AI Research Labs

70 cards · 12 categories

Cheatsheets

Every formula, metric and rule from the course, grouped by category for revision.

Formulas (11)

Activation functions (3)

Losses and output layers (5)

Backpropagation (8)

Training and optimisation (5)

Generalisation and data (5)

Regularisation (4)

Convolutional networks (5)

PyTorch (4)

Core concepts (4)

Comparisons (8)

Common traps (8)



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/dl

Deep Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

Formulas

The equations to know by heart, with what each symbol means.

Gradient descent

$x_{k+1} = x_k - \eta \nabla f(x_k)$. Momentum: $+ \alpha \Delta x_k$ with $\Delta x_k = x_k - x_{k-1}$.

1.3 Gradient Descent and Momentum

Expectation and variance

$\mathbb{E}[X] = \sum x_i p(x_i)$; $\mathrm{Var}(X) = \mathbb{E}[(X-\mu)^2]$. Std dev $= \sqrt{\mathrm{Var}}$.

1.4 Notation, Probability and Jargon

The neuron

$y = f(w^\top x + b)$. Pre-activation $z = w^\top x + b$; bias = weight on a constant input 1.

2.1 The Artificial Neuron

Decision boundary

$w^\top x + b = 0$: a line (2-D), plane (3-D) or hyperplane. w is normal to it and points to the positive side.

2.2 Decision Boundaries and Linear Separability

Layer and parameter count

$y^{(k)} = f_k(W_k y^{(k-1)} + b_k)$. n inputs, m neurons: W is $m \times n$, $mn + m$ parameters.

3.1 Layers, Matrices and Shapes

Universal approximation

Continuous h on compact K, any $\varepsilon > 0$: a one-hidden-layer network with enough units has $\sup_{x \in K} |h(x) - \mathrm{NN}(x)| < \varepsilon$. Non-polynomial activations (incl. ReLU).

5.1 The Universal Approximation Theorem

MSE gradients for a line

$J = \frac{1}{N}\sum (mx + c - y)^2$; $\partial_m J = -\frac{2}{N}\sum x(y - mx - c)$; $\partial_c J = -\frac{2}{N}\sum (y - mx - c)$.

6.1 Training as Minimising a Loss

Softmax and cross-entropy

$o_j = e^{s_j} / \sum_k e^{s_k}$; $H = -\sum_j z_j \log o_j$; gradient $\partial H / \partial s_j = o_j - z_j$.

6.3 Softmax and Cross-entropy

Backward recursion

$\delta^{(k)} = D_k W_{k+1}^\top \delta^{(k+1)}$, $D_k = \mathrm{diag}(f'(z^{(k)}))$.

7.5 Multiple Paths and Deeper Networks

Bias–variance

$\mathbb{E}[(\hat{y} - y)^2] = \mathrm{bias}^2 + \mathrm{variance} + \sigma^2$. Capacity ↑: bias ↓, variance ↑.

8.3 Underfitting, Overfitting and Learning Curves

Conv layer parameters

$k \cdot k \cdot C_{\text{in}} \cdot C_{\text{out}} + C_{\text{out}}$ (weights + one bias per filter).

10.1 From Fully Connected to Convolutional

Activation functions

Each activation, its derivative and when to use it.

Sigmoid

$\sigma(z) = 1/(1+e^{-z}) \in (0,1)$; $\sigma' = \sigma(1-\sigma) \le 0.25$ (at $z=0$). Saturates for large $|z|$.

4.1 Sigmoid, tanh, ReLU and Friends

tanh and ReLU

tanh $\in (-1,1)$, $\tanh' = 1 - \tanh^2 \le 1$. ReLU $= \max(0, z)$, derivative 0 or 1.

4.1 Sigmoid, tanh, ReLU and Friends

Leaky ReLU

$f(z) = z$ for $z \ge 0$, αz for $z < 0$ ($\alpha \approx 0.01$). Keeps a gradient on the negative side; fixes dying ReLU.

4.3 Dying ReLU, Leaky ReLU and ELU

Losses and output layers

Which output layer and loss fit which task.

Output activation by task

Regression: linear. Binary: one sigmoid. Multi-class: softmax. Hidden layers: ReLU (deep) or sigmoid/tanh (shallow).

4.4 Choosing Activation Functions

MSE vs MAE

MSE: estimates the mean; gradient $2e$, outliers dominate. MAE: estimates the median; gradient ± 1, robust; subgradient at 0.

6.2 MSE, MAE and Asymmetric Losses

Pinball loss

$r = \hat{y} - y$: αr if $r \geq 0$, $-\beta r$ if $r < 0$. Different costs for over- and under-prediction.

6.2 MSE, MAE and Asymmetric Losses

Stable softmax

Subtract $\max_k s_k$ from every logit before exponentiating. Same probabilities, no overflow.

6.3 Softmax and Cross-entropy

Task → output → loss

Regression: linear + MSE. Binary: sigmoid + BCE (BCEWithLogitsLoss). Multi-class: softmax + CE (CrossEntropyLoss).

6.4 Binary, Multi-class and Logits in PyTorch

Backpropagation

The chain rule, error signals and the gradient of every weight.

Vanishing gradients

Backprop multiplies one f' per layer. Sigmoid $f' \leq 0.25$, so 10 layers give $\leq 0.25^{10} \approx 10^{-6}$. ReLU ($f' = 1$ when active) avoids the shrinkage.

4.2 Vanishing Gradients

Chain rule on graphs

Multiply local derivatives along a path; add contributions over all paths from a node to the loss.

7.1 The Chain Rule and Computational Graphs

Single sigmoid neuron (MSE)

$\delta_i = \frac{2}{N}(\hat{y}_i - y_i)\hat{y}_i(1-\hat{y}_i)$; $\partial E/\partial w = X^\top \delta$; $\partial E/\partial b = \sum_i \delta_i$.

7.2 Backpropagation for a Single Sigmoid Neuron

One hidden layer (sigmoid hidden, linear out, ½ squared error)

$\delta^{(2)} = y - t$; $\partial w_j^{(2)} = \delta^{(2)} a_j$; $\delta_j^{(1)} = \delta^{(2)} w_j^{(2)} a_j(1-a_j)$; $\partial w_{ji}^{(1)} = \delta_j^{(1)} x_i$.

7.3 One Hidden Layer: Deriving Every Gradient

The pattern

Weight gradient = (error signal at its output end) × (activation at its input end). Bias gradient = error signal.

7.3 One Hidden Layer: Deriving Every Gradient

Matrix backprop (4 steps)

$\delta^{(2)}$ from the loss; $\partial W^{(2)} = H^\top \delta^{(2)}$; $\delta^{(1)} = (\delta^{(2)} W^{(2)\top}) \odot f'(S^{(1)})$; $\partial W^{(1)} = X^\top \delta^{(1)}$. Biases: column sums.

7.4 Backpropagation in Matrix Form

Output error by loss

Sigmoid+MSE: $\frac{2}{N}(\hat{Y} - Y) \odot \hat{Y} \odot (1 - \hat{Y})$. Linear+MSE: $\frac{2}{N}(\hat{Y} - Y)$. Softmax+CE: $(O - Z)/N$.

7.4 Backpropagation in Matrix Form

Fan-out

A node feeding several downstream nodes: its gradient is the sum over all paths. In matrix form, $W^\top \delta$ does the sum.

7.5 Multiple Paths and Deeper Networks

Training and optimisation

Gradient descent, learning rates, batches and checks.

Learning rate

Too small: slow. Too large: oscillates or diverges. Zero gradient ≠ minimum (could be a maximum, saddle or plateau).

1.3 Gradient Descent and Momentum

Online, batch, mini-batch

Online: update per example (noisy). Batch: per full dataset (stable, slow). Mini-batch: small groups, the usual choice. Epoch = one pass.

7.6 Mini-batches and Gradient Checking

Gradient check

$\partial E/\partial \theta_k \approx [E(\theta + \epsilon e_k) - E(\theta - \epsilon e_k)]/2\epsilon$, $\epsilon \approx 10^{-6}$. A debug test, 2 evaluations per parameter.

7.6 Mini-batches and Gradient Checking

Training recipe

Split + normalise → output/loss → overfit 100 examples → train, log curves → underfit? bigger/longer → overfit? regularise → test once.

8.5 Hyperparameters and a Training Recipe

Learning-rate diagnosis

Slow decrease: η too small. Oscillating or diverging cost: η too large. Search η and λ on a log scale.

8.5 Hyperparameters and a Training Recipe

Generalisation and data

Splits, leakage, learning curves and data preparation.

Generalisation

Training loss is a proxy. The goal is low error on unseen data from the same distribution; measure it on held-out data.

8.1 Fitting Is Not Learning

Three sets

Train: fit weights. Validation: every other choice, detect overfitting. Test: once, at the end. Typical 80/10/10 or 60/20/20.

8.2 Training, Validation and Test Sets

Diagnose

Both errors high: underfitting. Train low, validation much higher: overfitting (the gap). Watch learning curves every epoch.

8.3 Underfitting, Overfitting and Learning Curves

Scaling

Min–max $(x - \min)/(\max - \min)$; standardise $(x - \mu)/\sigma$. Statistics from training data only. Pixels ÷ 255.

8.4 Preparing the Data

Data checks

Missing: drop or impute + indicator. Categories: one-hot. Labels: inspect. Imbalance: confusion matrix, precision/recall, re-weight.

8.4 Preparing the Data

Regularisation

Weight decay, early stopping, dropout and augmentation.

L2 weight decay

$J + \frac{\lambda}{2}\sum w^2 \to W \leftarrow (1-\eta\lambda)W - \eta\,\partial J/\partial W$. Shrinks weights, none exactly 0. Biases not penalised.

9.1 L2 and L1 Penalties

Early stopping

Track validation cost each epoch, save the best weights, stop after "patience" epochs without improvement, restore the best. ≈ L2 for linear models.

9.2 Early Stopping

Inverted dropout

Train: drop each unit with prob. p, $\tilde{h} = \frac{1}{1-p} m \odot h$, new mask per batch. Test: no dropping, no scaling.

9.3 Dropout

Augmentation

Label-preserving transforms, random per mini-batch, training only. Images: shift, rotate, crop, brightness, flip (not digits/text). Audio: shift, noise.

9.4 Data Augmentation

Convolutional networks

Convolutions, pooling and parameter counts.

Why convolutions

Local connectivity (small patches) + weight sharing (same filter everywhere). 32×32×3 → 100 FC units: 307,300 params; 16 filters 3×3×3: 448.

10.1 From Fully Connected to Convolutional

Convolution

Slide filter, multiply-add per position (+ bias) → feature map. Output size $\lfloor (n + 2p - k)/s \rfloor + 1$.

10.2 The Convolution Operation

Pooling

Max or average over windows (e.g. 2×2, stride 2). No weights. Shrinks maps, adds tolerance to small shifts.

10.3 Pooling

CNN components

Conv: local features. ReLU: non-linearity. Pooling: shrink + shift tolerance. FC: combine features → prediction.

10.4 CNN Components and Feature Hierarchies

Feature hierarchy

Early: edges, corners. Middle: textures, parts. Deep: objects, faces. Each layer combines the one below over a larger area.

10.4 CNN Components and Feature Hierarchies

PyTorch

The code patterns and the mistakes that cost marks.

Logits rule

CrossEntropyLoss / BCEWithLogitsLoss take raw logits. Don't put softmax/sigmoid in the model when using them.

6.4 Binary, Multi-class and Logits in PyTorch

Dropout in PyTorch

`nn.Dropout(p)` after the activation; `model.train()` / `model.eval()`; never on the output layer. $p \approx 0.5$ for big FC layers, 0.1–0.2 otherwise.

9.3 Dropout

Training loop

`zero_grad()` → forward + loss → `loss.backward()` → `optimizer.step()`. `model.train()` / `model.eval()` + `torch.no_grad()` for validation.

11.3 Neural Networks in PyTorch

Classifier output

Return logits from the last `nn.Linear`; `nn.CrossEntropyLoss` applies log-softmax. Softmax only for reporting probabilities.

11.3 Neural Networks in PyTorch

Core concepts

The big ideas, each in a sentence or two.

XOR with a hidden layer

Hidden units compute OR ($x_1 + x_2 - 0.5$) and AND ($x_1 + x_2 - 1.5$); output $h_1 - h_2 - 1$. The non-linear hidden representation makes XOR separable.

3.2 Solving XOR with a Hidden Layer

What depth adds

Layer 1: half-planes. Layer 2 (AND): convex regions. Layer 3 (OR): unions, so non-convex or disconnected regions.

3.3 Why Depth Helps: Composing Regions

Bump intuition

$\sigma(k(x-a)) - \sigma(k(x-b)) \approx 1$ on $[a, b]$. Sum scaled bumps → staircase approximation; more bumps, smaller error, more units.

5.2 Building Functions from Bumps

UAT and classifiers

Approximate the continuous class probabilities (sigmoid/softmax), then apply a decision rule (argmax / threshold).

5.3 What the Theorem Does Not Promise

Comparisons

Side-by-side differences between ideas that are easy to mix up.

ML vs DL

ML: any model whose parameters are learned from data. DL: ML with multi-layered neural networks that learn their own representations.

1.1 Machine Learning and Deep Learning

Learning types

Supervised: labelled (x, y) → classification (category) or regression (number). Unsupervised: no labels → structure. Semi-supervised: few labels. Reinforcement: rewards → policy.

1.2 Types of Learning

Separable or not

AND, OR: linearly separable (one neuron works). XOR: not separable (needs a hidden layer).

2.2 Decision Boundaries and Linear Separability

Two ways to solve XOR

Add the feature $x_1 x_2$ (XOR $= x_1 + x_2 - 2x_1 x_2$), or add a hidden layer that learns the representation.

2.3 The XOR Problem

Vanishing vs dying

Vanishing: small f' factors multiply across layers (sigmoid). Dying ReLU: a unit stuck at $z < 0$, output and gradient 0.

4.3 Dying ReLU, Leaky ReLU and ELU

L1 vs L2

L1: pull $\lambda \operatorname{sign}(w)$, exact zeros, sparsity. L2: pull λw, small weights, the DL default.

9.1 L2 and L1 Penalties

Depth vs width

Depth: hierarchical features, efficient, harder to train (vanishing gradients). Width: easy to train, parallel, but may need far more neurons. Both add capacity, so both can overfit.

11.1 Depth versus Width

Tabular regression design

Dense ReLU layers (64→32→16), 1 linear output, MSE, L2 + early stopping (+ dropout). Standardise numerics, one-hot or embed categories.

11.2 Designing a Network for Tabular Data

Common traps

Mistakes that cost marks, and the correct statement.

Index, not power

$x^{(i)}$ is training example i; x^2 is a square. Write a legend of indices before long derivations.

1.4 Notation, Probability and Jargon

XOR proof in one line

Positives give $w_1 + w_2 + 2b > 0$; negatives give $w_1 + w_2 + 2b < 0$. Contradiction, so one neuron cannot solve XOR.

2.3 The XOR Problem

Shape conventions

Lecture: $Wx + b$, W is outputs×inputs. Code: $XW + b$, rows are examples, W is inputs×outputs.

3.1 Layers, Matrices and Shapes

Linear layers collapse

$W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$: one affine layer. Depth needs non-linear activations.

3.4 Why Non-linearity Is Essential

Sign is untrainable

Its derivative is 0 almost everywhere, so gradient descent gets no signal. Use sigmoid, tanh or ReLU.

4.1 Sigmoid, tanh, ReLU and Friends

UAT limits

Existence only: no bound on size, no training method, no guarantee GD finds it, nothing about generalisation or data needed.

5.3 What the Theorem Does Not Promise

Accuracy as a loss

Accuracy is piecewise constant: zero gradient. Train on cross-entropy; report accuracy.

6.4 Binary, Multi-class and Logits in PyTorch

Leakage

Same patient/clip in two sets; random split of time series; normalisation stats from all data; tuning on the test set.

8.2 Training, Validation and Test Sets