Online Softmax: Mathematical Derivation

1 Problem Definition

Given an input vector $\mathbf{x} = [x_1, ..., x_n]$, we want to compute the softmax function:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1}$$

2 Naive Approach with Max Trick

To prevent numerical overflow, the standard approach subtracts the maximum:

$$\text{softmax}(x_i) = \frac{e^{x_i - \max_j(x_j)}}{\sum_{j=1}^{n} e^{x_j - \max_j(x_j)}} \tag{2}$$

3 Online Algorithm

3.1 State Variables

At step k, we maintain:

1. Current maximum up to k elements:

$$m_k = \max(m_{k-1}, x_k) \tag{3}$$

2. Normalizer for k elements:

$$l_k = \sum_{i=1}^{k} e^{x_i - m_k} \tag{4}$$

3.2 Update Rules

The key innovation is the normalizer update formula:

$$l_k = l_{k-1} \cdot e^{m_{k-1} - m_k} + e^{x_k - m_k} \tag{5}$$

3.3 Proof of Correctness

Consider two cases:
Case 1: No New Maximum ($m_k = m_{k-1}$)

$$\begin{aligned} l_k &= l_{k-1} \cdot e^{m_{k-1}-m_k} + e^{x_k-m_k} \\ &= l_{k-1} \cdot 1 + e^{x_k-m_k} \\ &= \sum_{i=1}^{k-1} e^{x_i-m_k} + e^{x_k-m_k} \\ &= \sum_{i=1}^{k} e^{x_i-m_k} \end{aligned} \tag{6}$$

Case 2: New Maximum ($m_k > m_{k-1}$)

$$\begin{aligned} l_k &= l_{k-1} \cdot e^{m_{k-1}-m_k} + e^{x_k-m_k} \\ &= (\sum_{i=1}^{k-1} e^{x_i-m_{k-1}}) \cdot e^{m_{k-1}-m_k} + e^{x_k-m_k} \\ &= \sum_{i=1}^{k-1} e^{x_i-m_k} + e^{x_k-m_k} \\ &= \sum_{i=1}^{k} e^{x_i-m_k} \end{aligned} \tag{7}$$

3.4 Final Computation

After processing all n elements, for any index i:

$$\text{softmax}(x_i) = \frac{e^{x_i-m_n}}{l_n} \tag{8}$$

4 Complexity Analysis

4.1 Space Complexity

- Naive: $\mathcal{O}(n)$ additional space for temporary arrays
- Online: $\mathcal{O}(1)$ additional space for m_k and l_k

4.2 Time Complexity

- Naive: Multiple passes (find max, compute exp, sum)
- Online: Single pass, constant work per element

5 Advantages

1. Memory efficient (constant extra space)
2. Single pass through data
3. Numerically stable (always subtracts current maximum)
4. Cache friendly (sequential access)