PUBLIC LECTURE 6 CHEATSHEET

Reverse mode from local rules

Record the executed scalar graph, seed the loss, and propagate sensitivities backward.

Backprop is the reverse-sweep algorithm. Autograd is the engine that records operations and executes their backward rules.

1 · Derivative as a local model

Near u, a scalar function is approximately linear:

$$f(u+\Delta u) \approx f(u) + f'(u)\Delta u.$$

For many parameters, stack the partial derivatives:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \vdots \end{pmatrix}.$$

Gradient descent subtracts this vector. Reverse mode computes it efficiently when many inputs feed one scalar loss.

2 · One operation, one reusable rule

For $v = f(u)$, if $g_v = \partial \frac{\mathcal{L}}{\partial} v$ arrives from downstream:

$$\underbrace{g_u}_{\text{sent to } u} = \underbrace{g_v}_{\text{arriving at } v} \underbrace{\frac{\partial v}{\partial u}}_{\text{local slope}}.$$

Every operation needs only its local derivative and any forward values required to evaluate it.

3 · Scalar rule table

Forward	Backward contributions
$v = a + b$	$g_a \mathrel{+}= g_v; \quad g_b \mathrel{+}= g_v$
$v = ab$	$g_a \mathrel{+}= g_v b, \quad g_b \mathrel{+}= g_v a$
$v = u^2$	$g_u \mathrel{+}= g_v 2u$
$v = \varphi(u)$	$g_u \mathrel{+}= g_v \varphi'(u)$
$v = a - y$	$g_a \mathrel{+}= g_v, \quad g_y \mathrel{-}= g_v$

Here g_u is shorthand for $\partial \frac{\mathcal{L}}{\partial} u$; the subscript names the stored value.

4 · Branches must accumulate

If a value x influences the loss through several children $u_1, \ldots, u_k$:

$$g_x = \sum_{i=1}^{k} g_{u_i} \frac{\partial u_i}{\partial x}.$$

Use `+=`, not `=`. Overwriting one contribution silently drops every other path.

Example: $u = x^2$, $v = 3x$, $\mathcal{L} = u + v$. At $x = 4$:

$$g_x = 1(2x) + 1(3) = 8 + 3 = 11.$$

5 · Reverse-mode algorithm

1. run the forward program and create value nodes
2. store the operation and required operands on each result
3. topologically order the graph
4. initialize every gradient to zero
5. seed the scalar loss with $g_{\mathcal{L}} = 1$
6. visit operations in reverse topological order and accumulate

Why reverse mode for training?

Training maps millions of parameters to one scalar loss. One reverse sweep returns every parameter sensitivity together. Finite differences require two extra forward evaluations per checked coordinate; use them to debug, not to train.

Forward computes values. Backward computes how the final scalar loss responds to each stored value.

PUBLIC LECTURE 6 CHEATSHEET

Work one scalar graph end to end

The numbers below expose every stored value, local derivative, and accumulated gradient.

1 · Model, loss, and primitive graph

$$\hat{y} = wx + b, \quad \mathcal{L} = (\hat{y} - y)^2.$$

Use $(w, x, b, y) = (2, 3, 1, 10)$ and name the intermediate values:

$$m = wx, \quad a = m + b, \quad e = a - y, \quad \mathcal{L} = e^2.$$

This decomposition is the graph: leaves (w, x, b, y), stored values (m, a, e), and one scalar output $\mathcal{L}$.

2 · Forward pass

Value	Operation	Result
m	wx	6
a	$m + b$	7
e	$a - y$	-3
$\mathcal{L}$	e^2	9

Save operands such as x, w, e because the multiplication and square rules need them during backward.

3 · Backward pass

Seed $g_{\mathcal{L}} = 1$ and reverse the operations:

$$g_e = 1(2e) = -6.$$

$$g_a = -6, \quad g_y = (-6)(-1) = 6.$$

$$g_m = -6, \quad g_b = -6.$$

$$g_w = g_m x = -18, \quad g_x = g_m w = -12.$$

Final: $\partial \frac{\mathcal{L}}{\partial}(w, b, x, y) = (-18, -6, -12, 6)$.

4 · Read the sign before updating

Because $\partial \frac{\mathcal{L}}{\partial} w = -18$, a small gradient-descent step increases w:

$$w \leftarrow w - \eta(-18) = w + 18\eta.$$

With $\eta = 0.01$, update $w : 2 \to 2.18$ and $b : 1 \to 1.06$. The prediction changes from 7 to 7.60, toward the target 10.

5 · Autograd lifecycle

Concept	PyTorch
leaf value	`Tensor(requires_grad=True)`
creator rule	`grad_fn` on a non-leaf
reverse sweep	`L.backward()`
leaf derivative	`p.grad`
keep intermediate	`value.retain_grad()`
clear old gradients	`opt.zero_grad()`

Gradients accumulate by default. A second backward pass without clearing adds another contribution.

6 · Independent gradient check

For one coordinate θ_j, use the central difference

$$g_{\mathrm{FD}} = \frac{\mathcal{L}(\theta_j + \varepsilon) - \mathcal{L}(\theta_j - \varepsilon)}{2\varepsilon}.$$

Compare it with autograd in float64 at a few coordinates. Large disagreement usually means a graph, broadcasting, in-place mutation, nondifferentiability, or numerical-scale bug.

From-scratch implementation audit

- ✓ Every result remembers its parents and local backward rule.
- ✓ Backward order is reverse topological, not recursive guesswork.
- ✓ Branches accumulate with `+=`.
- ✓ Only a scalar output is seeded automatically.
- ✓ Tests cover shared inputs and repeated use of one value.