FOUNDATION TIER

MODULE 04

Loss Functions & Stability

Measuring model performance and ensuring numerical stability

TINYTORCH FOUNDATION TIER

Module 04: Losses

The Mathematical Conscience of Learning

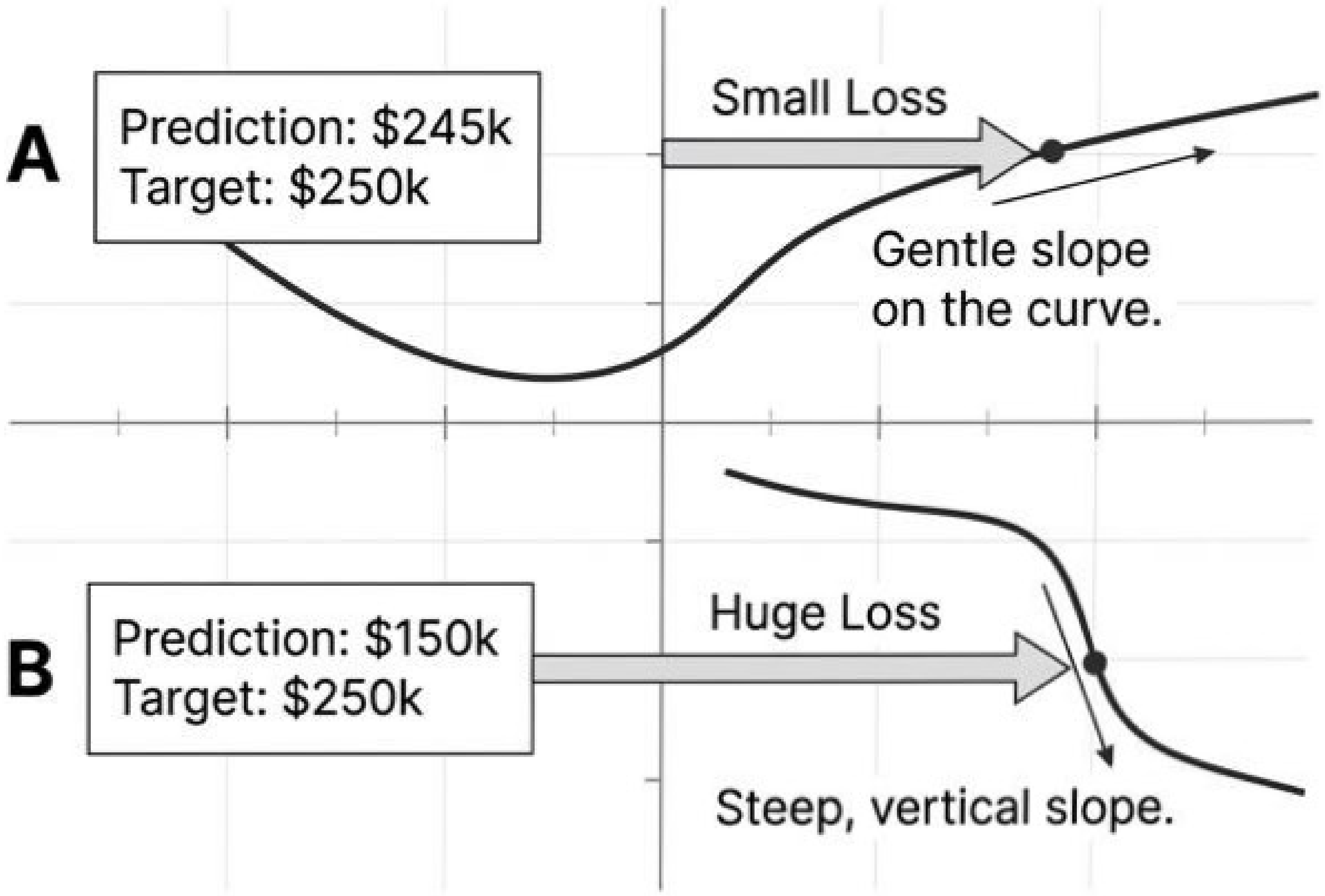
Prerequisites:
Tensor, Activations, Layers

Loss Transforms ‘Wrongness’ into a Differentiable Signal

Key Assertions

- The Core Concept: Quantifying the distance between prediction and reality.
- The Constraint: Must be **differentiable**.
- We need a curve, not a cliff. The optimizer needs to know *which direction* to move.

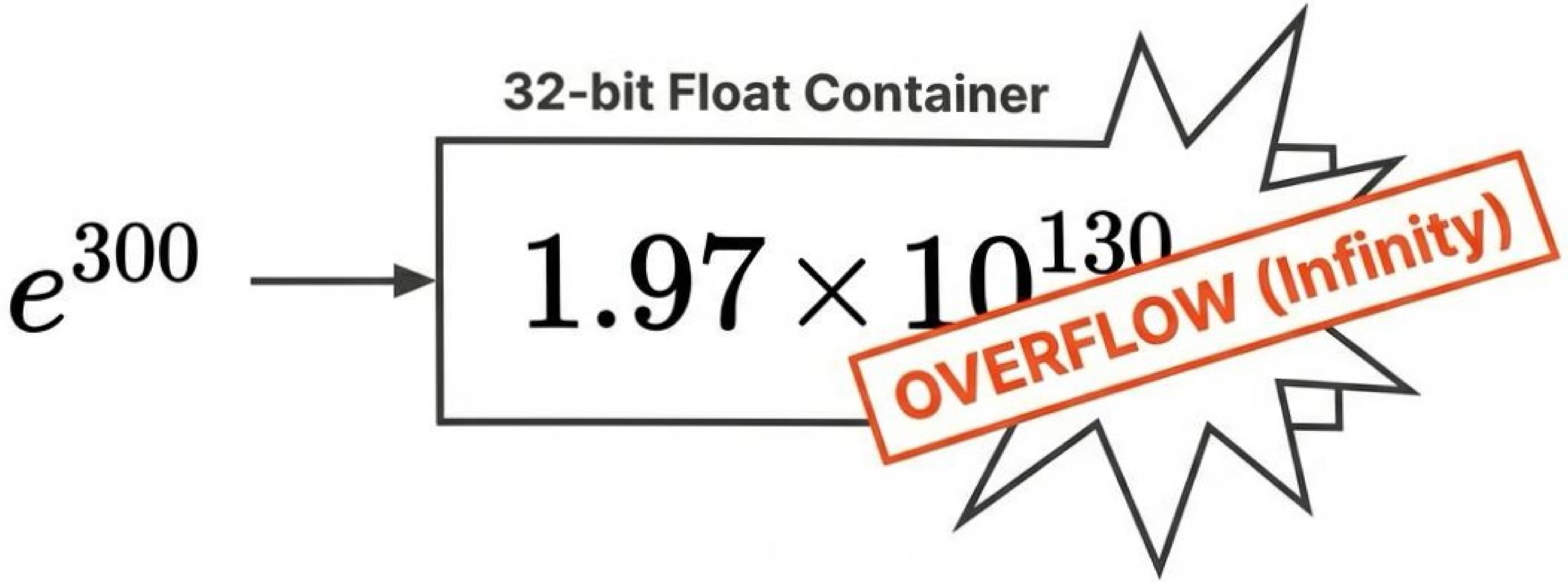
The Feedback Loop



The steepness of the curve becomes the gradient.

The Floating Point Trap

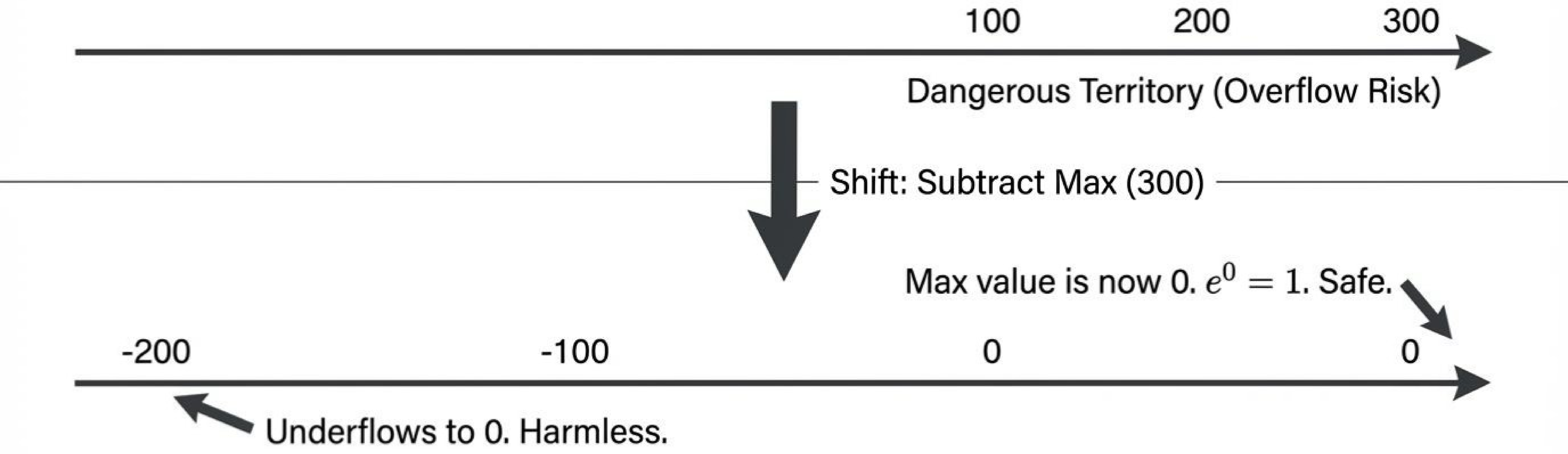
System Constraints vs. Mathematical Ideals



`1. logits = [100, 200, 300] -> 2. softmax = exp(300) / sum(…) -> inf → 3. loss = NaN →` **Training Crash**.

The Log-Sum-Exp Trick

Shifting the curve to safe ground.



$$\mathrm{softmax}(x) = \mathrm{softmax}(x - \max(x))$$

Implementation: log_softmax

"tinytorch/core/losses.py" in Public Sans in Obsidian Charcoal

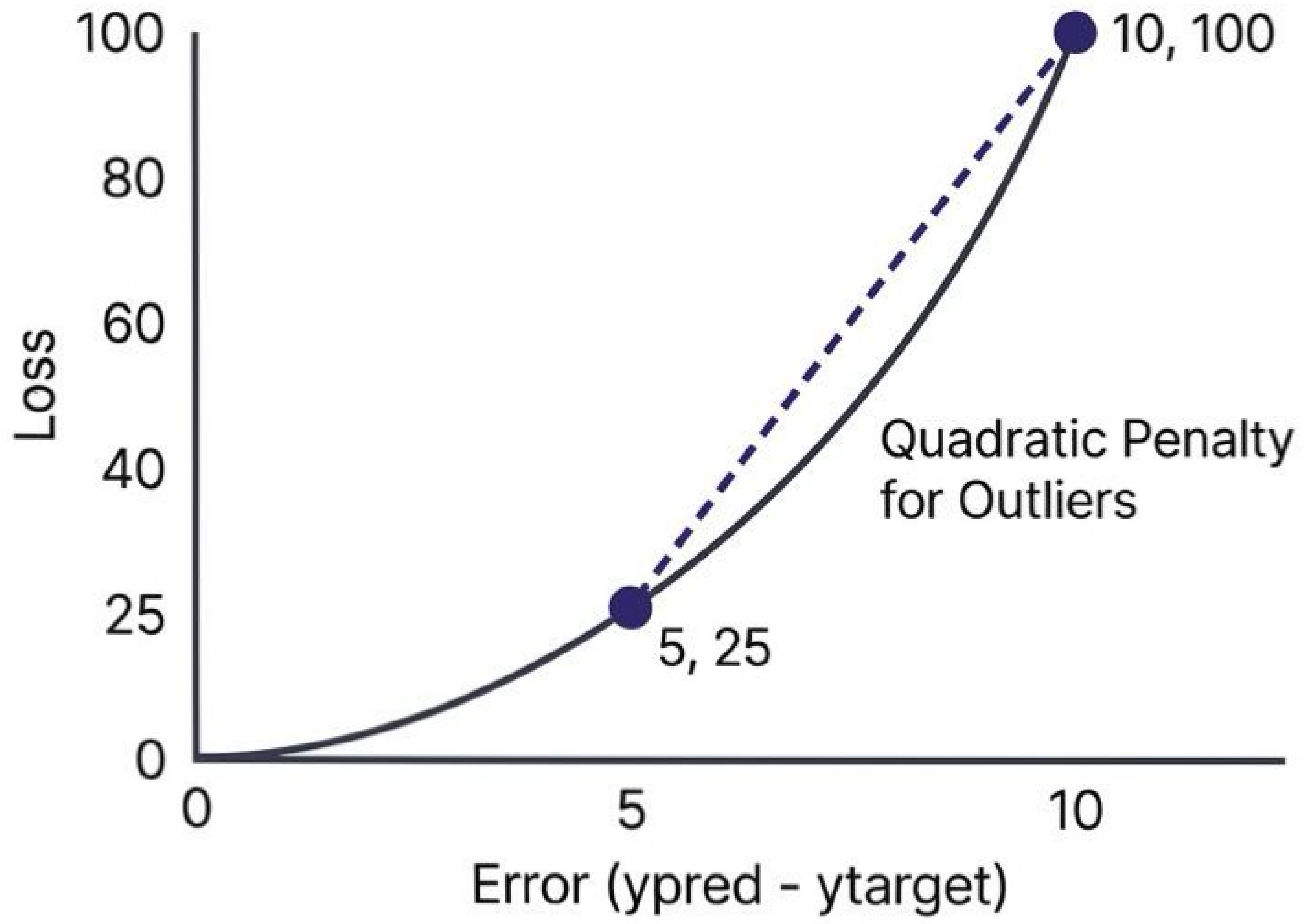
```python
def log_softmax(x: Tensor, dim: int = -1) -> Tensor:
    # 1. Find max for stability
    max_vals = np.max(x.data, axis=dim, keepdims=True)

    # 2. Subtract max (The Shift)
    shifted = x.data - max_vals

    # 3. Compute log-sum-exp safely
    log_sum_exp = np.log(np.sum(np.exp(shifted), axis=dim, keepdims=True))

    # 4. Result = input - max - log_sum_exp
    return Tensor(x.data - max_vals - log_sum_exp)
```

The Invariant:
No value > 0

Mathematically exact,
computationally safe.

Mean Squared Error (MSE)

For Regression: Distance, not surprise.



The Invariant:

$$\frac{1}{N}\sum(y_{pred} - y_{target})^2$$

Why Squared?

1. **Sign Independence:** -10 and +10 are treated equal.
2. **Severity:** An error of 10 is 4x worse than an error of 5.

Implementation: `MSELoss`

tinytorch/core/losses.py

```
def forward(self, predictions: Tensor, targets: Tensor) -> Tensor:
    # 1. Element-wise difference
    diff = predictions.data - targets.data

    # 2. Square the differences
    squared_diff = diff ** 2

    # 3. Mean reduction
    mse = np.mean(squared_diff)

    return Tensor(mse)
```

Systems Insight

- **Complexity:** $O(N)$
- **Memory:** Minimal.
- **Note:** No exponents, no logs. Linear scaling.

CrossEntropyLoss

For Classification: Measuring Surprise.

The Rules

Input
Logits (Unbounded)

→

Process
Maximize probability of correct class

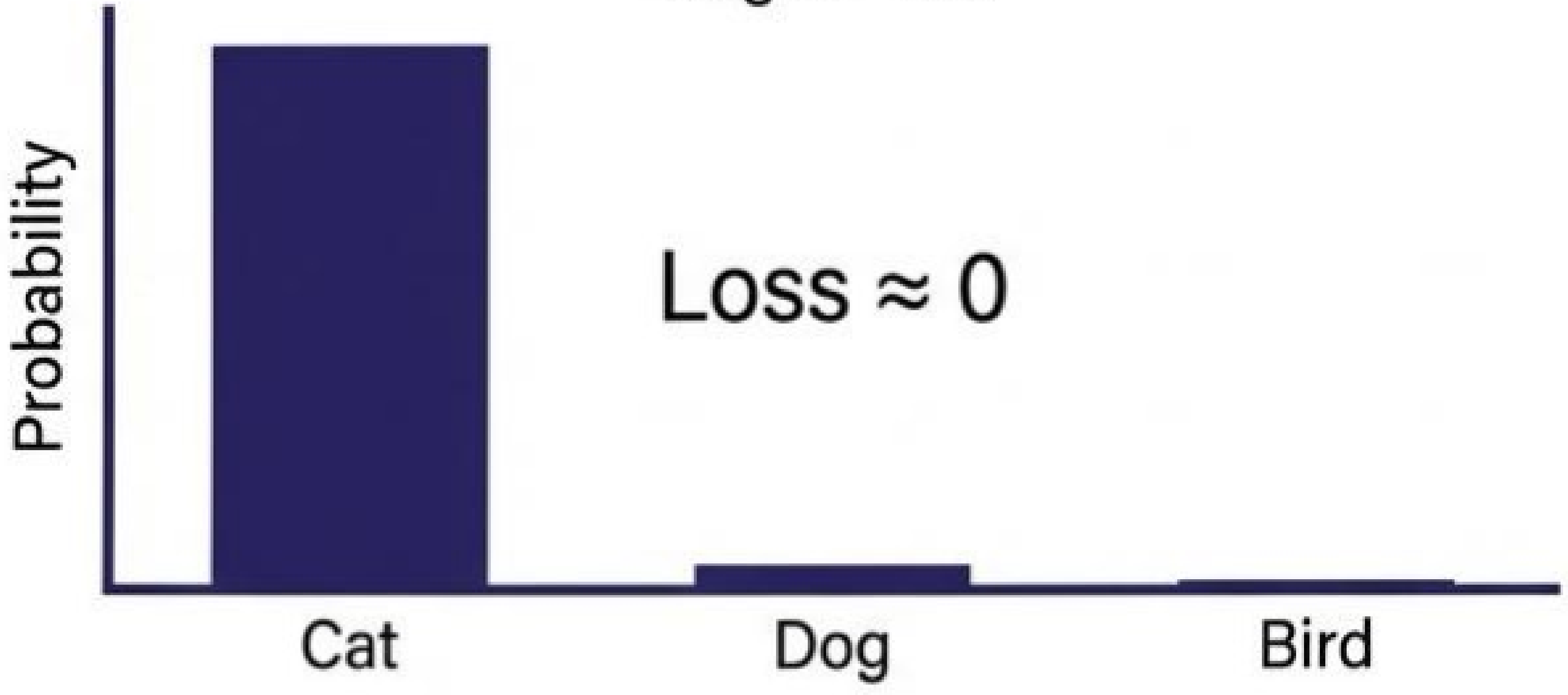
→

Target
Integer Class Indices

The Penalty Structure

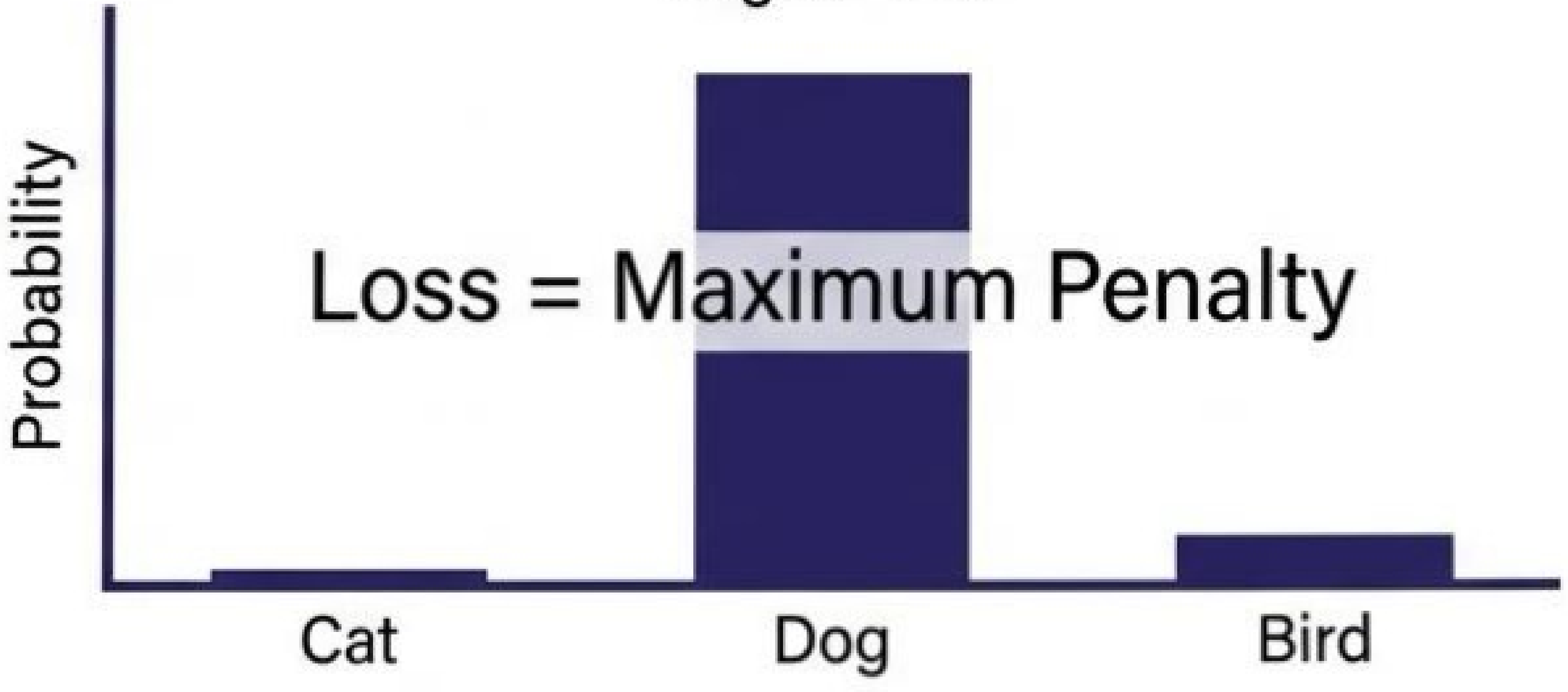
A. **Confidently Correct**

Target: "Cat"



B. **Confidently Wrong**

Target: "Cat"

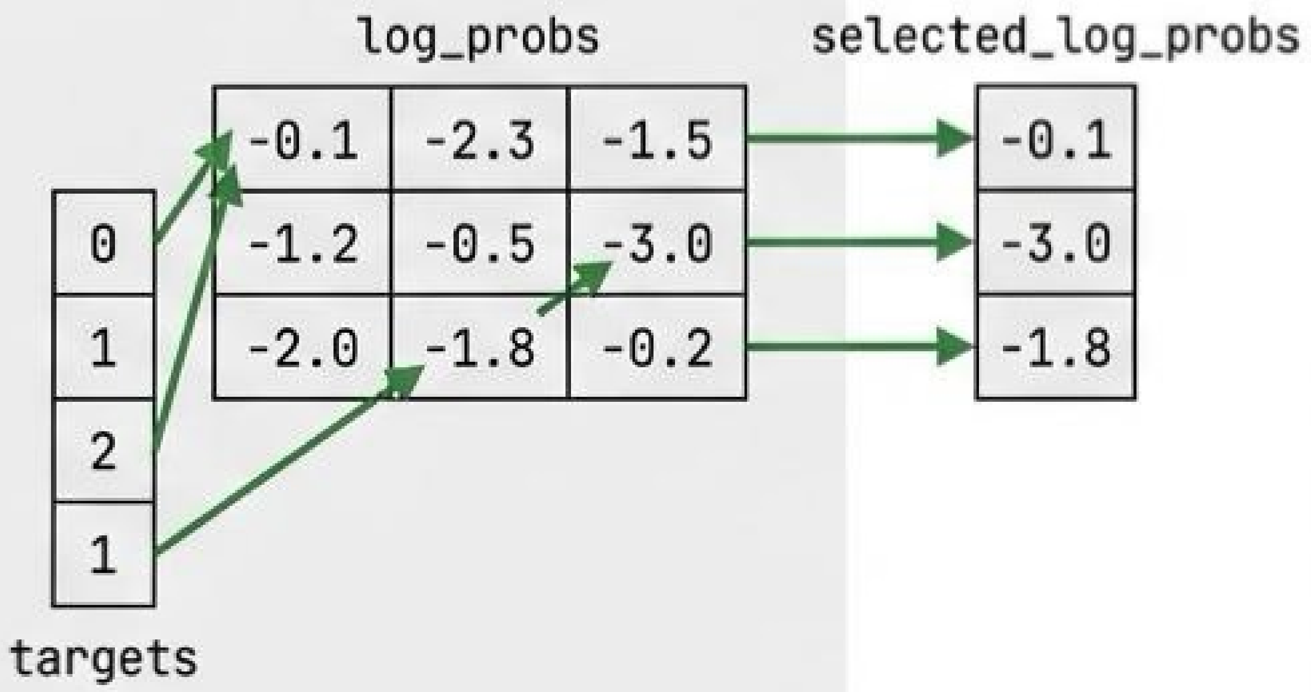


Implementation: `CrossEntropyLoss`

tinytorch/core/losses.py in Public Sans

```python
def forward(self, logits: Tensor, targets: Tensor) -> Tensor:
    # 1. Apply stable log_softmax
    log_probs = log_softmax(logits, dim=-1)

    # 2. Select probability of the correct class
    batch_size = logits.shape[0]
    target_indices = targets.data.astype(int)

    # Advanced Indexing: "Lookup Table" style
    selected_log_probs = log_probs.data[np.arange(batch_size), target_indices]

    # 3. Negative Log Likelihood
    return Tensor(-np.mean(selected_log_probs))
```

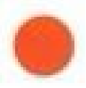
Advanced Indexing



System Constraint: The Memory Bottleneck

The $O(B \times C)$ Problem

Regression



1 Output Node

Large Language Model

50,000 Class Vocabulary

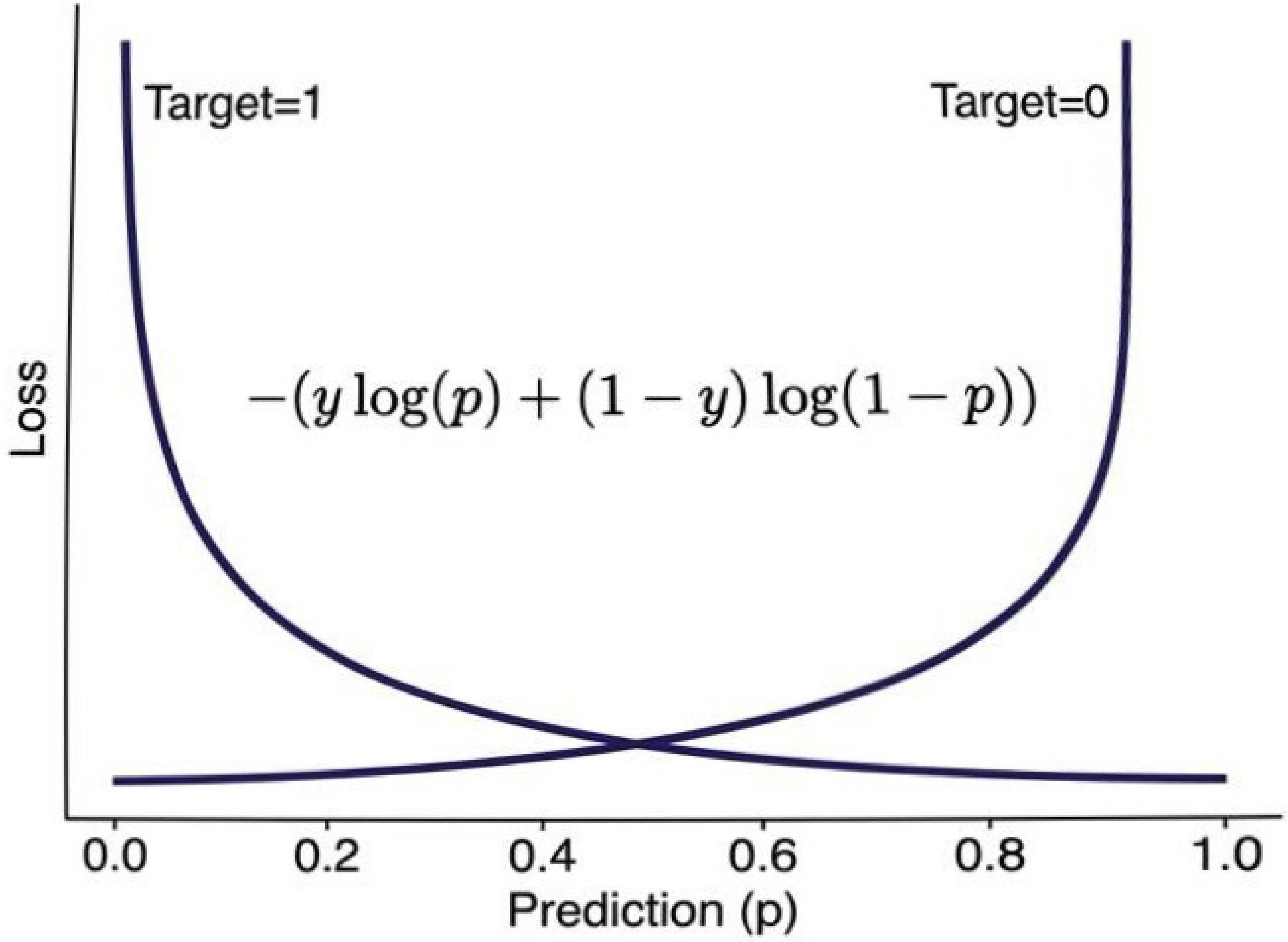
Scenario: `Batch 128, Vocab 50k, Float32`
Computation: `6.4 million exponentials per step.`
Memory: `76.8 MB per step (Loss calculation only).`

Impact: The Loss Function itself becomes the training bottleneck.

Binary Cross Entropy (BCE)

The Special Case: Yes vs. No

- Use Case: Spam, Fraud, Disease.
- Input: **Probabilities** (0.0 to 1.0).
- Logic: Symmetric Penalty. We penalize False Positives AND False Negatives.



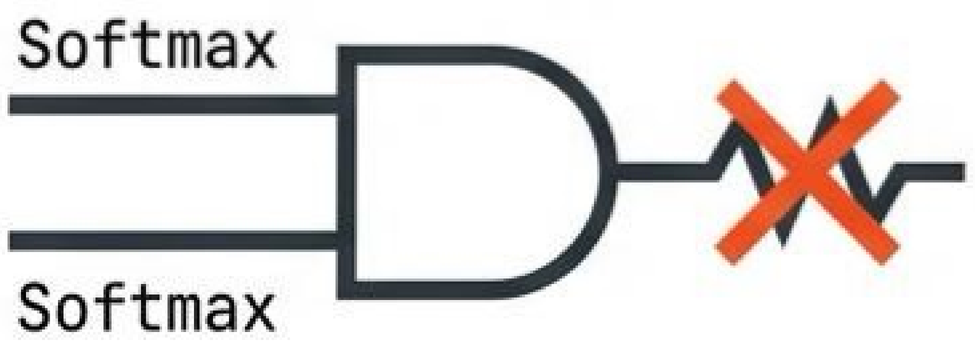
Implementation: `BinaryCrossEntropyLoss`

tinytorch/core/losses.py

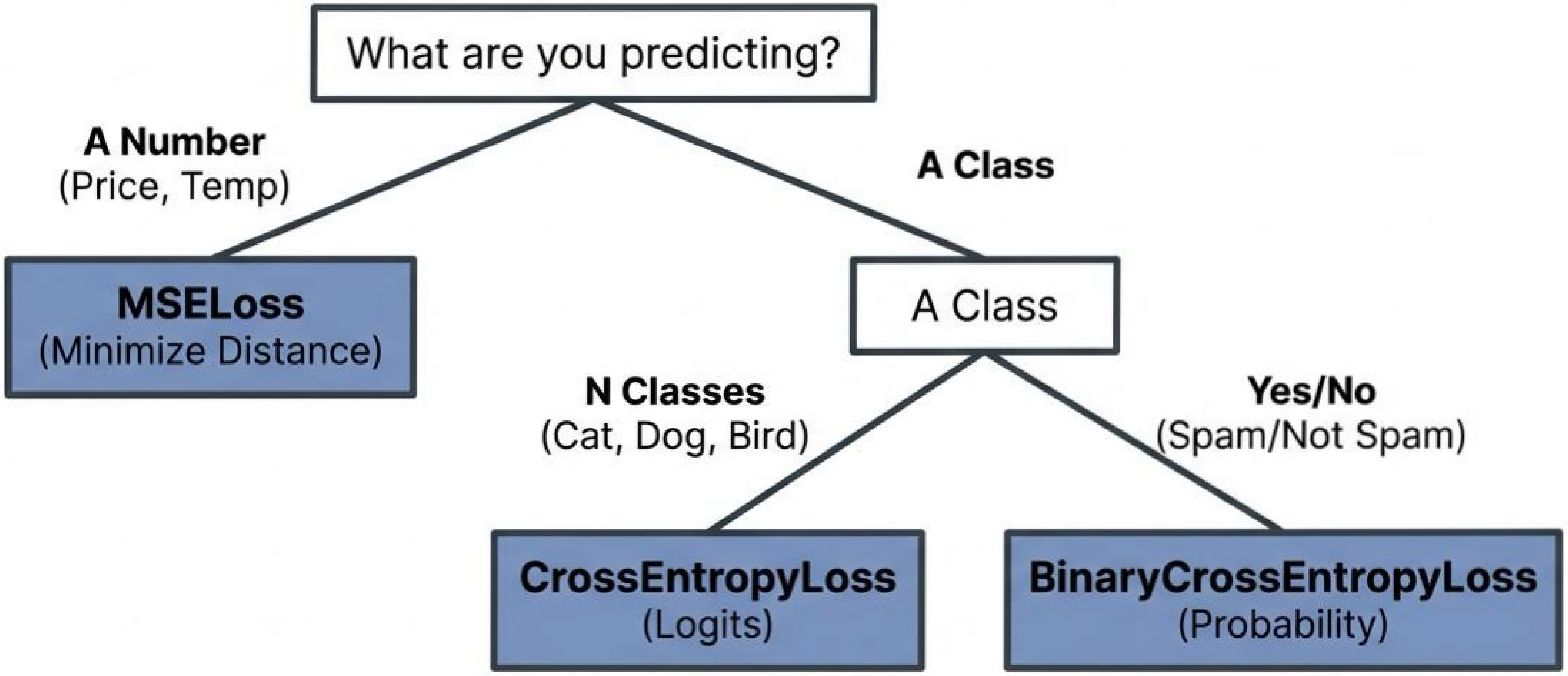
```
def forward(self, predictions: Tensor, targets: Tensor) -> Tensor:
    # 1. Clip to prevent log(0) -> NaN
    eps = 1e-7
    clamped_preds = np.clip(predictions.data, eps, 1 - eps)

    # 2. Compute BCE
    term_1 = targets.data * np.log(clamped_preds)
    term_2 = (1 - targets.data) * np.log(1 - clamped_preds)

    return Tensor(-np.mean(term_1 + term_2))
```

The Safety Rail. Prevents $-\infty$ when model is 'perfectly wrong'.

Systems Insight: Common Failure Modes

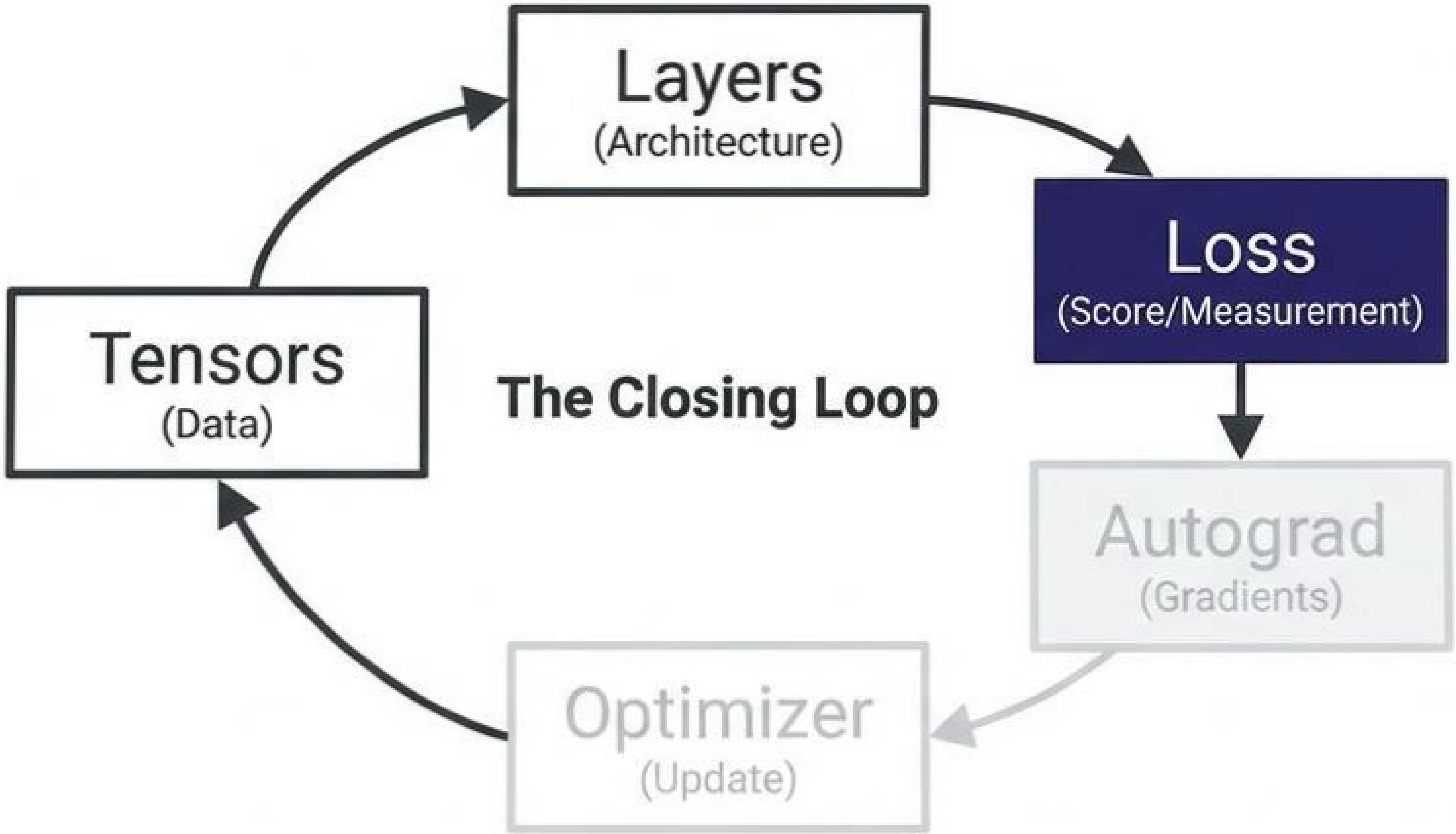
The Input Mismatch	The Shape Mismatch	The NaN Trap
		
Double Softmax	**Index Error**	**Log(0)**
Applying Softmax in the model AND the loss function.	Target index 5 when `Classes=3`.	Forgetting `np.clip` or `log_sum_exp`.
Result: Vanishing Gradients	**Result:** Crash.	**Result:** Training becomes unstable

Choosing the Right Feedback Signal



Inference: Loss is strictly for training. In production, use `argmax` or raw probability.

From Measurement to Improvement



We know *how* wrong the model is ($L = 5.2$).

Next, we calculate *how to fix it*.

Next Up: Module 05: DataLoader & Module 06: Autograd