AI Research Labs

Section 8 of 11 · 5 lessons

Generalisation and Data

Why fitting is not learning, how to split and prepare data, how to read learning curves, and a reliable training procedure.

8.1 Fitting Is Not Learning

8.2 Training, Validation and Test Sets

8.3 Underfitting, Overfitting and Learning Curves

8.4 Preparing the Data

8.5 Hyperparameters and a Training Recipe



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.

Where this section fits

Every lesson is a step in building a model. These are the steps this section teaches; the Big picture PDF and the online map show how they connect to the rest.

2

BUILDING AND TRAINING A NETWORK · STEP 2

Prepare the data

How do I split, scale and encode the data?

Lessons here: 8.2, 8.4 · feeds into Evaluate honestly, Choose the activations

7

BUILDING AND TRAINING A NETWORK · STEP 7

Update the weights

How do I turn gradients into a better network?

Lessons here: 8.5 · feeds into Evaluate honestly

8

BUILDING AND TRAINING A NETWORK · STEP 8

Evaluate honestly

Is it learning, or memorising?

Lessons here: 8.1, 8.3 · feeds into Regularise, Size the network

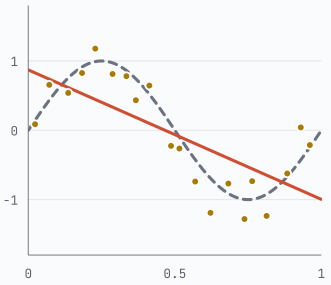
Fitting Is Not Learning

The goal of training is to predict new data well, not to match the training data. Polynomials of degree 1, 4 and 15 fitted to the same 20 noisy points show the difference.

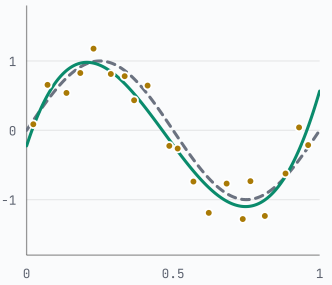
The real goal

Training minimises the loss on the training examples, but that loss is only a **proxy**. What we want is good predictions on **new** inputs from the same distribution: **generalisation**. A flexible enough network can fit any training set, even one whose labels were shuffled at random, without learning anything useful.

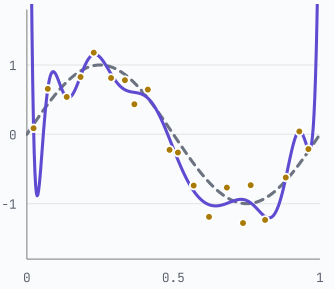
One dataset, three models



Degree 1: too simple. Misses the pattern (underfits).



Degree 4: follows the pattern, ignores the noise.



Degree 15: chases every point, wild in between (overfits).

Twenty noisy samples of $\sin 2\pi x$ (dashed). The degree-15 polynomial has the **lowest** error on these 20 points and the **worst** predictions between them. A lower training error does not mean a better model.

THE RULE

Whether a model generalises can only be measured on data it was **not** trained on. That's what validation and test sets are for (next lesson).

Common doubts

FIRST PASS **If the degree-15 fit passes near every point, why is it worse?**

Because the points contain noise. A model flexible enough to pass through every point is also fitting the noise, which won't be there in new data. Between the training points, nothing constrains it, so it swings wildly. The degree-4 fit captures the repeatable pattern and ignores the noise.

FIRST PASS **What are risk, empirical risk, ERM and regularised ERM, in plain words?**

New names for things you already do:

NAME	MEANING	YOU KNOW IT AS
Risk	Average loss on **all possible** data	What you care about, but can't compute
Empirical risk	Average loss on the **training** data	The training cost J
ERM	Choose the weights that minimise empirical risk	Ordinary training
Regularised ERM	Minimise empirical risk + a penalty	Training with L1 or L2 (lesson 9.1)

Example: training losses 0.2, 0.4 and 0.6 give empirical risk $J = 0.4$. With an L2 penalty of $\lambda \cdot \frac{1}{2} \sum w^2 = 0.1 \times 0.5 = 0.05$, regularised ERM minimises 0.45 instead.

Memory line: "Empirical risk = training loss. ERM = training. Regularised ERM = training with a penalty."

Cheatsheet

Generalisation

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/generalisation/fitting-is-not-learning

Training, Validation and Test Sets

What each data split is for, typical split sizes, the forms of data leakage that quietly ruin an evaluation, and k-fold cross-validation when data is scarce.

Three disjoint sets

SET	USED FOR	NEVER
Training	Fitting the weights by gradient descent	Judging the final model
Validation	Every other choice: architecture, learning rate, regularisation, number of epochs	Computing training gradients
Test	One final estimate of generalisation, at the very end	Tuning anything: look at it once

Typical splits are 80/10/10 or 60/20/20. Shuffle before splitting so all three sets have the same distribution.

Data leakage

Leakage is any route by which information from the evaluation data gets into training, making scores look better than they will be in use:

- **The same object in two sets:** several images of one patient, or consecutive video frames, split across train and test. Split by patient or by clip instead.
- **Time:** for time series, train on the past and test on the future, never a random split.
- **Preprocessing statistics** (means, standard deviations, min/max) computed on all the data. Compute them on the training set only.
- **Tuning on the test set**, even informally by trying a few models and keeping the best test score.

Scarce data: k-fold cross-validation

Split the training data into k folds. Train k times, each time validating on a different fold, and average the k validation scores. Every example is used for validation exactly once.

Common doubts

FIRST PASS **Why do I need both a validation set and a test set?**

Because every decision you make using the validation score slightly fits the model to the validation set. After choosing among many options, the best validation score is optimistic. The test set, untouched until the end, gives an honest estimate. Reusing it for decisions turns it into another validation set.

GOING DEEPER **If my training set already contains every possible input, do I still need a test set?**

No. In the XOR lab all four possible inputs are in the training set, so there is nothing new to generalise to and a test set adds nothing. A test set matters whenever the model must work on inputs it hasn't seen, which is almost always.

FIRST PASS **To judge generalisation, do I compare training with validation or with test?**

With **validation** while you're still making choices (model size, λ, when to stop), and with **test** only once at the end, to report the final number. If you look at the test score while choosing, the choice adapts to the test set and the reported number becomes too optimistic.

Thumb rule: "Validation is for choosing, test is for reporting."

FIRST PASS **What information "wouldn't be available at prediction time"? Examples of leakage?**

Ask of every feature: "At the moment I make the prediction, would I really have this value?" If not, it leaks the answer.

TASK	LEAKY FEATURE	WHY IT LEAKS
Does the patient have pneumonia?	"Took antibiotics"	Given **after** the diagnosis
Will this customer cancel?	"Cancellation reason"	Filled in only once they've cancelled
Will this loan default?	"Sent to debt collector"	Happens only after a default
Tomorrow's price	Any data from tomorrow onwards	That's the future

The second kind is test data leaking into training: normalising with statistics from the whole dataset (compute them on the training set only), the same patient or photo in two sets (split by person), or a random split of a time series (train on the past, test on the future).

Thumb rule: "If it comes from the future or from the test set, it's leakage." The symptom is a score that looks too good, then falls apart in real use.

Cheatsheet

Three sets

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

Leakage

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/generalisation/training-validation-and-test-sets

Underfitting, Overfitting and Learning Curves

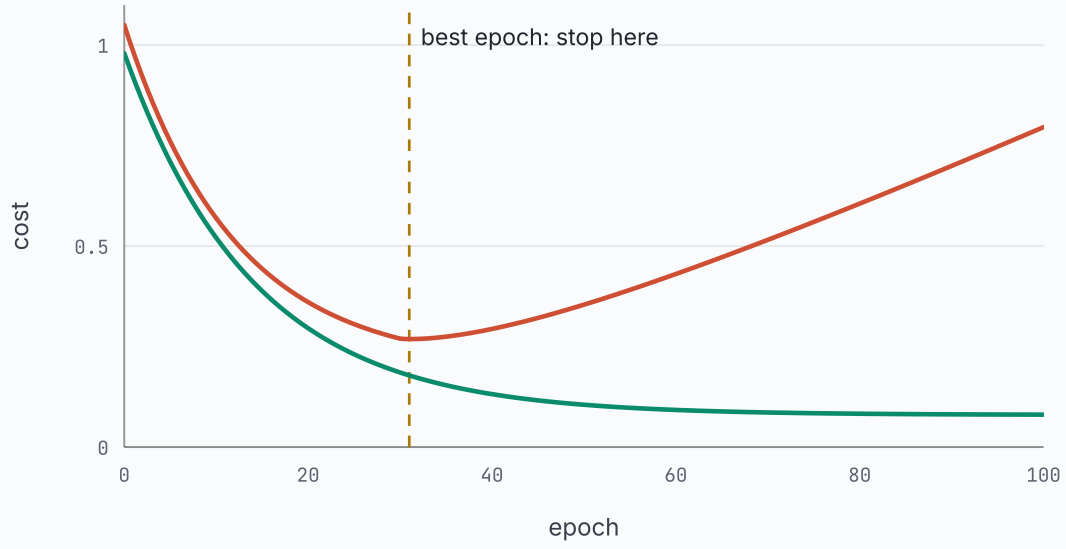
How to tell underfitting from overfitting with training and validation errors, how to read learning curves, and the bias–variance decomposition that explains both.

Diagnosing from the two errors

TRAINING ERROR	VALIDATION ERROR	DIAGNOSIS
High	High	**Underfitting**: too little capacity, or not trained enough
Low	Much higher	**Overfitting**: the gap measures it
Low	Low, close to training	Good fit

Training error alone never tells you: it keeps falling as capacity grows.

Learning curves



Training cost falls throughout. Validation cost reaches a minimum and then rises as the network starts fitting noise. Record both after every epoch.

The epoch with the lowest validation cost is where to stop: **early stopping** (Section 9).

Bias–variance decomposition

Model the target as $y = f(x) + \varepsilon$ with noise variance σ^2. For a model $\hat{y}(x)$ trained on a random training set:

$$\mathbb{E}\big[(\hat{y} - y)^2\big] = \underbrace{\big(\mathbb{E}[\hat{y}] - f(x)\big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}\big[(\hat{y} - \mathbb{E}[\hat{y}])^2\big]}_{\text{variance}} + \underbrace{\sigma^2}_{\text{noise}}$$

- **Bias:** systematic error of the model class. Low capacity (degree 1) has high bias: underfitting.
- **Variance:** how much the fitted model changes with the training sample. High capacity (degree 15) has high variance: overfitting.
- **Noise:** irreducible; no model can remove it.

More capacity trades bias for variance. Regularisation lowers variance at the cost of a little bias.

Common doubts

FIRST PASS **Why does the validation cost go up again if the network is still improving on training data?**

Early in training the network learns the general pattern, which helps both sets. Later it starts fitting quirks and noise specific to the training examples. That keeps lowering the training cost but makes predictions on new data worse, so validation cost rises.

FIRST PASS **What is the difference between convergence and overfitting?**

Convergence is about the optimiser: the training loss has stopped going down. **Overfitting** is about generalisation: the training loss keeps falling while the validation loss rises. They can happen together or apart. A network can overfit long before training converges, which is why you watch validation loss and stop early, and it can converge without overfitting if the model and data suit each other.

FIRST PASS **When do I check for bias and variance, and how?**

After every training run, by comparing training loss with validation loss, before you ever touch the test set.

TRAINING LOSS	GAP (VALIDATION - TRAINING)	PROBLEM	FIX
High	Small	High bias (underfitting)	Bigger model, train longer, lower λ
Low	**Large**	High variance (overfitting)	The fixes in the next doubt
Low	Small	Good fit	Stop and test

The trade-off appears when you fix things: a bigger model lowers bias but raises variance, and more regularisation does the opposite. Keep whichever setting gives the lowest **validation** loss.

Thumb rule: "Training loss shows bias. The gap shows variance."

FIRST PASS **What can I do to stop overfitting?**

Every fix falls into one of three groups:

GROUP	METHOD	HOW IT HELPS	LESSON
1. More or better data	More data	Harder to memorise many examples	8.1
	Data augmentation	"New" examples made by flipping, cropping or adding noise	9.4
2. Limit the model	Smaller model	Fewer neurons, less room to memorise noise	11.1
	L1 / L2	A penalty on big weights keeps the function smooth	9.1
	Dropout	Randomly switching off neurons stops co-adaptation	9.3
3. Limit the training	Early stopping	Stop when validation loss rises; keep the best epoch	9.2

All of them lower variance at the cost of a little bias, and their settings (λ, p, size, patience) are chosen on the validation set. Too much of any of them causes underfitting.

Memory line: "Feed it more, restrict the model, or stop it early."

Cheatsheet

Diagnose

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

Bias–variance

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/generalisation/underfitting-overfitting-and-learning-curves

Preparing the Data

Scaling features, handling missing values and categories, checking labels and dealing with unbalanced classes. Each with the reason it matters for a neural network.

Scale the features

If one feature lies in $[0.1, 0.7]$ and another in $[-10^4, 2 \times 10^4]$, the loss surface is badly conditioned (gradient descent zig-zags) and large inputs push sigmoid units into saturation. Rescale each feature j:

$$\text{min–max: } \tilde{x}_j = \frac{x_j - \min_j}{\max_j - \min_j} \in [0, 1], \qquad \text{standardisation: } \tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

Compute $\min_j, \max_j, \mu_j, \sigma_j$ on the **training set only** and apply the same transformation to validation, test and future inputs. Images: divide pixel values by 255. Standardise regression targets too.

> FROM THE LAB
>
> Inputs in $[0, 10]$ saturated the sigmoid hidden units and training stalled. Standardising them (training mean and standard deviation) fixed it. Always do this.

Other checks

ISSUE	WHAT TO DO
Missing features	Few affected examples: drop them. Otherwise impute with the training mean or median and add a "was missing" indicator
Categorical inputs	One-hot encode. An integer code (red = 1, green = 2, blue = 3) invents an order and spacing
Noisy labels	Labels are expensive and often wrong; inspect a sample by hand
Unbalanced classes	If 99% of examples are one class, predicting it always scores 99% accuracy. Use a confusion matrix, precision and recall; re-weight the loss or over-sample the rare class

Common doubts

FIRST PASS **Min–max or standardisation: which should I use?**

Both work for most networks. Standardisation is the common default and copes better with outliers (one extreme value squashes every other value under min–max). Min–max is natural when the range is known and fixed, such as pixel values from 0 to 255.

FIRST PASS

When do I use one-hot encoding?

When a feature is a category with **no natural order**: colour, city, blood type. Coding red = 1, green = 2, blue = 3 would tell the network that blue > green > red, an order that isn't true. One-hot gives each category its own 0/1 column, so all categories are equally far apart.

KIND OF CATEGORY	EXAMPLE	USE
No order	colour, city	One-hot
Real order	small < medium < large	Plain numbers 0, 1, 2
Thousands of categories	product IDs	Embeddings (lesson 11.2)

Class labels for multi-class outputs are one-hot too: cat = (1, 0, 0), compared with the softmax output by cross-entropy.

Thumb rule: "No order → one-hot. Real order → numbers."

FIRST PASS

What are precision, recall and F1, and how do I remember them?

- **Precision → Predicted:** of everything the model *predicted* positive, how many really are? $\frac{TP}{TP+FP}$
- **Recall → Real:** of everything that *really* is positive, how many did the model find? $\frac{TP}{TP+FN}$
- **F1** combines them: $F_1 = \frac{2PR}{P+R}$. It's low if **either** is low, so a model can't score well by being good at only one.

Example: there are 10 real positives; the model predicts 8 positives, of which 6 are right. Precision $= 6/8 = 0.75$, recall $= 6/10 = 0.6$, $F_1 = \frac{2\times0.75\times0.6}{1.35} \approx 0.67$.

Use them instead of accuracy when the classes are unbalanced.

Cheatsheet

Scaling

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

Data checks

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/generalisation/preparing-the-data

Hyperparameters and a Training Recipe

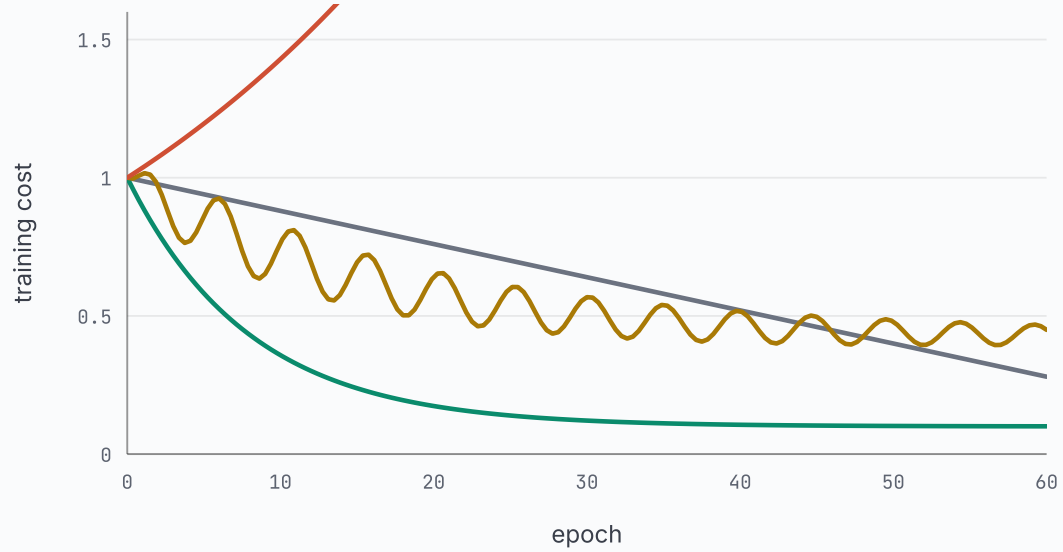
What hyperparameters are, how to search them, how to read a learning rate from the loss curve, and a seven-step procedure for training a network that works.

Hyperparameters

Everything not learned by gradient descent: the number of layers and units, activations, learning rate, batch size, number of epochs, regularisation strength λ, dropout rate and initialisation. Choose them on the **validation** set.

- **Grid search**: every combination; cost grows exponentially with the number of hyperparameters.
- **Random search**: usually better once you have more than two or three hyperparameters.
- **Bayesian optimisation**, or by hand guided by learning curves.
- Search η and λ on a **log scale** (0.001, 0.01, 0.1, ...).

Reading the learning rate from the loss curve



The learning rate matters most. A slowly falling cost: raise η. An oscillating or exploding cost: lower it.

A training recipe

1. **Data:** shuffle; split train/validation/test; normalise with training statistics; encode categories.
2. **Task:** pick the output layer and loss (Section 6).
3. **Sanity check:** a reasonably sized network should drive the training cost near zero on about 100 examples. If it can't, there's a bug in the code or data.

4. **Train** with mini-batch gradient descent or Adam, recording training and validation curves every epoch.
5. **Underfitting** (both errors high): bigger network, longer training, higher η; check normalisation and activations.
6. **Overfitting** (large gap): more data or augmentation, weight decay, dropout, early stopping, or a smaller network.
7. **Select** hyperparameters on validation, evaluate **once** on the test set, and report that number.

Common doubts

FIRST PASS **Why should a network be able to overfit 100 examples?**

Because memorising a tiny dataset is the easiest thing a network can do. If it can't even do that, the problem isn't generalisation: it's a bug (wrong labels, a broken gradient, a learning rate far off). It's a quick test that saves hours of tuning.

GOING DEEPER **What did the lab show matters most for training a small XOR network?**

Width had the largest effect: 2 hidden units converged in about 80% of random starts, 3 or more in all of them. The learning rate barely mattered within 0.3–3, but a huge one (300) pushed every unit into saturation on the first step and training stalled at loss 0.25. Very small initial weights (0.1) left all hidden units identical, on a plateau.

Cheatsheet

Training recipe

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

Learning-rate diagnosis

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/generalisation/hyperparameters-and-a-training-recipe