Debugging Guide & Best Practices 🐞

"If you don't audit the gradients, you are training on hope."

This document outlines the standard fixes for common pathologies in [PROJECT_TYPE] models.

---

1. The Physiology of Training (Weights & Gradients)

A. The Weight Update Ratio

We must ensure weights are actually moving, but not too fast.
$$ Ratio = \frac{\eta \cdot |\nabla W|}{|W|} $$

- **Target**: **1e-3** (The "Goldilocks Zone").
- **< 1e-5**: Model is frozen (Check Learning Rate, Detached Graphs).
- **> 1e-1**: Model is unstable (Check Clipping, Batch Size).

B. Gradient Norm Topology

- **Norm = 0.0**: **Broken Graph**. Check `requires_grad`, `.detach()`, or index operations.
- **Norm > 10.0**: **Exploding**. Use `clip_grad_norm_`.
- **Norm = NaN**: **Overflow**. Check for division by zero or `log(0)`.

---

2. Shape Hygiene 📐

Symptom: `RuntimeError: The size of tensor a (X) must match the size of tensor b (Y)`.

Best Practices

1. **Assert Inputs**:

```
B, T, D = x.shape
assert D == self.hidden_dim, f"Mismatch: {D} != {self.hidden_dim}"
```

2. **Use Einops**: Avoid `x.view()`. Use `rearrange(x, 'b t (h d) -> ...')` to be explicit.

---

3. Convergence Pathology 📉

1. **The "Hockey Stick"**: Loss drops instantly then flatlines.
   - *Diagnosis*: LR too high. Local Fast Basin.
2. **The "Slow Burn"**: Loss decreases linearly.
   - *Diagnosis*: LR too low.
3. **The "NaN Death"**: Loss becomes NaN at step N.

- *Diagnosis*: Optimization instability or Normalization Drift.

---

4. GPU/Hardware Tips 💻

- **OOM (Out of Memory)**: Reduce Batch Size, use Gradient Accumulation.
- **Slow Training**: Check CPU-GPU sync points (e.g. `print(tensor)` inside loop).

---

5. Agent Debugging Workflow 🤖

When you (an AI agent) encounter training issues, follow this decision tree:

Step 1: Gather Evidence (`view_file`, `run_command`)

```
# Check last 50 lines of log
tail -n 50 logs/train.log

# Check GPU memory
nvidia-smi

# Check if process is still running
ps aux | grep train
```

Step 2: Classify the Failure Mode

Symptom	Tools to Call	What to Check
Loss = NaN	`view_code_item`: Normalization layers, loss functions	Look for `log(0)`, division by zero, `sqrt(negative)`
Loss doesn't decrease	`view_code_item`: Optimizer init, learning rate schedule	Verify `requires_grad=True`, check LR value
OOM Error	`view_file`: Model config	Reduce `batch_size` or `hidden_dim`, use gradient checkpointing
Shape Mismatch	`view_code_item`: Forward pass where error occurs	Add `assert` statements for tensor shapes
Slow throughput	`grep_search`: Search for `.item()` or `print(tensor)` in training loop	Remove CPU-GPU sync points
MCP Server Not Found	Use **`view_file`** or **`view_file_outline`**	You tried to use `read_resource` or `list_resources` on a local path. Filesystem paths are not MCP servers.
Artifact Access Error	Use absolute path to `.gemini/antigravity/...`	Artifacts are local files. Access them via `view_file`.

Step 3: Implement Fix (`replace_file_content`)

Example: Loss = NaN at step 500

1. `view_file`: Check log for exact error line
2. `view_code_item`: View the layer where NaN occurs
3. **Hypothesis**: Entropy calculation has `log(0)`
4. `replace_file_content`: Add epsilon: `entropy = -torch.sum(p * torch.log(p + 1e-8))`
5. `run_command`: Relaunch training from checkpoint

Step 4: Document in Journal (`replace_file_content`)

Add to `journal.md`:

```
## Debugging Log

### [DATE] - NaN Loss at Step 500
**Symptom**: Loss became NaN during entropy calculation
**Root Cause**: Missing epsilon in log operation
**Fix**: Added `1e-8` epsilon to prevent `log(0)`
**Resolution**: Training resumed successfully from checkpoint
```

Step 5: Create Smoke Test (`write_to_file`)

Prevent regression:

```
# tests/test_entropy_stability.py
def test_entropy_no_nan():
    p = torch.tensor([0.0, 1.0])  # Edge case
    entropy = -torch.sum(p * torch.log(p + 1e-8))
    assert not torch.isnan(entropy), "Entropy should not be NaN"
```

Remember: Every bug you fix should generate:

1. A journal entry (documentation)
2. A test case (prevention)