PyTorch Workshop 1_From Gradient Flow to Optimisation Intuition_Fresh_v1.0

February 6, 2026

1 Workshop 1: From Gradient Flow to Optimisation Intuition

This workshop builds on Tutorials 1–4 and closes **Part 1** of the series, consolidating core ideas about PyTorch `autograd` and gradient flow before moving on to explicit optimisation in Workshop 2 and Part 2.

Rather than treating gradients as black-box training signals, the workshop emphasises **gradients as sensitivity measures**, especially in settings where model outputs are **tensor-valued** and gradients are computed using **explicit upstream directions**.

The problems are designed to shift perspective: - from *"what gradient does PyTorch give me?"* - to *"what does this gradient represent, and why?"*

The emphasis is on developing intuition for: - how gradients flow through structured linear and nonlinear computations, - how upstream gradients select directions in output space, - how `.grad` reflects sensitivity rather than optimisation, - and how gradient structure anticipates optimisation behaviour.

Key ideas explored include: - vector–Jacobian products as the fundamental object computed by `backward(v)`, - the role of upstream gradients in shaping gradient flow, - sparsity and structure in gradients induced by linear maps and nonlinearities (e.g., ReLU), - interpreting gradients statistically and geometrically rather than procedurally, - and using controlled experiments to reason about gradient behaviour.

This workshop serves as a conceptual bridge between: - `autograd` mechanics and gradient flow (Tutorials 3–4), - and **objective design and optimisation dynamics** (Workshop 2 and Part 2).

The focus is intentionally *not* on training pipelines, datasets, or optimisers, but on understanding how gradients behave in controlled settings—laying the groundwork for effective optimisation.

Recommended prerequisites: - Familiarity with PyTorch tensors and `.backward()` - Understanding of scalar vs tensor-valued gradients - Basic comfort with linear algebra and nonlinear activations

---

Author: Angze Li

Last updated: 2026-02-06

Version: v1.0

1.1 Problem 1: Sensitivity of a Feature Transformation (Warm-up)

In many models, inputs are transformed into feature representations before being used by a downstream objective. Understanding which inputs influence which features is critical for interpretability and optimisation.

Consider the following setup:

```
x = torch.randn(6, requires_grad=True)

W = torch.tensor([
    [1.0,  0.0,  0.0, 0.0, 0.0, 0.0],
    [0.0,  1.0,  0.0, 0.0, 0.0, 0.0],
    [0.0,  0.0,  2.0, 0.0, 0.0, 0.0],
    [0.0,  0.0,  0.0, 3.0, 0.0, 0.0],
], requires_grad=False)

out = torch.relu(W @ x) # what does this line do?
```

This produces a 4-dimensional tensor output, not a scalar.

1.1.1 Task

1. Construct an upstream gradient vector `v` such that:
   - the last feature of out is weighted most heavily,
   - the first two features are ignored.
2. Call:

```
out.backward(v)
```

3. Inspect `x.grad`.

1.1.2 Questions to think about

- Which components of `x` receive non-zero gradients?
- How does the structure of `W` affect gradient flow?
- How does ReLU change which inputs are "active"?
- Why is `x.grad` sparse in some cases?

1.1.3 Hint (optional)

Think in terms of which inputs influence which outputs, and which outputs are being emphasised by `v`.

1.1.4 Learning outcomes

After this problem, you should be comfortable with: - interpreting `.grad` as a sensitivity map, - reasoning about gradient flow through linear + nonlinear layers, - understanding how upstream weighting reshapes gradient influence.

This problem bridges Tutorials 3 and 4 cleanly and warms up the optimisation mindset.

1.2 Solution 1

[ ]: