Recap: Neural Networks I — How Learning Works

DS701 Session 19 — Mon Nov 9, 2026

Today’s plan

Backprop by hand — the recipe from this lecture — is examined on **Quiz 3**; **HW9** will practise it. Today’s activity is the first rep.

Knowledge-Check Review

KC 1: The recipe

State the backpropagation recipe from the lecture: what runs first and why, what the output node's gradient is set to, and what each of the three node types (`+`, `*`, `ReLU`) does with the gradient that arrives from its parent.

KC 2: The chain rule by hand

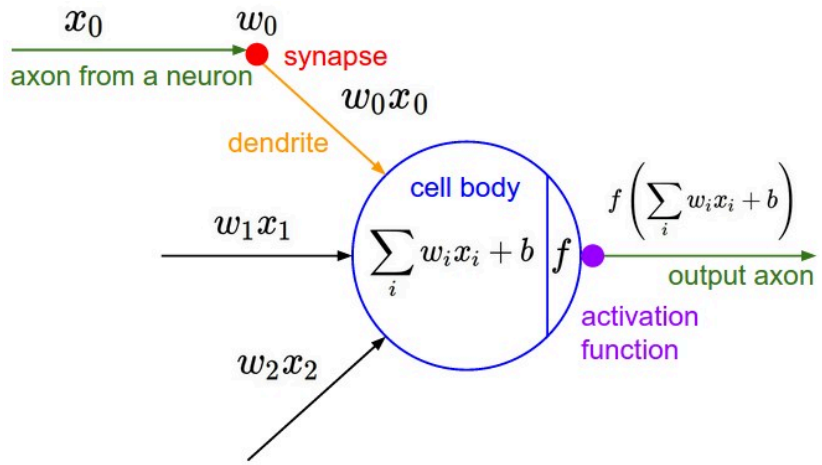
Take the lecture's three-stage graph, $d = a \cdot b$, $e = d + c$, $L = e \cdot f$, but with new values $a = 2$, $b = -1$, $c = 5$, $f = 3$. Run the forward pass, then propagate backwards to find $\partial L / \partial c$ and $\partial L / \partial b$. Show the chain of local derivatives you multiplied for $\partial L / \partial b$.

KC 3: Why gradient descent, and why mini-batches

In lecture 10 we solved linear regression in closed form (the normal equations). An MLP with an MSE loss also has a formula for its loss — so why do we train it by gradient descent instead of solving for the weights, and once we have chosen gradient descent, why do we compute the gradient on a mini-batch rather than on the whole training set every step?

Highlights

A neuron, a layer, a network — one composed function



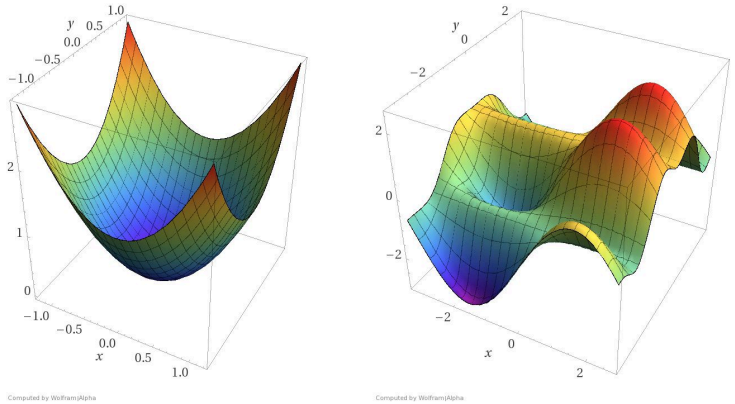
$$\text{out} = f\Big(\sum_i w_i x_i + b\Big)$$

Training is a loop: loss → gradient → update

1. **Loss** — a number that says how wrong the network is on the data: MSE $\frac{1}{N}\sum(\hat{y}_i - y_i)^2$, or cross-entropy for classes.
2. **Gradient** — $\nabla_\theta L$: how the loss changes with *every* parameter.
3. **Update** — step downhill:

$$\theta \leftarrow \theta - \eta \nabla_\theta L$$

4. Repeat until the loss stops falling.



Non-convex surface: no closed form — you *walk*.

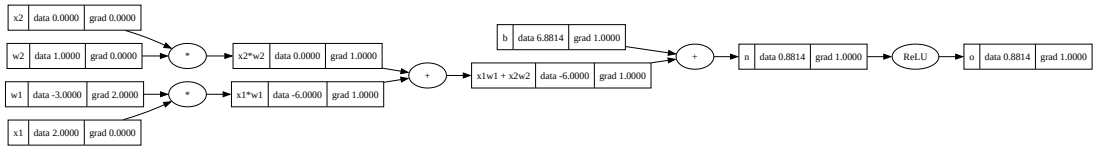
Where do the gradients come from?

$$L = \ell\Big(\, \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\, f(\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\, f(\cdots f(\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}) \cdots)),\ y\Big)$$

The chain rule, node by node

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial e} \cdot \frac{\partial e}{\partial c} \qquad \text{— gradient from above} \times \text{local derivative, both at the f}$$

The recipe on a real neuron

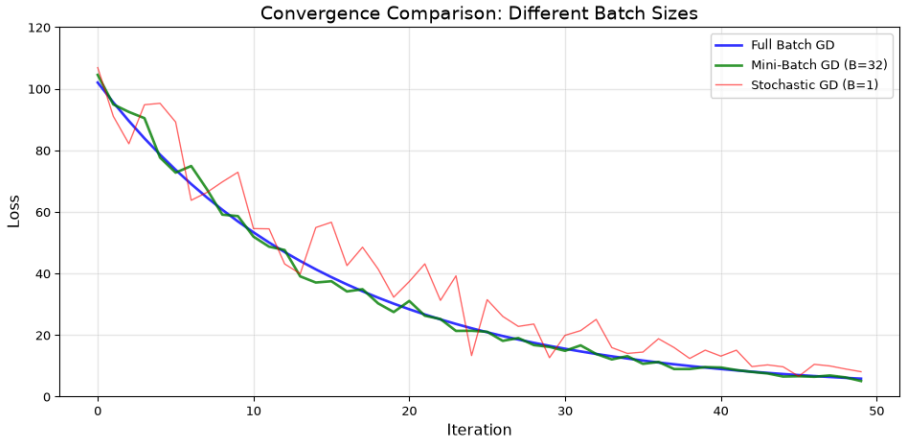


$$o = \mathrm{ReLU}(x_1 w_1 + x_2 w_2 + b),$$
with $x_1 = 2,\ x_2 = 0.$

Why *stochastic* gradient descent

Backprop gives $\nabla_{\mathbf{w}} \ell_i$ for **one** sample. How many before a step?

	per step	gradient	noise
full batch	all N	exact	none
SGD ($B = 1$)	1	very noisy	lots
mini-batch	$B \approx 32$–256	good enough	some



(illustration, not a real training curve)

In-Class Activity

Activity: backprop by hand, then by machine — and what the gradients tell you

Goal: train a single neuron with a hand-derived gradient; build a minimal `Value` autograd class (you write `backward` for `*` and `ReLU`) and check it against finite differences; use it to train a two-layer MLP on your section's toy dataset. Then the stretch: send gradients back through a deeper net with sigmoid / tanh / ReLU and watch which ones vanish, and see with your own eyes why all-zeros and too-large initialisations fail.

- Work in groups of 2–3. Open **your section's** notebook.
- Parts marked **(autograded)** are submitted to Gradescope; the rest is participation.
- Staff will circulate — be ready to explain any part of your work.
- Dependencies: `numpy`, `matplotlib`, `scikit-learn` (datasets only). No PyTorch.

Section A1 

Going deeper

Full lecture notes: Neural Networks I: How Learning Works

- The Key Insight — a sigmoid neuron is logistic regression; what layers add
- MLP Mathematical Formulation — the network as a composed function of θ
- Gradient Descent Algorithm and Learning Rate Matters
- Building the `Value` Class — the graph builds itself, version by version
- Manual Gradient Calculation → The Chain Rule → The Backpropagation Recipe
- Backprop on a Neuron — the worked ReLU neuron
- Mini-Batch Gradient Descent
- The full framework — automatic `backward()`, gradient accumulation, layers and a training loop: NN II — Compute Graph and Backpropagation
- Next session: making training work with scikit-learn — Neural Networks II