Deep Learning

How neural networks are structured -- layers, weights, activation functions -- how training works via backpropagation, and a brief look at CNN/RNN/transformer architectures.

Beginner 18 min

Last updated: August 22, 2026
Read online: https://www.learnforgex.com/en/topics/deep-learning

---

Deep Learning

The "Machine Learning" lesson ended by pointing at one specific family of learning algorithms that turned out to scale dramatically better than the others once enough data and computing power became available: **neural networks**, and specifically *multi-layered* ones. That family is Deep Learning -- the third circle on the map from "What Is Artificial Intelligence?", nested inside Machine Learning, which is nested inside AI. Deep Learning isn't a replacement for anything you learned in the previous lesson (training, inference, overfitting, the training/test split) -- it's a specific, extremely successful technique for doing all of it. This lesson keeps things conceptual and deliberately avoids the underlying math; the goal is an accurate mental model of how a neural network is structured and why "depth" matters, not the calculus behind it.

What Is Deep Learning?

Deep Learning is a subset of Machine Learning that uses **neural networks** -- structures loosely inspired by how biological neurons connect to one another -- as the mechanism for learning from data. "Deep" refers specifically to networks with many layers stacked on top of each other (you'll see exactly what a layer is in the next section). A "shallow" neural network with just one or two layers has existed since the 1950s; what makes *deep* learning distinct, and what gives this lesson its name, is stacking many such layers together, which turns out to let a network learn far more complex patterns than a shallow one can.

Neurons, Layers, and Networks: The Basic Building Blocks

A neural network is built from three kinds of layers, each made up of many simple units often called "neurons" or "nodes":

- **Input layer:** receives the raw features of an example (recall "features" from the "Machine Learning" lesson) -- for an image, this might be the brightness value of every pixel; for a sentence, a numeric representation of each word.
- **Hidden layer(s):** one or more layers between input and output, where the actual learning happens. Each node in a hidden layer takes the outputs of the previous layer, combines them, and passes a new value forward. A network is called "deep" specifically because it has many of these hidden layers stacked in sequence.
- **Output layer:** produces the network's final answer -- a probability that an image is a cat, a predicted house price, the next word in a sentence.

Data flows through the network from input, through each hidden layer in turn, to output -- this is called a **forward pass**, and it's what happens during inference (see "Training vs. Inference" in the previous lesson) once the network is already trained.

What Each Node Actually Does

Without getting into the underlying math, it helps to know the shape of what happens inside a single node: it takes the values coming in from the previous layer, combines them using adjustable numbers called **weights** (each connection between two nodes has its own weight, roughly representing "how much this input matters"), usually adds a **bias** -- a fixed number that belongs to the node itself, independent of its inputs -- and passes the result through a small mathematical function called an **activation function**. ("Parameters" doesn't mean only weights -- weights and biases together are what people mean by a network's parameters; this lesson won't go deeper into bias, just that it exists.) The activation function's job is to let the network represent *non-linear* patterns -- relationships that can't be captured by a simple straight-line combination of inputs, which describes almost every interesting real-world pattern (there's no straight-line formula for "is this a picture of a cat"). You don't need to know the formulas for specific activation functions to understand deep learning at this level -- just that this step is what allows a network to learn far more than "a weighted sum of its inputs."

Here's a tiny, concrete example -- not to teach math, just to make the "input + weight + activation" logic visible: say a node takes two inputs, `x1 = 2` and `x2 = 1`, with weights `w1 = 3` and `w2 = -1`. The node first computes the weighted sum: `(2 × 3) + (1 × -1) = 5`. It then passes that through an activation function -- using the common, simplified ReLU (rounds negative values to 0, leaves positive ones unchanged) as an example, 5 is already positive, so the result stays 5. In a real network the only difference is scale: thousands of nodes, millions of weights -- but what each one does is exactly this same few-line computation.

How a Network Learns: Loss and Backpropagation

Training a neural network follows the same training/inference split from the "Machine Learning" lesson, with a specific mechanism for the training half:

1. The network makes a forward pass on a training example and produces a prediction.
2. A **loss function** measures how wrong that prediction was, compared to the correct label (recall "labels" from the previous lesson) -- a single number where lower means "closer to correct."
3. **Backpropagation** works backward through the network, figuring out how much each individual weight contributed to that error.
4. Each weight is nudged slightly in the direction that would have reduced the error -- a process called **gradient descent**.
5. This repeats over and over, across many examples, until the network's predictions get reliably better.

That's the entire training loop, conceptually. This course deliberately doesn't derive the calculus behind backpropagation -- the important idea to take away is that training a deep network means repeating this predict-measure-error-adjust cycle millions of times, which is precisely why training is the slow, computationally expensive phase (see "Training vs. Inference" in the previous lesson) and why it depends so heavily on specialized hardware (GPUs, built for exactly this kind of repeated numerical computation).

Why "Deep"? The Role of Many Layers

Each layer in a network tends to learn a progressively more abstract representation of the input, building on the layer before it. A classic, intuitive example from image recognition: early layers might learn to detect simple edges and color gradients; middle layers combine those edges into shapes like curves and corners; later layers combine shapes into recognizable parts (an eye, an ear); the final layers combine parts into whole concepts ("cat"). No one explicitly programs a layer to look for edges or ears -- this hierarchy of representations emerges naturally from the training process described above. Stacking many layers is what makes this progressive abstraction possible; a shallow network with only one or two layers simply doesn't have enough steps to build up from raw pixels to a concept as abstract as "cat." It's worth flagging that this edge → shape → eye/ear → cat story is an *intuitive* illustration, not a guarantee -- a real network's layers generally do learn progressively more abstract patterns, but there's no guarantee any specific layer ends up learning a clean, human-nameable concept like "edge" or "ear."

> When you hear that a model has "billions of parameters," those parameters are almost entirely the weights and biases described in "What Each Node Actually Does" -- one number per connection between two neurons, and one per node, multiplied across every layer of a very deep, very wide network. More parameters roughly means more capacity to represent complex patterns, though it also means more data and compute are needed to train them well.

Common Types of Neural Networks

Different network *architectures* -- different ways of arranging layers and connections -- are suited to different kinds of data. Three are worth knowing by name, even at a purely conceptual level, because they'll come up again later in this course:

- **Convolutional Neural Networks (CNNs):** specialized for grid-like data such as images -- their layers are structured to detect local patterns (like the edges and shapes from the previous section) regardless of where in the image they appear.
- **Recurrent Neural Networks (RNNs):** a historically important architecture designed for sequential data (text, audio, time series), where a network processes one element at a time while keeping a form of "memory" of what came before it. RNNs were the standard approach for language tasks before transformers took over, and are now largely superseded by them.
- **Transformers:** a newer architecture (introduced in 2017) that models the relationships between tokens -- how much each part of the input matters to every other part -- using a mechanism called *attention*. Unlike RNNs, this lets a transformer compute those relationships largely in parallel during training rather than one element at a time in sequence, which is a big part of why they scale so much more efficiently on huge datasets. Transformers are the architecture behind essentially every large language model covered later in this course -- this lesson won't go deeper into how they work, but it's worth knowing the name now, since "Large Language Models" builds directly on it.

Why Deep Learning Took Off in the 2010s

"What Is Artificial Intelligence?" mentioned that deep learning started dramatically outperforming older techniques around 2012, once large labeled datasets and powerful GPUs became available -- this lesson can now explain *why* those two things mattered so much. Deep networks with many layers and millions (or billions) of weights need enormous amounts of training data to learn reliable patterns instead of overfitting (see "Overfitting and Underfitting" in the previous lesson), and the internet-scale datasets that became available in the 2000s and 2010s provided exactly that. Meanwhile, the repeated numerical computation described in "How a Network Learns: Loss and Backpropagation" -- millions of small adjustments across millions of weights -- happens to be precisely the kind of parallel computation that GPUs (originally built for rendering video game graphics) are extremely good at. Neither ingredient was new by itself; what changed around 2012 was having both at a large enough scale simultaneously.

Best Practices

- Don't reach for deep learning by default. It typically needs far more data and compute than other machine learning techniques (see "Machine Learning") to perform well -- for smaller or simpler problems, a non-deep-learning approach can be faster to build, cheaper to run, and easier to understand.
- When picking an architecture, match it to the kind of data: CNNs for grid-like/image data, RNNs or transformers for sequential/text data (see "Common Types of Neural Networks") -- using the wrong shape of network for your data usually means worse results for more effort.
- Remember that everything from "Overfitting and Underfitting" in the previous lesson still applies, and often applies *more* -- deep networks have enormous numbers of weights, which gives them more capacity to simply memorize training data if it's too small or repetitive.

Common Mistakes

- **Assuming "deep learning" and "AI" are interchangeable.** As the map in "What Is Artificial Intelligence?" showed, deep learning is a specific technique nested inside machine learning, nested inside AI -- useful, well-established AI exists that involves no neural networks at all.
- **Thinking a network "understands" what a hidden layer is doing.** The layered representations described in "Why 'Deep'? The Role of Many Layers" emerge automatically from training -- nobody designs a specific layer to detect "ears" by hand, and the exact patterns any given layer ends up encoding are usually not directly human-readable.
- **Assuming more layers or more parameters always means a better model.** Beyond a certain point, more capacity without enough matching data mostly increases the risk of overfitting (see the previous lesson) rather than improving real-world performance.

Summary, Cheat Sheet, and Glossary

Summary

- Deep Learning is Machine Learning using neural networks with many stacked hidden layers -- "deep" refers to that stack of layers.
- A network is built from an input layer, one or more hidden layers, and an output layer; data moves through it in a forward pass.
- Each node combines its inputs using adjustable weights and an activation function, which is what lets a network represent complex, non-linear patterns.

- Training repeats a predict-measure-error-adjust loop (forward pass, loss, backpropagation, gradient descent) across many examples -- this is why training deep networks is slow and GPU-hungry.
- Layers tend to learn increasingly abstract representations of the input; stacking many layers is what makes this progressive abstraction possible.
- CNNs, RNNs, and transformers are three common architectures for different kinds of data -- transformers, in particular, are the basis of every large language model covered later in this course.

Cheat Sheet

- Neural network = input layer + hidden layer(s) + output layer.
- Weight = an adjustable number on a connection between two nodes.
- Activation function = what lets a node represent non-linear patterns.
- Loss function = measures how wrong a prediction was.
- Backpropagation + gradient descent = how weights get adjusted during training.
- CNN = images. RNN = sequences (older approach). Transformer = sequences (modern approach, basis of LLMs).

Glossary

- **Neural network:** a learning structure made of layers of simple connected units ("neurons"), loosely inspired by biological neurons.
- **Layer:** a group of neurons that all receive input from the previous layer and pass output to the next one (input, hidden, or output).
- **Weight:** an adjustable number representing the strength of a connection between two neurons.
- **Bias:** an adjustable, fixed number that belongs to a node itself, independent of its inputs -- together with weights, it makes up a network's parameters.
- **Activation function:** a function applied inside a node that lets a network represent non-linear patterns.
- **Forward pass:** running an example through the network from input to output to produce a prediction.
- **Loss function:** measures how far a prediction is from the correct answer.
- **Backpropagation:** the algorithm that works backward through a network to determine each weight's contribution to the error.
- **Gradient descent:** the process of nudging weights in the direction that reduces the loss.
- **CNN / RNN / Transformer:** three neural network architectures suited to grid-like data, sequential data, and (in the transformer's case) sequential data processed all at once via attention.