1 - \sigma(u))$ (vanishing gradients risk).
- Tanh: $1 - \tanh^2(u)$.
- ReLU: $\max(0, u)$, derivative $1_{u>0}$.
- GELU/SiLU: smooth; good defaults in transformers.

Depth issues Vanishing/exploding gradients. Fix via: good init, normalization, residual connections, careful LR.

Initialization Xavier/Glorot (tanh-ish), He (ReLU-ish). Goal: preserve variance across layers.

NN regularization Weight decay, dropout, data augmentation, label smoothing (classification), early stopping.

Normalization BatchNorm (uses batch stats; train vs eval behavior). LayerNorm (per token/per example; standard in transformers).

Residual connection $h \leftarrow h + F(h)$ improves optimization, enables deep nets (ResNets, Transformers).

7. Convolution & CNNs

2D convolution (idea) Shared kernel slides over spatial locations (translation equivariance), parameter-efficient vs fully connected for images.

Output size (single dimension) Input size N, kernel K, padding P, stride S:

$$N_{\text{out}} = \left\lfloor \frac{N + 2P - K}{S} \right\rfloor + 1$$

2D applies per height/width. Channels: filters produce output channels.

Parameters For $C_{\text{in}} \to C_{\text{out}}$ conv with $K \times K$: params $= C_{\text{out}} \cdot (C_{\text{in}}K^2 + 1$ (bias)).

Pooling Max/avg pooling reduces spatial dims, increases invariance.

1x1 conv Mix channels, cheap; used in bottlenecks.

ResNet block $h \leftarrow h + F(h)$ (skip), often: Conv-BN-ReLU stacks. Benefits: optimization + depth.

Convolution beyond images 1D conv for sequences/time series; 3D conv for video/volumes.

8. Interpretability (practical)

Gradient-based Saliency:

$\nabla_x \text{score}(x)$; highlights sensitive pixels/tokens.
SmoothGrad: average gradients under noise.
Integrated Gradients: integrate ∇_x along path from baseline to input.

Class activation Grad-CAM: combine feature maps using gradients to get coarse spatial importance.

Perturbation Occlusion / masking: change input regions and measure output change.

Caveats Saturation, gradient shattering, baseline choice, correlation vs causation.

9. Sequence modeling (RNNs) & language modeling

Tokenization Map text to token IDs. Embedding: token → vector. Unembedding: vector → logits over vocab.

RNN Hidden state:

$$h_t = \phi(W_h h_{t-1} + W_x x_t + b)$$

Output logits from h_t; train with teacher forcing (predict next token).
Loss: sum of cross-entropies across time. Backprop through time (BPTT).
Issues: vanishing/exploding gradients; fix: gating (LSTM/GRU), gradient clipping.

LSTM (gate sketch) Forget/input/output gates control information flow; helps long dependencies.

Autoregressive LM Factorization:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

Train via NLL; generate via sampling from predictive distribution.

10. Transformers & post-training

Self-attention (single head) Given token representations $H \in \mathbb{R}^{T\times d}$:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V$$

$$\text{Attn}(H) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

M is mask (causal: $-\infty$ above diagonal). Output is $T \times d_v$ then projected back.

Multi-head Split into heads, apply attention per head, concatenate, project.

Transformer block Residual + norm + MLP:

$$H \leftarrow H + \text{Attn}(\text{LN}(H)), \quad H \leftarrow H + \text{MLP}(\text{LN}(H))$$

Positional information Add positional embeddings or use rotary/relative schemes (concept: break permutation symmetry).

Complexity Attention is $O(T^2 d)$ in sequence length T.

Generation controls Temperature τ on logits (z/τ), top-k, nucleus (top-p). Greedy vs sampling.

Post-training (concepts) Instruction tuning (supervised), preference optimization/RLHF (align outputs), safety filters. Goal: change behavior without rewriting pretraining.

11. Decision trees & ensembles

Decision tree split Pick feature/threshold to maximize impurity reduction.
Classification impurity: Gini $1 - \sum_k p_k^2$ or entropy $-\sum_k p_k \log p_k$.
Regression impurity: variance / MSE reduction.

Overfitting Deep tree: low bias, high variance. Control via max depth, min samples leaf, pruning.

Bagging Train models on bootstrap samples; average predictions. Reduces variance.

Random forest Bagging + feature subsampling at splits; strong baseline for tabular.

Boosting (gradient boosting) Add weak learners sequentially to reduce error (often trees). For squared loss: fit residuals.
General view: fit learner to (approx) negative gradient of loss.

Boosting knobs (memorize)
- # trees (often 100–500),
- max depth (often 3–8),
- learning rate (e.g., 0.01–0.3),
- subsampling/regularization to prevent overfit.

Ensembles & bias–variance Bagging/averaging reduces variance. Boosting tends to reduce bias (can overfit if too aggressive). Deep ensembles: train multiple NNs from different inits; average improves accuracy + uncertainty.

12. Uncertainty & calibration

Two types Aleatoric: inherent noise in data. Epistemic: model uncertainty (lack of data / multiple plausible models).

Regression likelihood view Gaussian:

$$y \mid x \sim \mathcal{N}(\mu(x), \sigma(x)^2)$$

NLL per point:

$$\ell = \frac{1}{2}\left(\frac{(y-\mu)^2}{\sigma^2} + \log \sigma^2\right) + \text{const}$$

If σ fixed, maximizing likelihood ⇔ minimizing MSE (up to scale). If $\sigma(x)$ predicted, include $\log \sigma^2$ term.

Classification calibration Model confidence should match empirical accuracy. Tools: reliability diagram, ECE concept.
Temperature scaling: replace logits z by z/T (tune T on validation).

Uncertainty methods Deep ensembles, MC dropout (approx), Bayesian-ish viewpoints. kNN distance as a crude uncertainty signal (far from data ⇒ less confident).

13. kNN & dimensionality reduction

kNN Given distance (often Euclidean on standardized features).
Classification: majority vote among k neighbors. Regression: average (or weighted by distance).
Bias–variance knob: small k low bias/high variance; large k high bias/low variance.
Curse of dimensionality: distances concentrate; need scaling/feature selection/dim reduction.

PCA (principal components) Center data: $X_c = X - \mathbf{1}\mu^\top$. Covariance $S = \frac{1}{n} X_c^\top X_c$.
Top components are eigenvectors of S. Using SVD: $X_c = U\Sigma V^\top$, PCs are columns of V.
Projection to r dims: $Z = X_c V_r$. Reconstruction: $\hat{X} = ZV_r^\top + \mu$.
Explained variance ratio: $\Sigma_{1:r}^2 / \sum_j \Sigma_j^2$.

When PCA helps Noise reduction, visualization, conditioning, faster models, mitigate multicollinearity.

14. Homework & midterm patterns (what repeats)

Hyperparameter tuning (SGD / regularization) State which hyperparameters you tune and why: learning rate η, batch size B, epochs, momentum/Adam, weight decay λ, early stopping. Use validation loss; do not touch test until final.

Sparse vs dense features If you expect only a few relevant features: ℓ_1 (LASSO) encourages sparsity/feature selection. If many weak features: ℓ_2 stabilizes.

Feature engineering for linear models Interactions, polynomials, hinges for piecewise linear structure; standardize; include bias; encode categories; consider log transforms for heavy-tailed targets.

Fourier features intuition Increasing number/frequency of features increases capacity: training error drops; test error may show U-shape (overfit). Regularize / select k by validation.

Conv as linear layer viewpoint Convolution is a linear operator with parameter sharing and sparse connectivity; helpful for reasoning about shapes, parameter counts, and gradients.

Transformer-as-algorithm (attention can route info) Use attention weights to copy/aggregate specific tokens based on content + position; multi-layer can implement simple procedures (e.g., select a token, compute aggregate, write to output).

15. Exam checklist (fast execution)

- Write objective: $\frac{1}{n}\sum \ell + \lambda\Omega$; define variables; list shapes.
- For linear/quad: take gradient, set to zero; use $X^\top X$ forms.
- For logistic/softmax: use $\partial L/\partial z = p - y$.
- For SGD questions: talk stability (LR), noise (batch), and validation tuning; mention early stopping.
- For overfit/underfit: propose *one* concrete fix per failure mode (reg/data/architecture).
- For NN/CNN/Transformer: report tensor shapes, output sizes, parameter counts, and main invariances.

16. Linear algebra facts that unlock many exam questions

Projection view of least squares Let $\mathcal{C} = \text{Col}(X)$. OLS prediction $\hat{y} = Xw^*$ is the orthogonal projection of y onto $\mathcal{C}$:

$$\hat{y} = P_{\mathcal{C}} y, \quad r = y - \hat{y} \perp \mathcal{C}, \quad X^\top r = 0$$

Hat matrix (OLS): $H = X(X^\top X)^{-1}X^\top$, so $\hat{y} = Hy$, $r = (I - H)y$. (Useful: many identities become projection geometry.)

LOOCV shortcut (very high-yield) For OLS, leaving out point i:

$$\hat{y}_i^{(-i)} = \hat{y}_i - \frac{r_i}{1 - h_{ii}}, \qquad \text{LOOCV residual} = \frac{r_i}{1 - h_{ii}}$$

So

$$\text{LOOCV MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{r_i}{1 - h_{ii}}\right)^2.$$

Ridge has analogous hat matrix $H_\lambda = X(X^\top X + n\lambda I)^{-1}X^\top$ (same form; use $h_{ii}^{(\lambda)}$).

Conditioning + ridge If $X^\top X$ has eigenvalues $\{\sigma_j^2\}$, then ridge uses $\sigma_j^2 + n\lambda$ (improves smallest eigenvalues ⇒ better conditioning).

17. Probabilistic viewpoint (MAP = regularization)

Gaussian regression ↔ MSE If $y|x \sim \mathcal{N}(w^\top x, \sigma^2)$ (fixed σ), maximizing likelihood ⇔ minimizing $\sum(y_i - w^\top x_i)^2$.

Ridge as MAP Prior $w \sim \mathcal{N}(0, \tau^2 I)$ gives MAP objective:

$$\min_w \frac{1}{2\sigma^2}\|Xw-y\|^2+\frac{1}{2\tau^2}\|w\|^2 \quad \Rightarrow \quad \lambda \propto \sigma^2/\tau^2$$

Interpretation: ridge encodes belief that large weights are unlikely.

LASSO as MAP Laplace prior $p(w_j) \propto e^{-|w_j|/b}$ yields ℓ_1 penalty $\propto \|w\|_1$ (sparsity prior).

18. LASSO / ℓ_1 mechanics (proximity = sparsity)

Soft-thresholding operator (memorize)

$$\text{prox}_{\lambda\|\cdot\|_1}(v) = \text{sign}(v)\odot\max(|v|-\lambda,0)$$

This is the "reason" ℓ_1 creates exact zeros.

Coordinate descent update (standard form) For standardized columns x_j and squared loss + ℓ_1:

$$w_j \leftarrow \frac{1}{\frac{1}{n}\|x_j\|^2}\,\mathcal{S}\Big(\frac{1}{n}x_j^\top(y-X_{-j}w_{-j}),\ \lambda\Big), \quad \mathcal{S}(a,\lambda)=\text{sign}(a)\max(|a|-\lambda,0)$$

19. Logistic regression second-order facts (Newton/IRLS)

Vectorized gradient + Hessian Binary logistic with $p=\sigma(Xw+b)$:

$$\nabla_w L = X^\top(p-y), \qquad \nabla_w^2 L = X^\top W X, \quad W=\text{diag}(p_i(1-p_i))$$

Newton step / IRLS

$$w \leftarrow w-(X^\top W X)^{-1}X^\top(p-y)$$

Interpretation: locally weighted least squares; converges fast near optimum; needs regularization if ill-conditioned.

20. Classification losses and useful derivatives

Cross-entropy and label smoothing For one-hot y and prediction p:

$$\text{CE}(y,p) = -\sum_k y_k \log p_k$$

Label smoothing: $y' = (1-\alpha)y + \alpha\frac{1}{K}\mathbf{1}$ (reduces overconfidence).

Brier score (calibration-friendly)

$$\text{Brier} = \frac{1}{n}\sum_{i=1}^n\sum_{k=1}^K (p_{ik}-y_{ik})^2$$

Lower is better; sensitive to probability quality.

21. Calibration + uncertainty formulas worth writing

ECE (Expected Calibration Error) definition Bin predictions by confidence into bins B_m:

$$\text{ECE} = \sum_{m=1}^M \frac{|B_m|}{n}\,|\text{acc}(B_m)-\text{conf}(B_m)|$$

where $\text{conf}(B_m)$ is average predicted confidence and $\text{acc}(B_m)$ is empirical accuracy in the bin.

Ensemble predictive mean/variance (decomposition) Models $m = 1..M$ output mean $\mu_m(x)$ and variance $\sigma_m^2(x)$:

$$\bar\mu = \frac{1}{M}\sum_m \mu_m, \qquad \text{Var}(y|x) \approx \underbrace{\frac{1}{M}\sum_m \sigma_m^2}_{\text{aleatoric}} + \underbrace{\frac{1}{M}\sum_m(\mu_m-\bar\mu)^2}_{\text{epistemic}}$$

22. BatchNorm / LayerNorm / Dropout (precise statements)

Inverted dropout (the version used in practice)

Keep prob q; during training: $h \leftarrow (m\odot h)/q$ with $m\sim\text{Bernoulli}(q)$. At test time: do nothing (expectation already matched by $1/q$ scaling).

BatchNorm (per-feature over batch) For feature x in batch: $\hat x = \frac{x-\mu_B}{\sqrt{\sigma_B^2+}}$, output $\gamma\hat x+\beta$. Train uses batch stats (μ_B,σ_B); eval uses running averages.

LayerNorm (per-example over features) Normalize each token/example across its feature dimensions; no dependence on batch; standard in transformers.

23. Transformers: shapes + parameter counts (exam-grade)

Shapes Sequence length T, model dim d. $H\in\mathbb{R}^{T\times d}$. With h heads, $d_k = d/h$.

$$Q=HW_Q,\ K=HW_K,\ V=HW_V \quad (W_*\in\mathbb{R}^{d\times d})$$

Attention weights: $\text{softmax}(QK^\top/\sqrt{d_k}+M)\in\mathbb{R}^{T\times T}$. Output: $AV\in\mathbb{R}^{T\times d}$ (after head concat + output projection).

Typical parameter count per block (rule-of-thumb) Projections: $W_Q,W_K,W_V,W_O \Rightarrow$ about $4d^2$ params (bias ignored). MLP with hidden d_{ff}: about $2dd_{ff}$ params. Common choice $d_{ff}\approx 4d \Rightarrow$ MLP $\approx 8d^2$. Total block $\approx 12d^2$ (rough scale).

24. Boosting: one-line derivations that answer most prompts

Gradient boosting principle Fit h_m to negative gradient of loss:

$$r_i^{(m)} = -\left.\frac{\partial \ell(y_i,F(x_i))}{\partial F}\right|_{F=F_{m-1}}, \quad h_m \approx \arg\min_h \sum_i \big(r_i^{(m)}-h(x_i)\big)^2$$

Update: $F_m = F_{m-1}+\nu h_m$.

Squared loss $\ell=\frac{1}{2}(y-F)^2 \Rightarrow r = y-F$ (residuals).

Logistic / Bernoulli deviance (binary) With $p=\sigma(F)$ and $y\in\{0,1\}$:

$$\ell = -y\log p-(1-y)\log(1-p) \Rightarrow r = y-p$$

(Write this and you answer most "why boosting works" questions.)

Random forests (one high-yield fact) Out-of-bag (OOB) estimate: each tree predicts points not in its bootstrap sample; average OOB error approximates test error without separate validation.

25. kNN refinements that score easy points

Distance-weighted kNN Weights $w_i \propto \frac{1}{d(x,x_i)+}$ or $\exp(-d^2/\tau)$:

$$\hat y(x) = \frac{\sum_{i\in\mathcal{N}_k(x)} w_i y_i}{\sum_{i\in\mathcal{N}_k(x)} w_i}$$

Density / uncertainty heuristic Large nearest-neighbor distances $\Rightarrow$ lower density $\Rightarrow$ higher uncertainty / lower confidence.

26. PCA: statements + minimal proofs you can cite

Optimality statement (reconstruction error) Among all rank-r linear projections, PCA minimizes squared reconstruction error:

$$\min_{\text{rank}(P)=r}\|X_c - X_cP\|_F^2$$

Solution: project onto top-r eigenvectors of covariance (or top-r right singular vectors).

Reconstruction error equals discarded variance If eigenvalues of covariance are $\lambda_1\ge\cdots\ge\lambda_d$:

$$\text{Min reconstruction error} \propto \sum_{j=r+1}^d \lambda_j$$

Interpretation: keeping top components keeps maximum variance; discarded eigenvalues quantify what you lose.

Whitening (if asked) After PCA: $Z=X_cV_r$; whiten by $Z_w = Z\Sigma_r^{-1}$ (unit covariance), sensitive to noise.

27. Small, high-value inequalities / facts

Jensen (one-liner) For convex ϕ: $\phi(\mathbb{E}[X]) \le \mathbb{E}[\phi(X)]$ (useful for "why log-sum-exp is convex" style prompts).

Log-sum-exp is convex

$$\text{LSE}(z) = \log\sum_j e^{z_j} \text{ is convex}; \quad \nabla\text{LSE}(z) = \text{softmax}(z)$$

(Quick proof uses Hessian = covariance of softmax distribution, PSD.)

Lecture addendum (high-yield from slides L1–L12)

L1 — What ML is; problem types; pipeline; generalization

- ML sketch: dataset → choose model class f_θ → train θ by minimizing loss → deploy on new data.
- Key challenges: **data** quality/coverage, **model** choice, **training** (optimization), **generalization** (train → test).
- Supervised tasks: classification (discrete), regression (continuous), segmentation (structured output).
- Other learning: unsupervised (no labels), self-supervised (create labels from data), RL/post-training (optimize behavior with feedback).
- Practical reminder: always separate train/val/test; tune on val; report once on test.

L2 — Multiple linear regression; encoding; transformed linear models; model selection

- Encode data as a numerical matrix X; handle categorical vars with **one-hot encoding** (avoid fake linear order).
- Transformed linear models: pick features $\phi_j(x)$ (still linear in weights): $\hat y = \sum_j w_j\phi_j(x)$.
- Examples of ϕ_j: polynomials (x,x^2,x^3), interactions, $\sin(\omega x),\cos(\omega x)$ (periodic/time-series), $1/x$ (reciprocal effects).
- Multivariate polynomial features grow fast $\Rightarrow$ model selection is mandatory (degree / number of features).
- Model selection: test/validation curves typically show **U-shape** vs capacity (underfit → sweet spot → overfit).

L3 — Regularization: why it works; geometry; sparsity; smoothing priors

- Regularization trades fit vs simplicity: $\min \hat R(w)+\lambda\Omega(w)$; larger $\lambda \Rightarrow$ simpler/less variance; too large $\Rightarrow$ underfit.

- Equivalent constrained view: $\min \hat{R}(w)$ s.t. $\Omega(w) \leq c$.
- Geometry: ℓ_2 ball is round $\Rightarrow$ shrink all coords; ℓ_1 ball has corners $\Rightarrow$ **sparse** solutions (feature selection).
- Weight decay (deep nets) is ℓ_2-style regularization.
- Smoothness regularization idea (ordered coefficients): penalize differences, e.g. $\sum_i (w_{i+1} - w_i)^2$ to prefer smooth parameter profiles.

L4 — Optimization for ERM: GD/SGD; nonconvexity; implicit bias

- Generic ERM is an optimization problem; key design choices: initialization, step-size/LR schedule, minibatch size, stopping criterion.
- Full-batch GD vs SGD: SGD uses noisy gradients; cheaper per step; needs LR schedule; often helps in large-scale training.
- Deep nets are **non-convex**; still succeed in practice; many distinct low-train-loss solutions (overparameterization).
- Implicit regularization: optimization choice (SGD, early stopping, etc.) can bias which solution you end up with among many.

L5 — Neural nets as feature learning; capacity; universal approximation intuition

- Key shift: linear models need hand-designed $\phi(x)$; neural nets **learn features** jointly with predictor.
- Two-layer MLP parameter count (input D, hidden H, output C): $\approx DH + H + HC + C$.
- Universal Approximation (intuition): with enough hidden units, can build localized "bumps" and sum them to approximate complex functions.
- Practical: capacity rises with width/depth $\Rightarrow$ use regularization + validation to control generalization.

L6 — Training deep nets: backprop, dropout, weight decay, vanishing/exploding

- Backprop is repeated chain rule through layers; always track shapes; for linear layer $y = Wh + b$: $\partial L/\partial W = g\, h^\top$.
- Dropout: randomly mask a fraction of activations during training; reduces co-adaptation; tune dropout probability.
- Weight decay: shrink weights; slide note: **equivalent to ℓ_2 for SGD but not exactly for Adam**.
- Exploding/vanishing activations/gradients: deep compositions can blow up or die; mitigations: good init, normalization, residuals, careful LR.

L7 — Convolutions: why; how; pooling; multi-channel filters

- Convolution = linear layer with **weight sharing** and locality; far fewer params than fully-connected on images.
- Multiple filters $\Rightarrow$ multiple output channels; multi-channel conv mixes channels via filter tensor.
- Pooling (e.g., maxpool): slide over patches, take max; adds invariance + downsampling.
- Stride/padding change output size; sanity-check shapes at every layer in CNN questions.

L8 — Interpretability + "beyond image classification" (segmentation/graphs)

- CNN hierarchy: early filters detect simple patterns (edges/textures); later layers compose into higher-level features.
- Interpretability tool: **feature visualization / activation maximization** — optimize input to maximize neuron activation.
- Filter size tradeoff: larger kernels (e.g., 7×7, 11×11) capture broader spatial context earlier (costlier).
- Beyond classification examples referenced: segmentation systems; graph neural nets (graph convolutions).
- Distill "circuits/early vision" theme: early-layer patterns can be interpretable as small computational motifs.

L9 — Sequence modeling with RNNs: tokens, embeddings, next-token loss

- Sequence modeling objective: given previous tokens, predict the next token.
- Represent text as token IDs; one-hot is conceptually simple but inefficient; use **embedding** lookup.
- RNN keeps hidden state h_t; outputs logits each step; apply softmax to get next-token distribution.
- Training loss: sum of cross-entropies over positions; labels are "next token" (shifted sequence).
- Practical issues: long-range dependencies; gradient stability; (standard fixes: gating, clipping, better architectures).

L10 — Transformers: self-attention, LayerNorm, residual blocks, decoder-only LM

- Self-attention = content-based lookup: compare **query** to **keys**, mix **values** by similarity weights.
- Core form: $A = \text{softmax}(QK^\top/\sqrt{d_k} + \text{mask})$, output AV.
- Decoder-only LM: causal mask prevents attending to future tokens; used for next-token prediction.
- Transformer block pattern: (pre-)LayerNorm + attention + residual; (pre-)LayerNorm + MLP + residual.
- Unembedding: map hidden state to vocab logits; softmax gives token probabilities.

L11 — Decision trees + ensembles: splitting, overfitting, bagging/forests/boosting

- Tree induction: recursively split feature/threshold to reduce loss/impurity; leaves store prediction (class majority or mean).
- Classification impurities: Gini $1 - \sum_k p_k^2$; entropy $-\sum_k p_k \log p_k$.
- Regression trees: choose splits to reduce MSE/variance; leaf predicts mean.
- Overfitting: deep trees fit noise; control with max depth/min leaf size/pruning.
- Ensembles: bagging reduces variance; random forests add feature subsampling; boosting fits residuals/gradients stagewise.

L12 — Uncertainty: epistemic vs aleatoric; ensembles; calibration (ECE); kNN

- Uncertainty matters for decisions; separate **aleatoric** (data noise) vs **epistemic** (model uncertainty).
- Regression uncertainty via likelihood: predict $(\mu(x), \sigma(x)^2)$ and use NLL $\ell = \frac{(y-\mu)^2}{2\sigma^2} + \frac{1}{2}\log\sigma^2 + \text{const}$.
- Deep ensembles: train multiple models; disagreement gives epistemic signal; averaging improves accuracy and calibration.
- Calibration metric referenced: Expected Calibration Error (ECE).
- kNN viewpoint: uncertainty higher when test point is far from training points (neighbor distance / density).