Softmax Calculation: Manual Step-by-Step

Problem

Given scores = [10, 0, -5], apply the softmax function manually.

Softmax Formula

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Step-by-Step Calculation

Step 1: Calculate exponentials for each score

- $e^{10} = 22026.4658$
- $e^{0} = 1.0000$
- $e^{-5} = 0.0067$

Step 2: Sum all exponentials

$$\text{Sum} = 22026.4658 + 1.0000 + 0.0067 = 22027.4725$$

Step 3: Normalize (divide each by the sum)

$$\text{softmax}(10) = \frac{22026.4658}{22027.4725} = 0.9999545 \approx 99.995\%$$

$$\text{softmax}(0) = \frac{1.0000}{22027.4725} = 0.0000454 \approx 0.005\%$$

$$\text{softmax}(-5) = \frac{0.0067}{22027.4725} = 0.0000003 \approx 0.0003\%$$

Result

$$\text{softmax}([10, 0, -5]) = [0.9999545, 0.0000454, 0.0000003]$$

Verification: $0.9999545 + 0.0000454 + 0.0000003 = 1.0000002 \approx 1$ ✓

Question 1: Which value dominates after softmax, and why?

Answer: The first value (score = 10) **completely dominates** with ~99.995% of the probability.

Why?

- The exponential function e^x amplifies differences dramatically
- $e^{10} = 22,026$ is **22,026 times** larger than $e^0 = 1$
- e^{10} is **3.3 million times** larger than $e^{-5} = 0.0067$
- When one exponential is much larger than others, it gets nearly all the probability mass
- Softmax essentially picks the maximum value when differences are large

Question 2: What does this tell you about why we need the scale factor $\frac{1}{\sqrt{d_k}}$ in attention?

Answer: Without scaling, attention scores become **too extreme**, leading to several problems:

Problem: Unscaled Dot Products Grow with Dimension

When computing $Q \cdot K^T$ for high-dimensional vectors:

- Dot products naturally grow in magnitude as dimension d_k increases
- For random unit vectors, dot product variance $\approx d_k$
- Example: If $d_k = 64$, dot products might be in range [-8, 8] or larger

What Happens Without Scaling?

If we get large scores like [10, 0, -5]:

- Softmax outputs: [0.9999, 0.0000, 0.0000]
- **Almost all attention** goes to one position
- Other positions get essentially **zero attention**
- Model can't learn to attend to multiple relevant positions

Gradient Problem (Vanishing Gradients)

When softmax is too "peaked":

- Gradient $\frac{\partial \text{softmax}}{\partial x_i} \approx 0$ for non-dominant values
- Training becomes very slow or stuck
- Model can't adjust attention weights effectively

Solution: Scale by $\frac{1}{\sqrt{d_k}}$

Example with scaling:

Original scores: [10, 0, -5]

If $d_k = 4$, scale factor $= \sqrt{4} = 2$

Scaled scores: $\frac{[10, 0, -5]}{2} = [5, 0, -2.5]$

Calculate softmax:

- $e^5 = 148.41$
- $e^0 = 1.00$
- $e^{-2.5} = 0.082$

Sum = 149.492

Softmax output: [0.9929, 0.0067, 0.0005]

Still dominated by first value, but more reasonable!

With even larger d_k (like 64 used in transformers):

- Scale factor $= \sqrt{64} = 8$
- Scaled scores: [1.25, 0, -0.625]
- Softmax: [0.6011, 0.1711, 0.2278]
- **Much more balanced distribution!**

Key Benefits of Scaling:

1. **Keeps scores in stable range** (~[-3, 3] typically)
2. **Prevents softmax saturation** (too peaked distributions)
3. **Maintains healthy gradients** for learning
4. **Allows multi-headed attention** to attend to multiple positions
5. **Makes training stable** regardless of dimension size

Mathematical Intuition

For random vectors q, k with unit variance:

$$\text{Var}(q \cdot k) \approx d_k$$

Dividing by $\sqrt{d_k}$ normalizes variance:

$$\text{Var}\left(\frac{q \cdot k}{\sqrt{d_k}}\right) \approx 1$$

This keeps softmax input statistics consistent across different model dimensions.