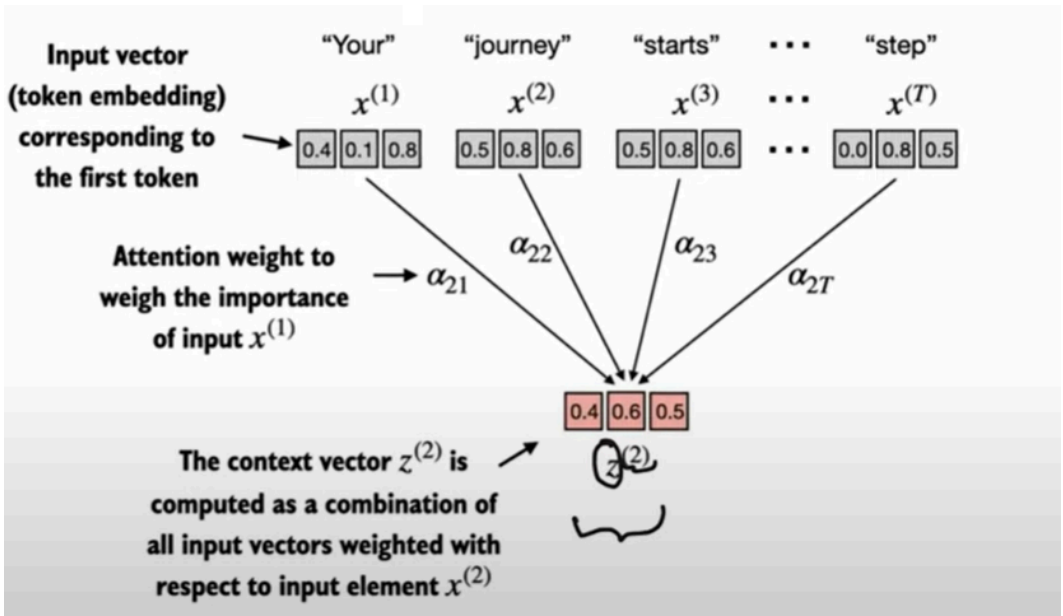
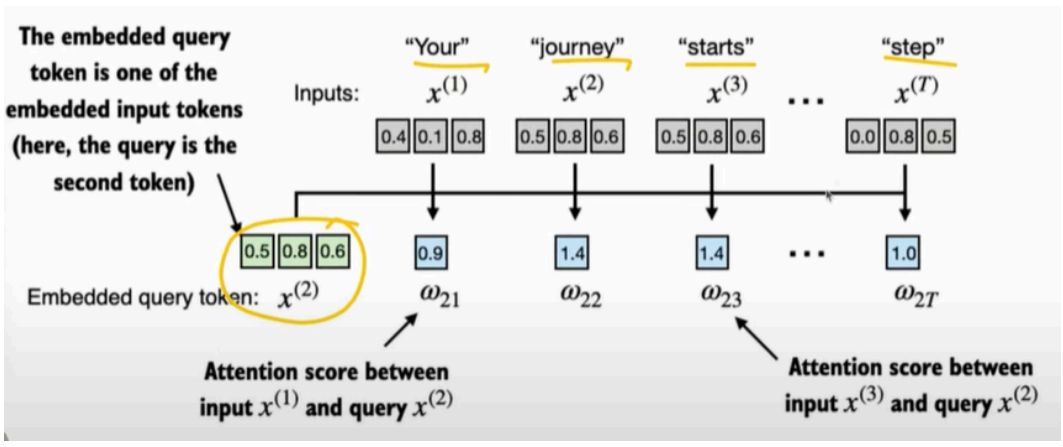
Lecture 14: Simplified Self Attention

Context Vector: Not only contains semantic meaning but also contains information about how one word related to other.





The first step is to find out the **attention score**, which helps us decide how much importance should be given to each word in the input.

The intermediate attention scores are calculated between the query token and each input token. These are calculated by doing the dot product between the query token and every other input token.

Why dot product? Because dot product quantifies how much two vectors are aligned. Higher the dot product, high the similarity and attention scores between two elements.

Why Softmax Should Be Used Instead of Raw Dot-Products for Attention Weights

Raw dot-product scores:

$s_i = x_i \cdot q$

are unbounded real numbers. They may be negative, large, small, or clustered. Attention mechanisms, however, require **weights** that behave like probabilities—values between 0 and 1 that sum to 1. Softmax provides this transformation.

Reasons for Using Softmax

1️⃣ Converts arbitrary scores into a probability distribution

Softmax transforms scores into:

$$\alpha_i = \frac{e^{s_i}}{\sum_{j=1}^{n} e^{s_j}}$$

This guarantees:

- 0<αi<1
- ∑iαi=1

So each αi becomes a proper attention weight.

2️⃣ Emphasizes important differences

Exponentiation magnifies relatively larger scores and suppresses smaller ones.

This sharpening effect allows the attention mechanism to highlight the most relevant tokens more clearly.

3️⃣ Smooth, differentiable, and gradient-friendly

Softmax is continuous and differentiable everywhere, making gradient-based optimization stable and effective.

Softmax Formula Illustrated with 6 Tokens

Given six raw attention scores:

[s1,s2,s3,s4,s5,s6]

their softmax-weighted attention values become:

$$\alpha_1 = \frac{e^{s_1}}{e^{s_1} + e^{s_2} + e^{s_3} + e^{s_4} + e^{s_5} + e^{s_6}}$$

...and similarly for α2,α3,...,α6

The denominator remains the same for all six—just a sum of exponentials.

The Softmax Formula Used by PyTorch

PyTorch does **not** compute:

$$e^{s_i}$$

directly.

Instead, it computes:

$$\alpha_i = \frac{e^{\,s_i - \max(s)}}{\sum_j e^{\,s_j - \max(s)}}$$

This is mathematically equivalent to the standard formula—subtracting a constant from every score does not change the final probabilities.

✨ Why PyTorch Uses the “Subtract max” Trick

The reason is **numerical stability**.

Exponentials can overflow extremely quickly.

If one of the scores is large (e.g., 50), then:

$$e^{50} \approx 3 \times 10^{21}$$

which can overflow floating-point limits, produce infinities, and destabilize the entire computation.

Subtracting the maximum score shifts all values:

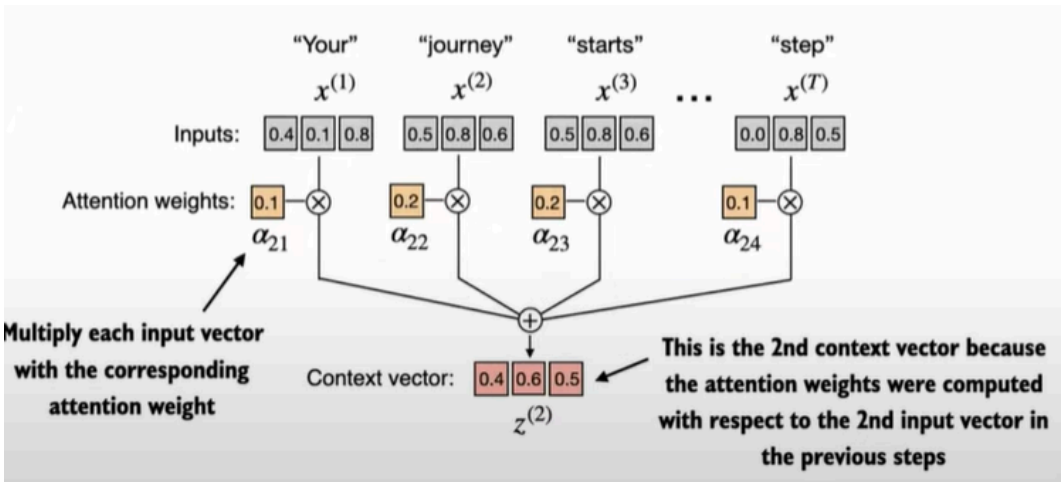
$$s - \max(s) = [s_1 - m,\ s_2 - m,\ ...,\ s_6 - m]$$

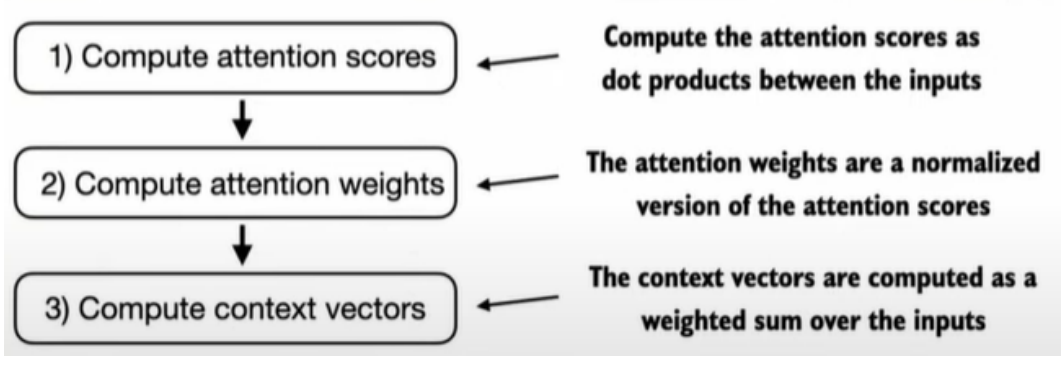
Now the largest value becomes 0, and:

e^0 = 1

Every other term becomes an exponential of a negative number, which stays safely in range.

After computing the normalized attention weights, we calculate the context vector z by multiplying the embedded input tokens xi, with the corresponding attention weights and then summing the resultant vectors.





Why attention needs trainable weights

Take the sentence:

"the cat sat on the mat because it was warm."

The key issue:

Different *"it"*s in different sentences refer to different things. A fixed attention rule can't know which token matters.

Without trainable weights, attention would basically act like a polite but clueless intern—treating all words as equally relevant or following some fixed pattern. That fails immediately:

- Sometimes **"it" → "cat"**
- Sometimes **"it" → "mat"**
- Sometimes **"it" → weather**
- Sometimes **"it" → something not even in the sentence**

A non-trainable attention mechanism has *no ability* to adapt to these differences.

What trainable weights actually fix

Trainable matrices (Q, K, V) learn:

- whether “it” tends to look back toward nouns
- how verbs modify relevance
- how position affects reference
- how semantics (warm, sat, mat) shape connections

So in this sentence:

- Q for **“it”** learns to seek noun-like keys
- K for **“cat”** and **“mat”** encode their semantic roles
- The dot product produces a high score for the correct antecedent

In many contexts:

- **“it” attends more to “cat”** (animate)
- In this sentence:

  “it” attends more to “mat” (because “warm” matches an object, not an animal)

Only trainable weights let those patterns emerge.