AI Research Labs

Section 1 of 11 · 4 lessons

Foundations of Deep Learning

What deep learning is, the types of learning, the notation, and how gradient descent minimises an objective.

1.1 Machine Learning and Deep Learning

1.2 Types of Learning

1.3 Gradient Descent and Momentum

1.4 Notation, Probability and Jargon



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.

1

BUILDING AND TRAINING A NETWORK · STEP 1

Frame the task

What should the network predict, and from what?

Lessons here: 1.1, 1.2 · feeds into Choose the output and loss

7

BUILDING AND TRAINING A NETWORK · STEP 7

Update the weights

How do I turn gradients into a better network?

Lessons here: 1.3 · feeds into Evaluate honestly

◆

UNDERNEATH THE MAP · FOUNDATION

Notation, probability and jargon

Lessons here: 1.4 · feeds into Compute the gradients

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.

A model, and learning its parameters

Machine learning (ML) is model building. A **model** is a compact mathematical description of how inputs relate to outputs. **Learning** is the procedure that finds good values for the model's **parameters** from examples, instead of a person writing every rule by hand.

$$\hat{y} = mx + c$$

Here x is the input, $\hat{y}$ the prediction, and m and c are the parameters. If learning finds $m = 5$ and $c = 30$, the model predicts $\hat{y} = 5 \cdot 6 + 30 = 60$ for $x = 6$. The *form* of the model (a straight line) is chosen in advance; the data decide its parameters.

The learning loop

Building a model is a loop, not a single step:

> **Data → Model → Learning → Evaluation** → back to the start until performance is acceptable.

Evaluation is part of the loop because a model that fits its training examples can still fail on new ones (Section 8).

Where deep learning fits

Deep learning (DL) is the part of ML that uses **neural networks with many layers**. Each layer transforms its input into a new representation, and later layers build on earlier ones, so the network learns useful features itself instead of relying on hand-made ones.

KEY DISTINCTION

ML is the broad field. DL is ML with multi-layered neural networks. Not every ML model is a neural network (decision trees and linear regression aren't), and not every neural network is deep.

APPLICATION AREA	EXAMPLES
Prediction	Weather, demand, prices
Pattern recognition	Recognising and segmenting objects in images
Sequential decision-making	Games, robotics, control
Generative AI	Generating text and images

A short history

WHEN	MILESTONE	WHY IT MATTERS
1943	McCulloch and Pitts model a neuron as a logic circuit	The first mathematical abstraction of a neuron
1959	ADALINE (Widrow and Hoff)	An early real application: cancelling echoes on phone lines
1986	Backpropagation popularised	Made training multi-layer networks practical
2000s	GPUs	Enough computation for much larger networks
2010	ReLU popularised	Eased training of deep networks (Section 4)
2012	A deep CNN wins ImageNet by a wide margin	Convolutional networks take over computer vision (Section 10)

Common doubts

FIRST PASS **Is a model the same thing as the learning algorithm?**

No. The **model** is the mathematical mapping, such as $\hat{y} = mx + c$ or a neural network with its layers. The **learning algorithm** (for example gradient descent) is the procedure that adjusts the model's parameters to fit the data. The same model can be trained by different algorithms.

GOING DEEPER **If deep learning is just machine learning, why does it get its own name?**

Because of what the layers do. Classic ML often needs people to design the input features. A deep network learns a **sequence of representations** itself: early layers find simple patterns and later layers combine them. That ability to learn features is what made DL so effective on images, audio and text.

Cheatsheet

ML vs DL

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/foundations/machine-learning-and-deep-learning

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.

Four ways to learn

TYPE	TRAINING INFORMATION	GOAL AND EXAMPLES
Supervised	Labelled pairs $(x^{(i)}, y^{(i)})$	Learn the mapping $x \to y$. **Classification** predicts a category; **regression** predicts a number.
Unsupervised	Inputs only, no labels	Find structure: clustering, anomaly detection, density estimation.
Semi-supervised	A few labelled examples, many unlabelled	Use the unlabelled data to help: name one face in a photo cluster and the rest follow.
Reinforcement	An agent acts and receives rewards or penalties	Learn a policy that maximises reward: games, robotics, control.

Notation for training data

A supervised training set has N examples, written $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$. The superscript in brackets, (i), is an **index** that picks out example i, not a power. $x^{(i)}$ is usually a vector of features and $y^{(i)}$ its label.

Classification or regression?

Look at the target, not the inputs:

- "Will this customer leave? (yes/no)" and "which digit is this?" are **classification**: a finite set of labels.
- "What grade will a student get from their hours of study?" is **regression**: a number.

The type of target decides the output layer and the loss of a network (Section 6).

> SCOPE
>
> This course covers supervised learning with neural networks. Reinforcement learning needs extra machinery (Markov decision processes) and is outside its scope; know the high-level distinction.

Common doubts

FIRST PASS **Does $x^{(2)}$ mean x squared?**

No. In ML notation $x^{(2)}$ means "the second training example". A power would be written without brackets, x^2. Mixing the two up is a common source of mistakes in derivations, so it's worth writing a small legend of what each index means.

Cheatsheet

Learning types

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/foundations/types-of-learning

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.

Training is optimisation

Learning means choosing parameters that make an objective (a **cost** or **loss**) as small as possible:

$$\min_{x \in \mathbb{R}^d} f(x)$$

Here x is the vector of parameters and f the objective. At an interior minimum of a differentiable function, the gradient is zero: $\nabla f(\hat{x}) = 0$. That condition is **necessary but not sufficient**: the gradient is also zero at a maximum, at a **saddle point** and on a flat plateau.

The gradient descent update

The gradient ∇f points in the direction of steepest local *increase*, so step the other way:

$$x_{k+1} = x_k - \eta \, \nabla f(x_k)$$

η (eta) is the **learning rate**, a hyperparameter you choose.

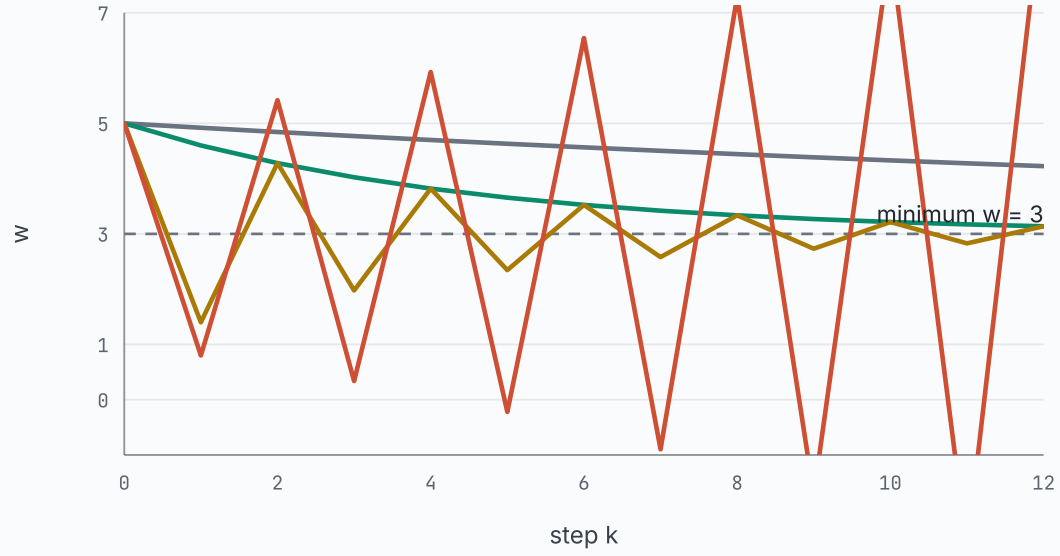
WORKED EXAMPLE

Minimise $f(w) = (w-3)^2$, whose minimum is at $w = 3$. The derivative is $f'(w) = 2(w-3)$. Start at $w_0 = 5$ with $\eta = 0.1$:

- $w_1 = 5 - 0.1 \cdot 2(5-3) = 5 - 0.4 = 4.6$
- $w_2 = 4.6 - 0.1 \cdot 2(1.6) = 4.6 - 0.32 = 4.28$
- $w_3 = 4.28 - 0.1 \cdot 2(1.28) = 4.024$

Each step multiplies the distance to the minimum by $1 - 2\eta = 0.8$: $w_k - 3 = 2 \cdot 0.8^k$.

What the learning rate does



Gradient descent on $f(w) = (w - 3)^2$ from $w = 5$. The distance to the minimum is multiplied by $(1 - 2\eta)$ each step: shrinking for $0 < \eta < 1$, oscillating when the factor is negative, and growing when $\eta > 1$.

- **Too small:** progress is very slow.
- **Too large:** updates overshoot, oscillate or diverge.
- **Narrow valleys** (steep one way, shallow the other) make plain gradient descent zig-zag.
- **Local minima, saddle points and plateaus** can stall it.

Momentum

Momentum adds a fraction α of the previous step:

$$x_{k+1} = x_k - \eta \nabla f(x_k) + \alpha \Delta x_k, \qquad \Delta x_k = x_k - x_{k-1}$$

Steps in a consistent direction build up speed, while zig-zag components cancel out. This helps on badly conditioned objectives. **RMSprop** and **Adam** go further by adapting the step size for each parameter; Adam is a common default in practice.

Common doubts

FIRST PASS **Why do we subtract the gradient instead of adding it?**

The gradient points uphill: it's the direction in which f increases fastest. We want f to **decrease**, so we move the opposite way. In the example, at $w = 5$ the derivative is $+4$ (the function rises to the right), so we move left, towards 3.

GOING DEEPER **If the gradient is zero, have we found the best parameters?**

Not necessarily. A zero gradient only says the surface is locally flat. It could be a minimum, a maximum, a **saddle point** (up in one direction, down in another) or a plateau. In high-dimensional neural network losses, saddle points and plateaus are common, which is one reason momentum and Adam help.

FIRST PASS **What exactly is an objective function?**

It is the single number that training tries to make as small as possible, written as a function of the parameters. The data is held fixed; only the parameters change. In deep learning it is usually the average loss over the training examples, sometimes plus a penalty on the weights.

In the worked example, $f(w) = (w - 3)^2$ is the objective and w is the parameter. At $w = 5$ its value is $f(5) = 4$; at $w = 3$ it is 0, the best possible.

FIRST PASS **What is a gradient?**

With one parameter, it is the derivative: the slope of the objective at the current point, meaning how much f changes for a small change in w. At $w = 5$, $f'(5) = 2(5 - 3) = 4$, so nudging w up by 0.01 raises f by about 0.04.

With many parameters, the gradient ∇f is the vector of partial derivatives, one per parameter. Together they point in the direction of steepest increase, and the length of the vector says how steep the surface is there. A network with a million weights has a gradient with a million entries.

FIRST PASS **What is the difference between the objective's value and its gradient?**

The **value** says how bad the parameters are right now: the height of the surface. The **gradient** says which way is downhill and how steep it is: the slope. For $f(w) = (w - 3)^2$:

- $w = 4$: value $f(4) = 1$, gradient $f'(4) = 2$.
- $w = 1$: value $f(1) = 4$, gradient $f'(1) = -4$.
- $w = 3$: value 0, gradient 0.

$w = 1$ and $w = 5$ have the same value, 4, but opposite gradients, because they sit on opposite sides of the minimum. The update uses only the gradient; the value is what you watch to see whether training is working.

FIRST PASS **What do a positive and a negative gradient tell me?**

The sign says which way the objective rises. **Positive**: f goes up as w goes up, so decrease w. **Negative**: f goes down as w goes up, so increase w. Subtracting $\eta \nabla f$ does the right thing automatically.

At $w = 1$: $f'(1) = -4$, so $w_1 = 1 - 0.1 \times (-4) = 1.4$, a step to the right, towards 3. The size of the gradient sets the size of the step, which is why steps shrink as you approach the minimum.

GOING DEEPER **What is the difference between a minimum, a maximum, a saddle point and a plateau?**

At all four the gradient is zero, or nearly zero, so the gradient alone can't tell them apart. The shape around the point can:

- **Minimum**: every direction goes up. $f(w) = (w-3)^2$ at $w = 3$.
- **Maximum**: every direction goes down. $-(w-3)^2$ at $w = 3$. Gradient descent moves away from it at the slightest push.
- **Saddle point**: up in some directions, down in others, like a horse's saddle. $f(x, y) = x^2 - y^2$ at $(0, 0)$. Common in high-dimensional network losses.
- **Plateau**: a wide, almost flat region. The gradient is tiny everywhere, so steps are tiny and training looks stuck, for example when sigmoid units saturate.

The curvature (second derivative) separates them: positive in every direction means a minimum, negative in every direction a maximum, mixed signs a saddle.

GOING DEEPER **Why does $eta = 1$ make the example oscillate?**

Each update multiplies the distance to the minimum by $1 - 2\eta$: $w_{k+1} - 3 = (1 - 2\eta)(w_k - 3)$. With $\eta = 1$ the factor is -1: the distance stays the same and only the side flips.

From $w_0 = 5$: $w_1 = 5 - 1 \times 4 = 1$, then $w_2 = 1 - 1 \times (-4) = 5$, and so on. It bounces between 5 and 1 forever and never converges.

- $0 < \eta < 0.5$: approaches 3 steadily from one side.
- $\eta = 0.5$: lands on 3 in one step.
- $0.5 < \eta < 1$: overshoots each time but closes in (the $\eta = 0.9$ curve in the chart).
- $\eta > 1$: overshoots further each time and diverges.

The 2 is the curvature, $f''(w) = 2$. In general a step size above $2/\text{curvature}$ diverges.

GOING DEEPER **How do GD, SGD, momentum, RMSprop and Adam differ?**

All of them use the same gradient from backpropagation. They differ only in how they turn it into a step.

- **Gradient descent** (batch): the gradient over the whole training set each step. Exact, but every step is expensive.
- **SGD** (stochastic or mini-batch): the gradient on one example or a small batch. A noisy estimate, but many cheap steps, and the noise can help escape saddle points.
- **Momentum**: adds a fraction of the previous step. Speeds up along a consistent direction and damps zig-zag.
- **RMSprop**: divides each parameter's step by a running average of its recent squared gradients. Parameters with large gradients take smaller steps; flat directions take larger ones.
- **Adam**: momentum and RMSprop together, with a correction for the first few steps. The usual default, often with $\eta = 10^{-3}$.

FIRST PASS

What exactly is a point on the objective's graph?

One choice of parameters and how good it is. The horizontal position is the parameter value, w; the height is the objective at that value, $f(w)$. The point $(5, 4)$ on $f(w) = (w - 3)^2$ means "with $w = 5$, the objective is 4".

Training starts at one point and moves along the curve, one update at a time, towards the lowest point. With two parameters the graph is a surface and a point is a spot on it; with a million parameters you can't draw it, but the idea is the same.

FIRST PASS

How does momentum actually help?

Plain gradient descent only looks at the slope where it stands now. In a long narrow valley the slope points mostly across the valley, so it zig-zags from wall to wall and creeps along the floor.

Momentum keeps a running memory of past steps. The side-to-side parts point in opposite directions on alternate steps and cancel out; the part along the valley points the same way every time and builds up. The result: fewer wasted steps, faster progress along gentle slopes, and enough speed to roll through small bumps and flat stretches.

FIRST PASS

How do I know when gradient descent has converged, and when to stop?

Plot the objective against the iteration number: the **convergence curve**. A healthy curve falls steeply at first, then flattens. It has converged when it stops falling: the change per step, or the size of the gradient, drops below a small tolerance such as 10^{-6}. A curve that zig-zags or climbs means the learning rate is too large.

For neural networks, don't wait for the training loss to flatten. Watch the **validation** loss and stop when it stops improving: that is **early stopping** (lesson 9.2). In practice you also cap the number of epochs.

Cheatsheet

Gradient descent

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

Learning rate

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/foundations/gradient-descent-and-momentum

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.

Notation

SYMBOL	MEANING
x, w	Lower case: vectors (column vectors by default)
W, X	Upper case: matrices
$w^\top$	Transpose: the column vector w written as a row
w_i	Subscript: component i of w
$x^{(i)}$	Training example i (an index, not a power)
$\partial J / \partial w$	Partial derivative of the objective J with respect to w
∇J	The gradient: all partial derivatives of J collected in a vector

The four tools you need are **linear algebra** (layers are matrix products), **calculus** (especially the chain rule, for backpropagation), **optimisation** (gradient descent) and **probability** (outputs as probabilities, losses as likelihoods).

Expectation and variance

A discrete random variable has a probability mass function whose probabilities sum to 1; a continuous one has a density, and probabilities are areas under it.

$$\mathbb{E}[X] = \sum_i x_i \, p(x_i), \qquad \mathrm{Var}(X) = \mathbb{E}\big[(X-\mu)^2\big], \ \mu = \mathbb{E}[X]$$

WORKED EXAMPLE

X is 0 with probability 0.75 and 10 with probability 0.25.

- $\mathbb{E}[X] = 0(0.75) + 10(0.25) = 2.5$
- $\mathrm{Var}(X) = 0.75(0-2.5)^2 + 0.25(10-2.5)^2 = 4.6875 + 14.0625 = 18.75$
- Standard deviation $= \sqrt{18.75} \approx 4.33$, in the same units as X.

Vocabulary

TERM	MEANING
Parametric model	Described by a fixed set of parameters, such as $y = mx + c$ or a network's weights
Hyperparameter	A setting chosen outside gradient-based learning: learning rate, number of layers, width, dropout rate
Bias	Two meanings: the bias term b of a neuron, or systematic error (the bias–variance trade-off). A third, social bias, is about unfair outcomes for groups.
Overfitting	Fitting the training examples, noise included, so closely that new examples are predicted badly
Regularisation	Any change to learning meant to improve generalisation rather than training error (Section 9)

Common doubts

FIRST PASS **What is the difference between a parameter and a hyperparameter?**

Parameters (weights and biases) are learned by gradient descent. **Hyperparameters** are choices made around that process: the learning rate, how many layers and units, the batch size, the dropout rate. They are chosen using validation data (Section 8), not by gradient descent.

FIRST PASS **What is the difference between parametric and non-parametric models?**

A **parametric** model has a fixed shape described by a fixed number of parameters, however much data you have. Linear regression is always $y = \beta_0 + \beta_1 x_1 + \ldots$: more data improves the βs but doesn't add new ones. A neural network is parametric too: its size is fixed by the architecture.

A **non-parametric** model's complexity grows with the data. **K-nearest neighbours** keeps every training example and predicts from the closest ones, so the "model" is the dataset itself.

GOING DEEPER **Does non-parametric mean the model has no parameters?**

No. It means there is no **fixed, finite** set of parameters that defines the model. K-nearest neighbours still has settings such as K and the distance measure, but those are hyperparameters you choose, not values learned from data. What grows with the data is everything the model must store: all n training points.

Cheatsheet

Expectation and variance

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

Index, not power

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

Online version with the interactive quiz: airesearchlabs.si/learn/dl/foundations/notation-probability-and-jargon