The big picture · 2 maps · 17 steps

The big *picture*

Why you are learning what you are learning: how every lesson in the course helps build a neural network that works.

1 From a neuron to a trained network: the deep learning workflow

◆ Why any of it works



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.

The big picture · 44 lessons · 2 maps

How it all fits *together*

Running example: handwritten digits. 70,000 greyscale images of 28×28 pixels, each labelled 0–9. The same data can go down either lane: a dense network on the 784 flattened pixels, or a convolutional network on the image itself. Then both are trained the same way.

MAP 1 · BUILDING AND TRAINING A NETWORK: FROM A NEURON TO A TRAINED NETWORK: THE DEEP LEARNING WORKFLOW

1. **Frame the task**
   Lessons 1.1, 1.2 · feeds Choose the output and loss
2. **Prepare the data**
   Lessons 8.2, 8.4 · feeds Evaluate honestly, Choose the activations
3. **Does the input have spatial structure?**
   Lessons 10.1 · feeds Understand the neuron, Convolve

DENSE LANE Tabular or flattened inputs

- M1 **Understand the neuron**
  Lessons 2.1, 2.2, 2.3 · feeds Stack layers
- M2 **Stack layers**
  Lessons 3.1, 3.2, 3.3, 3.4 · feeds Choose the activations
- M3 **Size the network**
  Lessons 11.1, 11.2 · feeds Regularise

CONVOLUTIONAL LANE Images and spatial data

- C1 **Convolve**
  Lessons 10.2
- C2 **Pool**
  Lessons 10.3
- C3 **Stack into a hierarchy**
  Lessons 10.4 · feeds Choose the output and loss, Size the network

4. **Choose the activations**
   Lessons 4.1, 4.2, 4.3, 4.4 · feeds Compute the gradients
5. **Choose the output and loss**
   Lessons 6.1, 6.2, 6.3, 6.4 · feeds Compute the gradients
6. **Compute the gradients**
   Lessons 7.1, 7.2, 7.3, 7.4, 7.5 · feeds Update the weights
7. **Update the weights**
   Lessons 1.3, 7.6, 8.5, 11.3 · feeds Evaluate honestly
8. **Evaluate honestly**
   Lessons 8.1, 8.3 · feeds Regularise, Size the network
9. **Regularise**
   Lessons 9.1, 9.2, 9.3, 9.4 · feeds Prepare the data, Update the weights

↺ Back to step 7 (update the weights): retrain with the change, then evaluate again.

UNDERNEATH THE MAP: WHY ANY OF IT WORKS

- ◆ **Universal approximation**
  Lessons 5.1, 5.2, 5.3 · feeds Stack layers, Size the network
- ◆ **Notation, probability and jargon**
  Lessons 1.4 · feeds Compute the gradients

In the order you *study*

The map follows the deep learning workflow, the order you build a model in, so its lessons come from all over the course. This page lists the lessons in course order, with the workflow step each one serves. Study in this order; use the map to see where each lesson fits.

SECTION	LESSONS	WORKFLOW STEP
01 **Foundations of Deep Learning**	1.1 Machine Learning and Deep Learning 1.2 Types of Learning	1 Frame the task
	1.3 Gradient Descent and Momentum	7 Update the weights
	1.4 Notation, Probability and Jargon	◆ Notation, probability and jargon
02 **The Neuron and Linear Separability**	2.1 The Artificial Neuron 2.2 Decision Boundaries and Linear Separability 2.3 The XOR Problem	M1 Understand the neuron
03 **Multi-layer Perceptrons**	3.1 Layers, Matrices and Shapes 3.2 Solving XOR with a Hidden Layer 3.3 Why Depth Helps: Composing Regions 3.4 Why Non-linearity Is Essential	M2 Stack layers
04 **Activation Functions**	4.1 Sigmoid, tanh, ReLU and Friends 4.2 Vanishing Gradients 4.3 Dying ReLU, Leaky ReLU and ELU 4.4 Choosing Activation Functions	4 Choose the activations
05 **The Universal Approximation Theorem**	5.1 The Universal Approximation Theorem 5.2 Building Functions from Bumps 5.3 What the Theorem Does Not Promise	◆ Universal approximation
06 **Loss Functions and Output Layers**	6.1 Training as Minimising a Loss 6.2 MSE, MAE and Asymmetric Losses 6.3 Softmax and Cross-entropy 6.4 Binary, Multi-class and Logits in PyTorch	5 Choose the output and loss
07 **Backpropagation**	7.1 The Chain Rule and Computational Graphs 7.2 Backpropagation for a Single Sigmoid Neuron 7.3 One Hidden Layer: Deriving Every Gradient 7.4 Backpropagation in Matrix Form 7.5 Multiple Paths and Deeper Networks	6 Compute the gradients
	7.6 Mini-batches and Gradient Checking	7 Update the weights
08 **Generalisation and Data**	8.1 Fitting Is Not Learning	8 Evaluate honestly
	8.2 Training, Validation and Test Sets	2 Prepare the data
	8.3 Underfitting, Overfitting and Learning Curves	8 Evaluate honestly
	8.4 Preparing the Data	2 Prepare the data
	8.5 Hyperparameters and a Training Recipe	7 Update the weights

SECTION	LESSONS	WORKFLOW STEP
09 **Regularisation**	9.1 L2 and L1 Penalties 9.2 Early Stopping 9.3 Dropout 9.4 Data Augmentation	9 Regularise
10 **Convolutional Neural Networks**	10.1 From Fully Connected to Convolutional	? Does the input have spatial structure?
	10.2 The Convolution Operation	C1 Convolve
	10.3 Pooling	C2 Pool
	10.4 CNN Components and Feature Hierarchies	C3 Stack into a hierarchy
11 **Network Design and PyTorch**	11.1 Depth versus Width 11.2 Designing a Network for Tabular Data	M3 Size the network
	11.3 Neural Networks in PyTorch	7 Update the weights

1 Frame the task

What should the network predict, and from what?

KEY FORMULA

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

Labelled examples: a supervised task, either regression (a number) or classification (a category).

FACT OR FIGURE

Deep learning learns its own features in layers, but it still needs a clearly defined target: the type of y decides the output layer and the loss.

SKIP THIS STEP AND…

You build a network for the wrong target, such as predicting a digit's pixel brightness when you needed its label.

DIGITS EXAMPLE

Input: a 28×28 image. Target: one of 10 digits. A supervised, 10-class classification task.

Learn it

Taught in Section 1 (Foundations of Deep Learning) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **1.1 Machine Learning and Deep Learning**
  Machine learning builds a model and learns its parameters from data. Deep learning is the part of machine learning that uses many-layered neural networks. What that means, the learning loop, and a short history.
- **1.2 Types of Learning**
  Supervised, unsupervised, semi-supervised and reinforcement learning: what training information each uses, what it aims for, and the notation for training examples.

How it connects

- → **Choose the output and loss** (Step 5): The type of target decides the output layer and the loss

Next: Prepare the data Back to the map

Online: airesearchlabs.si/learn/dl/map/frame

2 Prepare the data

How do I split, scale and encode the data?

KEY FORMULA

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

Statistics from the training set only.

FACT OR FIGURE

Unscaled inputs in the thousands push sigmoid units into saturation, where $\sigma' \approx 0$ and learning stalls.

SKIP THIS STEP AND…

Statistics computed on all the data, or the same object in train and test, leak information and inflate the test score.

DIGITS EXAMPLE

Divide pixels by 255; keep 60,000 images for training (with some held out for validation) and 10,000 for the final test.

Learn it

Taught in Section 8 (Generalisation and Data) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **8.2 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.
- **8.4 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.

How it connects

- → **Evaluate honestly** (Step 8): The held-out sets are what evaluation measures
- → **Choose the activations** (Step 4): Badly scaled inputs saturate sigmoid units
- ← **Regularise** (Step 9): Data augmentation creates new training examples from the data you already have

Previous: Frame the task **Next:** Does the input have spatial structure? Back to the map

Online: airesearchlabs.si/learn/dl/map/data

? Does the input have spatial structure?

Dense layers, or convolutions?

KEY FORMULA

$$\underbrace{3072 \times 100 + 100}_{\text{dense: } 307{,}300} \quad \text{vs} \quad \underbrace{16 \times 27 + 16}_{\text{conv: } 448}$$

FACT OR FIGURE

A convolution's parameter count depends on the filter size and number of filters, not the image size, thanks to weight sharing.

SKIP THIS STEP AND…

A dense layer on raw images ignores that nearby pixels belong together and needs a huge number of weights.

DIGITS EXAMPLE

A dense network on the 784 flattened pixels works (typically about 98% accuracy); a CNN on the image does better (over 99%) with fewer weights.

Learn it

Taught in Section 10 (Convolutional Neural Networks) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **10.1 From Fully Connected to Convolutional**
  Why a fully connected layer is a poor fit for images, and the two ideas that fix it: local connectivity and weight sharing. With the parameter count that shows the difference.

How it connects

- → **Understand the neuron** (Dense lane · step M1): Features without spatial structure: the dense lane
- → **Convolve** (Convolutional lane · step C1): Images and spatial data: the convolutional lane

Previous: Prepare the data **Next:** Understand the neuron or Convolve Back to the map

Online: airesearchlabs.si/learn/dl/map/fork

M1 Understand the neuron

What can one neuron do, and what can it never do?

KEY FORMULA

$$y = f(w^\top x + b), \qquad \text{boundary: } w^\top x + b = 0$$

FACT OR FIGURE

XOR's positives need $w_1 + w_2 + 2b > 0$ and its negatives $w_1 + w_2 + 2b < 0$: no single neuron can compute it.

SKIP THIS STEP AND…

You expect one linear unit to separate classes that no straight boundary can separate.

DIGITS EXAMPLE

One neuron can tell "is this pixel pattern more like a 1 or a 0?" only if a single weighted sum separates them.

Learn it

Taught in Section 2 (The Neuron and Linear Separability) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **2.1 The Artificial Neuron**
  A neuron computes a weighted sum of its inputs, adds a bias and applies an activation function. What each part does, a worked example, and why the bias is just a weight on a constant input.
- **2.2 Decision Boundaries and Linear Separability**
  A neuron with a threshold activation splits the input space with a straight line (a hyperplane). Which problems that can solve: AND and OR worked through, and the definition of linear separability.
- **2.3 The XOR Problem**
  A proof that no single neuron can compute XOR, what happens when you try to train one anyway, and two ways out: hand-made features or a hidden layer.

How it connects

- → **Stack layers** (Dense lane · step M2): XOR shows why a hidden layer is needed
- ← **Does the input have spatial structure?** (The fork): Features without spatial structure: the dense lane

Previous: Does the input have spatial structure? **Next:** Stack layers Back to the map

Online: airesearchlabs.si/learn/dl/map/m-neuron

M2 Stack layers

How do hidden layers create new representations?

KEY FORMULA

$$y^{(k)} = f_k\left(W_k\, y^{(k-1)} + b_k\right), \qquad W_k : m \times n$$

FACT OR FIGURE

Without non-linear activations, any stack of layers collapses to one: $W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$.

SKIP THIS STEP AND…

Linear hidden layers add parameters but no expressive power.

DIGITS EXAMPLE

A 784 → 128 → 10 network has 101,770 parameters, almost all in the first layer.

Learn it

Taught in Section 3 (Multi-layer Perceptrons) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **3.1 Layers, Matrices and Shapes**
  A multi-layer perceptron stacks layers of neurons. How one layer is written as a matrix product, how to read the shapes, how to count parameters, and the two conventions (lecture vs code) you must not mix.
- **3.2 Solving XOR with a Hidden Layer**
  An explicit 2–2–1 network that computes XOR, checked for all four inputs. The hidden layer changes the representation so that one output neuron can separate the classes.
- **3.3 Why Depth Helps: Composing Regions**
  With threshold neurons, the first layer cuts the space into half-planes, the next combines them into convex regions, and the next into unions of regions. That is the geometric picture of what depth adds.
- **3.4 Why Non-linearity Is Essential**
  Stacking layers without non-linear activations gives nothing new: any stack of linear layers collapses to a single linear layer. The algebra, a numerical check, and the consequence for XOR.

How it connects

- → **Choose the activations** (Step 4): Non-linear activations are what make depth useful
- ← **Understand the neuron** (Dense lane · step M1): XOR shows why a hidden layer is needed
- ← **Universal approximation** (Foundation): Why one hidden layer is enough in principle

Previous: Understand the neuron **Next:** Size the network Back to the map

Online: airesearchlabs.si/learn/dl/map/m-layers

M3 Size the network

How deep, how wide, and for which task?

KEY FORMULA

$$\text{params} = \sum_{k} (n_{k-1} n_k + n_k)$$

Width grows parameters with the product of adjacent layers.

FACT OR FIGURE

Depth can represent structured functions with far fewer units than one wide layer, but it is harder to train.

SKIP THIS STEP AND…

An oversized network on a small dataset memorises it; an undersized one underfits.

DIGITS EXAMPLE

For digits as a dense network: one or two hidden layers of 128–256 ReLU units, and a 10-way output.

Learn it

Taught in Section 11 (Network Design and PyTorch) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **11.1 Depth versus Width**
  More layers or more units per layer? How depth and width differ in expressive power, parameter count and trainability, and why most successful networks are deep rather than very wide.
- **11.2 Designing a Network for Tabular Data**
  A worked design for a regression problem: predicting house prices from features such as size, bedrooms and location. Architecture, activations, loss, regularisation, categorical features and the challenges to expect.

How it connects

- → **Regularise** (Step 9): Large networks need regularising
- ← **Stack into a hierarchy** (Convolutional lane · step C3): Depth and width are design choices for convolutional networks too
- ← **Evaluate honestly** (Step 8): High training and validation loss means underfitting: the network needs more capacity
- ← **Universal approximation** (Foundation): Why depth is still chosen in practice

Previous: Stack layers **Next:** Choose the activations Back to the map

Online: airesearchlabs.si/learn/dl/map/m-design

C1 Convolve

How does a filter find a pattern anywhere in the image?

KEY FORMULA

$$\text{output size} = \left\lfloor \frac{n + 2p - k}{s} \right\rfloor + 1$$

FACT OR FIGURE

A 3×3 filter weighting columns $+1, 0, -1$ responds to vertical edges: brighter on the left gives a positive output.

SKIP THIS STEP AND…

Without weight sharing, the same edge detector would have to be learned separately at every position.

DIGITS EXAMPLE

A 28×28 digit through 16 filters of 3×3 (padding 1) gives 16 feature maps of 28×28.

Learn it

Taught in Section 10 (Convolutional Neural Networks) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **10.2 The Convolution Operation**
  Slide a small filter over the input, multiply and add at each position: that produces a feature map. A full worked example by hand, what the filter detects, channels, and the output-size formula.

How it connects

- ← **Does the input have spatial structure?** (The fork): Images and spatial data: the convolutional lane

Previous: Does the input have spatial structure? **Next:** Pool Back to the map

Online: airesearchlabs.si/learn/dl/map/c-conv

C2 Pool

How do feature maps shrink while keeping what matters?

KEY FORMULA

$$\max\text{ pool }\{1,3,4,2\}=4, \qquad \text{average}=2.5$$

FACT OR FIGURE

Pooling has no learned parameters; 2×2 pooling with stride 2 quarters the number of values.

SKIP THIS STEP AND…

Without any downsampling, maps stay large and units never see a big enough region to recognise whole shapes.

DIGITS EXAMPLE

2×2 max pooling turns 28×28 maps into 14×14, then 7×7.

Learn it

Taught in Section 10 (Convolutional Neural Networks) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **10.3 Pooling**
  Pooling shrinks feature maps by summarising small regions. Max and average pooling worked out by hand, and what pooling buys: less computation and tolerance to small shifts.

Previous: Convolve **Next:** Stack into a hierarchy Back to the map

Online: airesearchlabs.si/learn/dl/map/c-pool

C3 Stack into a hierarchy

How do layers build up from edges to objects?

KEY FORMULA

[conv → ReLU → pool] × several, then flatten → dense → output

FACT OR FIGURE

Two stacked 3×3 convolutions already see a 5×5 region of the image; the region grows with depth and pooling.

SKIP THIS STEP AND…

Leaving out the ReLUs between convolutions collapses the stack into a single linear filter.

DIGITS EXAMPLE

Early filters find strokes and edges; deeper ones respond to loops and line combinations that make up each digit.

Learn it

Taught in Section 10 (Convolutional Neural Networks) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **10.4 CNN Components and Feature Hierarchies**
  The role of each part of a CNN (convolutional layers, activations, pooling and fully connected layers) and how stacking them builds features from edges to textures to whole objects.

How it connects

- → **Choose the output and loss** (Step 5): Dense layers and a softmax finish the network
- → **Size the network** (Dense lane · step M3): Depth and width are design choices for convolutional networks too

Previous: Pool **Next:** Choose the activations Back to the map

Online: airesearchlabs.si/learn/dl/map/c-stack

4 Choose the activations

Which non-linearity in hidden layers, and which at the output?

KEY FORMULA

$$\sigma'(z) = \sigma(z)\big(1-\sigma(z)\big) \le 0.25, \qquad \mathrm{ReLU}'(z) \in \{0, 1\}$$

FACT OR FIGURE

Ten sigmoid layers multiply at most $0.25^{10} \approx 10^{-6}$ into the gradient: vanishing gradients.

SKIP THIS STEP AND…

Deep sigmoid networks barely train their early layers; ReLU units with a too-high learning rate can die.

DIGITS EXAMPLE

ReLU in the hidden layers; softmax over the 10 digit classes at the output.

Learn it

Taught in Section 4 (Activation Functions) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **4.1 Sigmoid, tanh, ReLU and Friends**
  The common activation functions side by side: their formulas, ranges and derivatives, where each is used, and why the sign function cannot be trained by gradient descent.
- **4.2 Vanishing Gradients**
  Why early layers of a deep sigmoid network barely learn: backpropagation multiplies one activation derivative per layer, and sigmoid derivatives are at most 0.25. The numbers, the mechanism and the fixes.
- **4.3 Dying ReLU, Leaky ReLU and ELU**
  ReLU solves vanishing gradients for active units but creates a new failure: a unit stuck on the negative side gets zero gradient forever. How it happens and the remedies.
- **4.4 Choosing Activation Functions**
  Which activation to use where: ReLU for deep hidden layers, sigmoid for shallow networks and binary outputs, linear for regression outputs, softmax for multi-class outputs.

How it connects

- → **Compute the gradients** (Step 6): Every backprop step multiplies by the activation derivative
- ← **Prepare the data** (Step 2): Badly scaled inputs saturate sigmoid units
- ← **Stack layers** (Dense lane · step M2): Non-linear activations are what make depth useful

Previous: Size the network or Stack into a hierarchy **Next:** Choose the output and loss Back to the map

Online: airesearchlabs.si/learn/dl/map/act

5 Choose the output and loss

How do I measure how wrong the network is?

KEY FORMULA

$$H = -\sum_j z_j \log o_j, \qquad \frac{\partial H}{\partial s_j} = o_j - z_j$$

FACT OR FIGURE

Probability 0.95 on the correct class costs $-\ln 0.95 \approx 0.05$; probability 0.01 costs 4.6. Confident mistakes are punished hard.

SKIP THIS STEP AND…

A softmax inside the model plus `nn.CrossEntropyLoss` applies softmax twice and distorts the gradients.

DIGITS EXAMPLE

10 logits → `nn.CrossEntropyLoss`; report accuracy, but train on cross-entropy.

Learn it

Taught in Section 6 (Loss Functions and Output Layers) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **6.1 Training as Minimising a Loss**
  Training has two steps: choose a loss that measures how wrong the model is, then minimise it. Worked through for a straight line with mean squared error: the gradients, one update, and the loss going down.
- **6.2 MSE, MAE and Asymmetric Losses**
  Regression losses compared: MSE punishes big errors hard and estimates the mean; MAE is robust to outliers and estimates the median; pinball losses price over- and under-prediction differently.
- **6.3 Softmax and Cross-entropy**
  Softmax turns class scores into probabilities; cross-entropy scores those probabilities. A worked example, the numerical-stability trick, and the derivation of the famously simple gradient: predicted minus target.
- **6.4 Binary, Multi-class and Logits in PyTorch**
  Which output layer and loss go with which task, binary cross-entropy worked out, why accuracy can't be the training loss, the best accuracy any classifier can reach, and the logits rule that trips people up in PyTorch.

How it connects

- → **Compute the gradients** (Step 6): The loss sets the output error that backprop starts from
- ← **Frame the task** (Step 1): The type of target decides the output layer and the loss
- ← **Stack into a hierarchy** (Convolutional lane · step C3): Dense layers and a softmax finish the network

Previous: Choose the activations **Next:** Compute the gradients Back to the map

Online: airesearchlabs.si/learn/dl/map/loss

6 Compute the gradients

How does each weight learn its share of the blame?

KEY FORMULA

$$\delta^{(k)} = \left(\delta^{(k+1)} W_{k+1}^{\top}\right) \odot f'\left(S^{(k)}\right), \qquad \frac{\partial E}{\partial W_k} = (\text{input})^{\top} \delta^{(k)}$$

FACT OR FIGURE

Backpropagation gets every gradient for about the cost of one extra forward pass, by reusing each layer's error signal.

SKIP THIS STEP AND…

Following only one path when a unit feeds several outputs: the gradient must sum over every path.

DIGITS EXAMPLE

For the dense digit network: $\delta^{(2)} = (O - Z)/N$ at the softmax output, then back through $W^{(2)\top}$ and the ReLU derivatives.

Learn it

Taught in Section 7 (Backpropagation) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **7.1 The Chain Rule and Computational Graphs**
  Backpropagation is the chain rule applied backwards through the network, reusing every intermediate result. The chain rule on a small graph, with numbers, and the two rules: multiply along a path, add across paths.
- **7.2 Backpropagation for a Single Sigmoid Neuron**
  The derivation you are most likely to be asked for: the gradient of mean squared error for one sigmoid neuron over a batch of examples, step by step, with a numerical example and the matrix form.
- **7.3 One Hidden Layer: Deriving Every Gradient**
  The classic exam derivation: a network with one sigmoid hidden layer and a linear output, trained with squared error. Output-layer gradients, the hidden error signal, input-layer gradients, and a full numerical example.
- **7.4 Backpropagation in Matrix Form**
  The same algorithm for a whole batch at once: four steps, the shape of every matrix, the element-wise product, how the output error changes with the loss, and a complete NumPy implementation.
- **7.5 Multiple Paths and Deeper Networks**
  When a hidden unit feeds several outputs, its gradient is a sum over paths. A two-hidden-layer, two-output example, the general backward recursion, and how it explains vanishing gradients.

How it connects

- → **Update the weights** (Step 7): The gradients feed the weight update
- ← **Choose the activations** (Step 4): Every backprop step multiplies by the activation derivative
- ← **Choose the output and loss** (Step 5): The loss sets the output error that backprop starts from
- ← **Notation, probability and jargon** (Foundation): Derivations depend on clear index notation

Previous: Choose the output and loss **Next:** Update the weights Back to the map

Online: airesearchlabs.si/learn/dl/map/backprop

7 Update the weights

How do I turn gradients into a better network?

KEY FORMULA

$$\theta \leftarrow \theta - \eta \nabla_\theta J$$

on mini-batches; momentum or Adam in practice.

FACT OR FIGURE

An oscillating or exploding loss means η is too large; a crawling one, too small. Search it on a log scale.

SKIP THIS STEP AND…

Forgetting `optimizer.zero_grad()` sums old gradients into every step.

DIGITS EXAMPLE

Mini-batches of 64 images: about 940 updates per epoch on 60,000 training images.

Learn it

Taught in Section 1 (Foundations of Deep Learning), Section 7 (Backpropagation), Section 8 (Generalisation and Data) and Section 11 (Network Design and PyTorch) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **1.3 Gradient Descent and Momentum**
  Training means minimising an objective. Gradient descent repeatedly steps against the gradient. A worked example, what the learning rate does, why a zero gradient is not proof of a minimum, and how momentum helps.
- **7.6 Mini-batches and Gradient Checking**
  How often to update: online, full-batch and mini-batch gradient descent compared. And how to test a hand-written gradient with central finite differences before trusting it.
- **8.5 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.
- **11.3 Neural Networks in PyTorch**
  Define a network, choose a loss and optimiser, and write the training loop. A 100 → 50 → 10 classifier line by line, the logits rule, train and eval modes, and a custom activation function.

How it connects

- → **Evaluate honestly** (Step 8): Training progress means little until it is measured on held-out data
- ← **Compute the gradients** (Step 6): The gradients feed the weight update
- ← **Regularise** (Step 9): Weight decay changes the update rule, and early stopping decides when training stops

Previous: Compute the gradients **Next:** Evaluate honestly Back to the map

Online: airesearchlabs.si/learn/dl/map/optimise

8 Evaluate honestly

Is it learning, or memorising?

KEY FORMULA

$$\mathbb{E}\left[(\hat{y} - y)^2\right] = \text{bias}^2 + \text{variance} + \sigma^2$$

FACT OR FIGURE

A big enough network can fit randomly shuffled labels: low training error alone proves nothing.

SKIP THIS STEP AND…

Choosing models on the test set makes the reported score optimistic.

DIGITS EXAMPLE

Watch training and validation loss every epoch; if validation loss turns up while training loss keeps falling, the network is overfitting.

Learn it

Taught in Section 8 (Generalisation and Data) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **8.1 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.
- **8.3 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.

How it connects

- → **Regularise** (Step 9): A train–validation gap calls for regularisation
- → **Size the network** (Dense lane · step M3): High training and validation loss means underfitting: the network needs more capacity
- ← **Prepare the data** (Step 2): The held-out sets are what evaluation measures
- ← **Update the weights** (Step 7): Training progress means little until it is measured on held-out data

Previous: Update the weights **Next:** Regularise Back to the map

Online: airesearchlabs.si/learn/dl/map/evaluate

9 Regularise

How do I close the gap between training and validation?

KEY FORMULA

$$W \leftarrow (1-\eta\lambda)W - \eta\frac{\partial J}{\partial W}, \qquad \tilde{h} = \frac{1}{1-p}\, m \odot h$$

FACT OR FIGURE

Inverted dropout scales the kept units by $1/(1-p)$ in training, so nothing changes at test time.

SKIP THIS STEP AND…

Forgetting `model.eval()` leaves dropout on during evaluation, giving noisy, worse results.

DIGITS EXAMPLE

Small shifts and rotations of each digit as augmentation (never 180°, which turns a 6 into a 9), dropout before the output layer, early stopping.

Learn it

Taught in Section 9 (Regularisation) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **9.1 L2 and L1 Penalties**
  Add a penalty on the size of the weights to the loss. L2 gives weight decay: every step shrinks the weights a little. L1 pushes small weights to exactly zero. The derivation and the difference.
- **9.2 Early Stopping**
  Watch the validation cost every epoch, keep the best weights, and stop when it stops improving. Why it regularises, and the patience setting.
- **9.3 Dropout**
  Randomly switch off hidden units during training. Inverted dropout as lectured: drop with probability p, scale the survivors by 1/(1−p), change nothing at test time. Why it prevents co-adaptation and how to use it in PyTorch.
- **9.4 Data Augmentation**
  When you can't collect more data, make more from what you have: transformed copies that keep the right label. Which transformations suit images, audio and tabular data, and which ones break the label.

How it connects

- → **Prepare the data** (Step 2): Data augmentation creates new training examples from the data you already have
- → **Update the weights** (Step 7): Weight decay changes the update rule, and early stopping decides when training stops
- ← **Size the network** (Dense lane · step M3): Large networks need regularising
- ← **Evaluate honestly** (Step 8): A train–validation gap calls for regularisation

Previous: Evaluate honestly **Next:** Update the weights Back to the map

Online: airesearchlabs.si/learn/dl/map/regularise

Universal approximation

THE IDEA

$\sup_{x \in K} |h(x) - \mathrm{NN}(x)| < \varepsilon$ with one hidden layer

WHY IT MATTERS

A network with enough units can approximate any continuous function, so the architecture is never the hard limit. But it says nothing about size, training or generalisation.

Learn it

Taught in Section 5 (The Universal Approximation Theorem) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **5.1 The Universal Approximation Theorem**
  One hidden layer with enough neurons can approximate any continuous function on a bounded domain as closely as you like. The precise statement, what each condition means, and the extension that covers ReLU.
- **5.2 Building Functions from Bumps**
  The intuition behind universal approximation in one dimension: two steep sigmoids make a bump, many bumps make a staircase, and narrower steps track any continuous curve.
- **5.3 What the Theorem Does Not Promise**
  The universal approximation theorem is an existence result. It says nothing about how many neurons, how to find the weights, or whether the network will generalise. Plus: how it applies to classifiers with discrete outputs.

How it connects

- → **Stack layers** (Dense lane · step M2): Why one hidden layer is enough in principle
- → **Size the network** (Dense lane · step M3): Why depth is still chosen in practice

Next: Notation, probability and jargon Back to the map

Online: airesearchlabs.si/learn/dl/map/f-uat

Notation, probability and jargon

THE IDEA

$\mathbb{E}[X] = \sum_i x_i p(x_i)$; $x^{(i)}$ is example i, not a power

WHY IT MATTERS

Vectors, matrices, gradients and probabilities are the language of every later step.

Learn it

Taught in Section 1 (Foundations of Deep Learning) of the course. The map follows the deep learning workflow, the order you build a model in; the course teaches in the order that is easiest to learn, so the numbers can jump.

- **1.4 Notation, Probability and Jargon**
  The maths toolkit: vectors, matrices and index notation, expectation and variance with a worked example, and the ML vocabulary you will meet throughout the course.

How it connects

- → **Compute the gradients** (Step 6): Derivations depend on clear index notation

Previous: Universal approximation Back to the map

Online: airesearchlabs.si/learn/dl/map/f-maths