HSTU

The Learning Process: A Step-by-Step Walkthrough

The Scenario:

- Our HSRNN is in the middle of training on the Parity Task.
- It sees an input sequence where the correct final answer is "Odd" (D=1), but its current weights cause it to predict "Even" (D=0).
- The loss is high. We call loss.backward().

Step 1: The Error Signal Arrives at the End
The loss function calculates a large error. This error signal begins to travel backward through the network. It flows back through the final fc_out layer and the torch.cat operation, arriving at the final V_t and D_t states.

Step 2: The Signal Reaches the "Flip" Logic
The error flows backward from D_t to the line that calculated it:
D_t = D_prev * (1 - spike) + (1 - D_prev) * spike

The autograd engine calculates how a small change in the spike variable would have affected the final D_t. Since the prediction was wrong, it determines that the spike **should have been different.** For example, if spike was 0 but should have been 1 to get the right answer, a strong error signal is sent to the spike variable.

Step 3: The "Magic" of the Surrogate Gradient
Now the error signal arrives at the spike = spike_fn(V_t - theta) line. This is the critical moment.

- **The "True" Gradient's Response:** If we were using the true, mathematical gradient, it would be 0. The error signal would hit a wall. The journey would end.
  The V_t variable and all the weights that created it would receive a gradient of 0. **No learning would occur.**
- **The Surrogate Gradient's Response (How it *Actually* Works):** The spike_fn's custom backward function takes over. It looks at two things:
  1. The **strong error signal** coming from the D_t calculation (the "what to do" signal).
  2. The value of V_t - theta that it saved during the forward pass (the "where we are" signal).

It then makes an intelligent decision based on our "useful lie":

    - **If V_t was very close to theta:** The surrogate's bell-curve shape is at its peak. It takes the strong incoming error signal and multiplies it by a large number (e.g., 0.9). It passes a **strong gradient** back to V_t.
    - **If V_t was far from theta:** The surrogate's bell-curve is near its tails. It takes the strong incoming error signal and multiplies it by a very small number (e.g., 0.01). It passes a **weak gradient** back to V_t.

Step 4: The Weights are Updated
The gradient has now successfully crossed the "non-differentiable bridge." It flows from V_t back to the linear_in layer and its weights (W_in). The optimizer receives this gradient.

- The gradient is essentially an instruction that says: **"Change the weights W_in in a way that would have pushed V_t just a little bit higher, so that it would have crossed the threshold and produced the correct spike."**

The optimizer follows this instruction and nudges the weights of the linear_in layer.

Step 5: The Next Forward Pass
On the next training example, the weights are slightly better. The V_t potential is now slightly more likely to be on the correct side of the threshold. The model makes a slightly less wrong prediction.

This process repeats millions of times, and through these tiny, guided nudges—all directed by the "useful lie" of the surrogate gradient—the network's weights slowly converge to the perfect configuration that solves the problem.