Seq2Seq

Anyway, to illustrate how to solve a **sequence-to-sequence** problem using a **seq2seq Encoder-Decoder** model, let's create one that translates English sentences into Spanish.

English → Spanish

For example, someone might say…

Likewise, not all Spanish sentences are the same length, so we need something that can generate different length sentences as output.

Lastly, the Spanish translation of an English sentence can have a different length than the original.

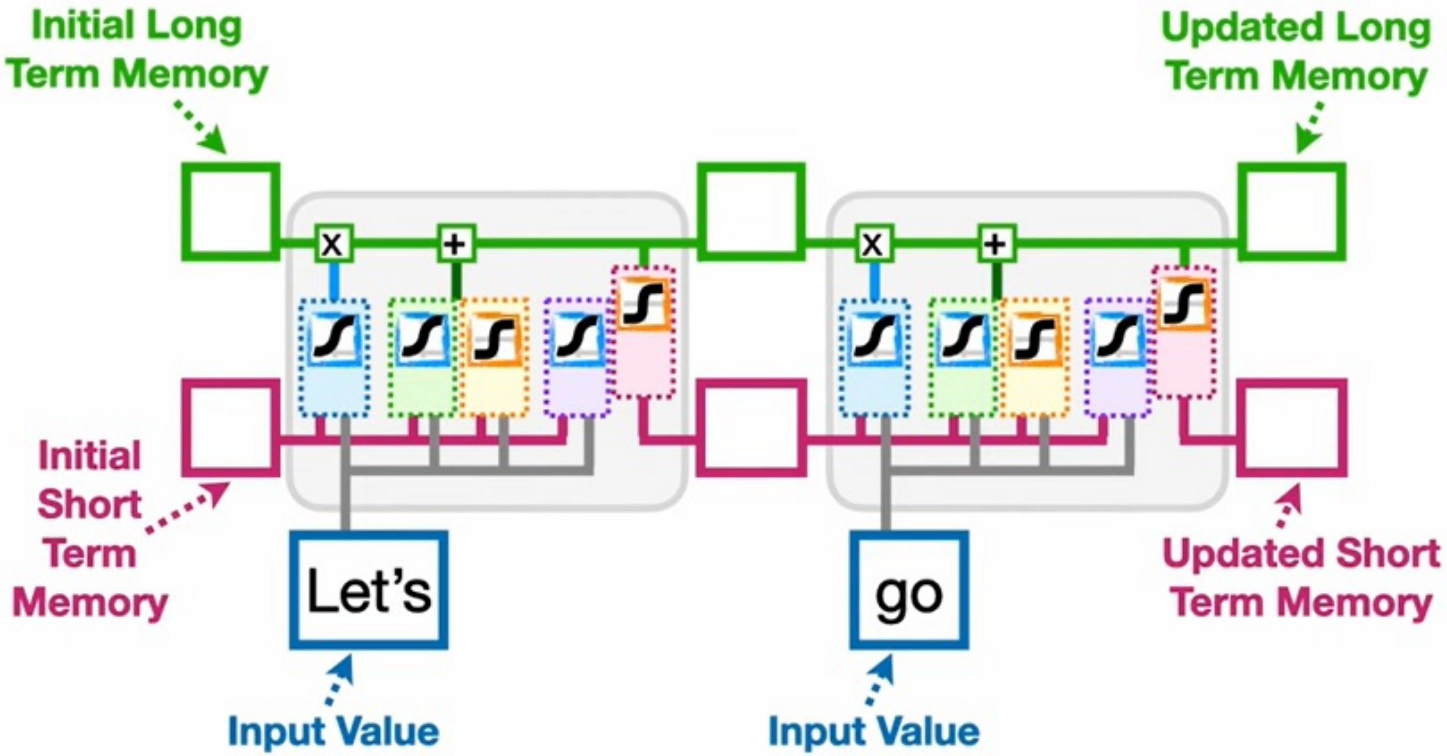
For example, the two word English sentence, **Let's go**…

…translates to the one word Spanish sentence, **Vamos**.

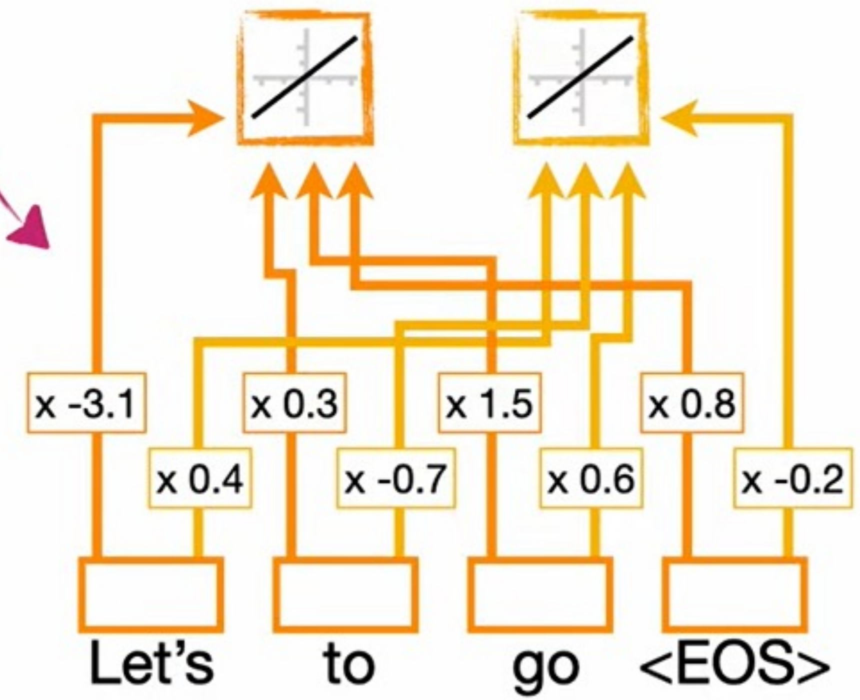
English Sentences	**Spanish Sentences**
Let's go.	Vamos.

So we need our **seq2seq Encoder-Decoder** model to be able to handle variable input and variable output lengths.

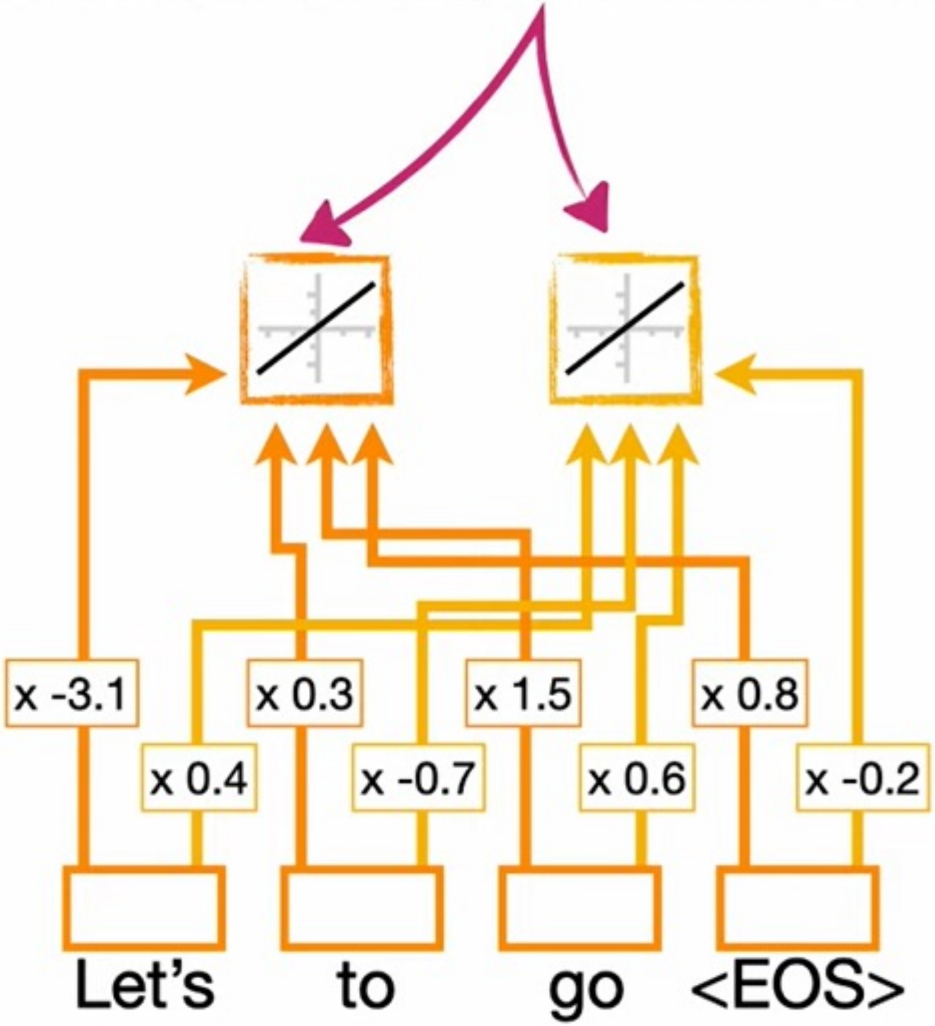
English Sentences		**Spanish Sentences**
Let's go.	→	Vamos.



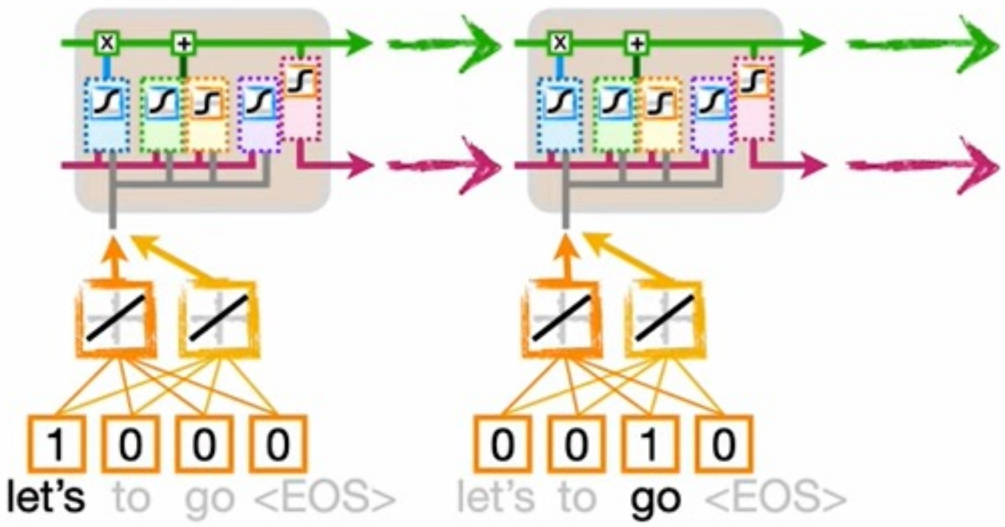
Instead, we use an **Embedding Layer** to convert the words into numbers.



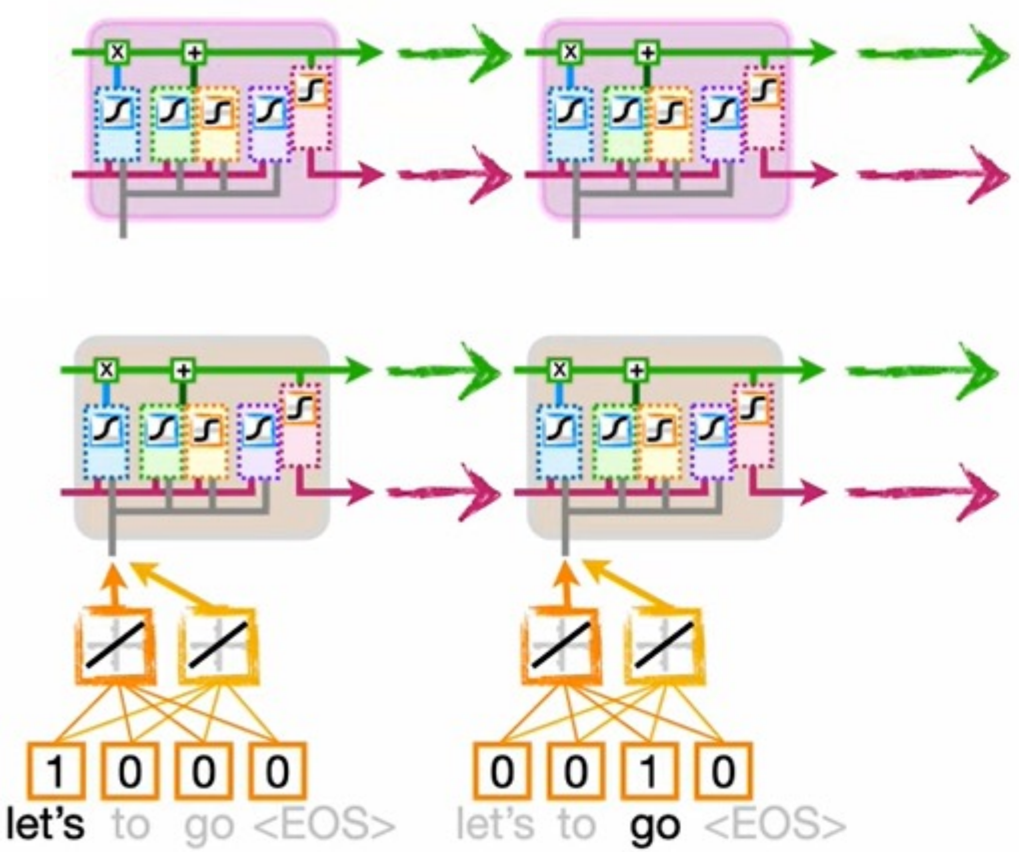
ALSO NOTE: In this example, we're just creating **2** embedding values per **Token** instead of hundreds or thousands.

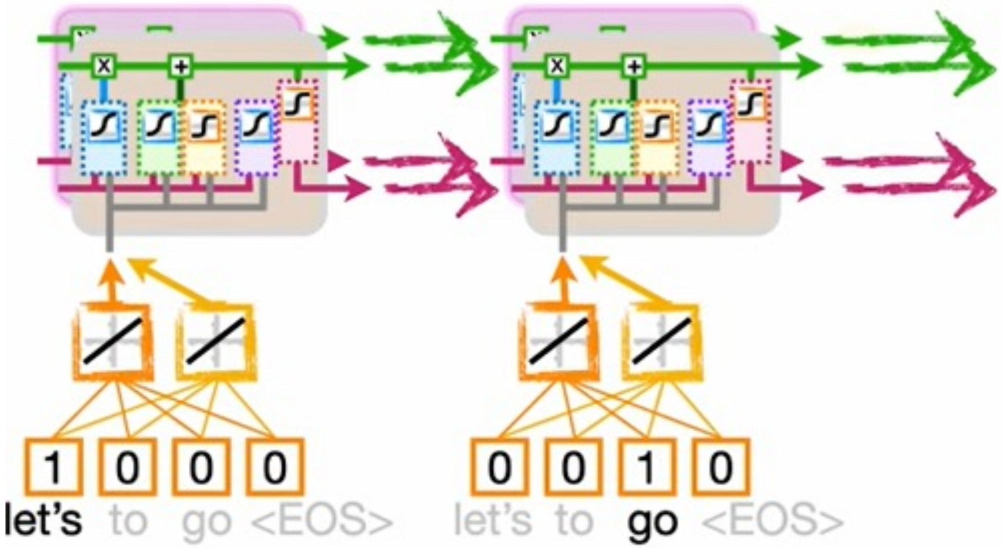


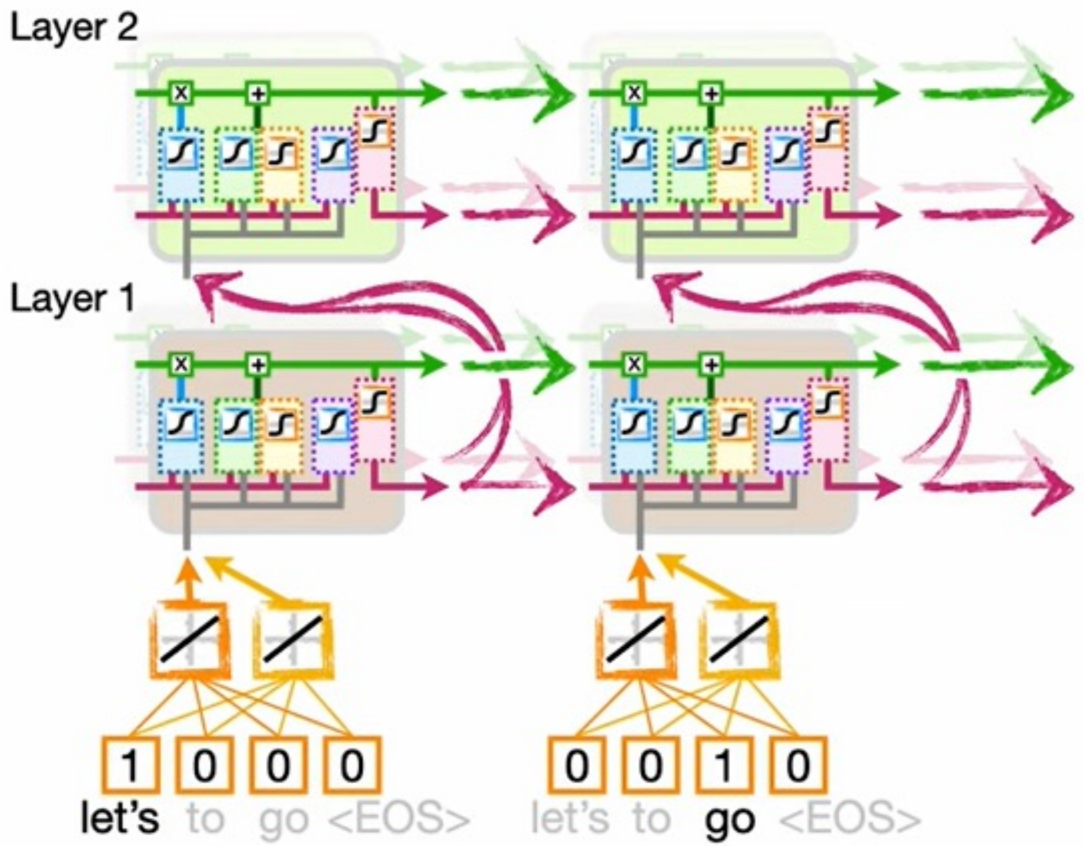
NOTE: To be clear, when we unroll the **LSTM** and the **Embedding Layer**, we reuse the exact same **Weights** and **Biases** no matter how many times we unroll them.



To keep things simple, we'll just add one additional **LSTM** cell to this stage.



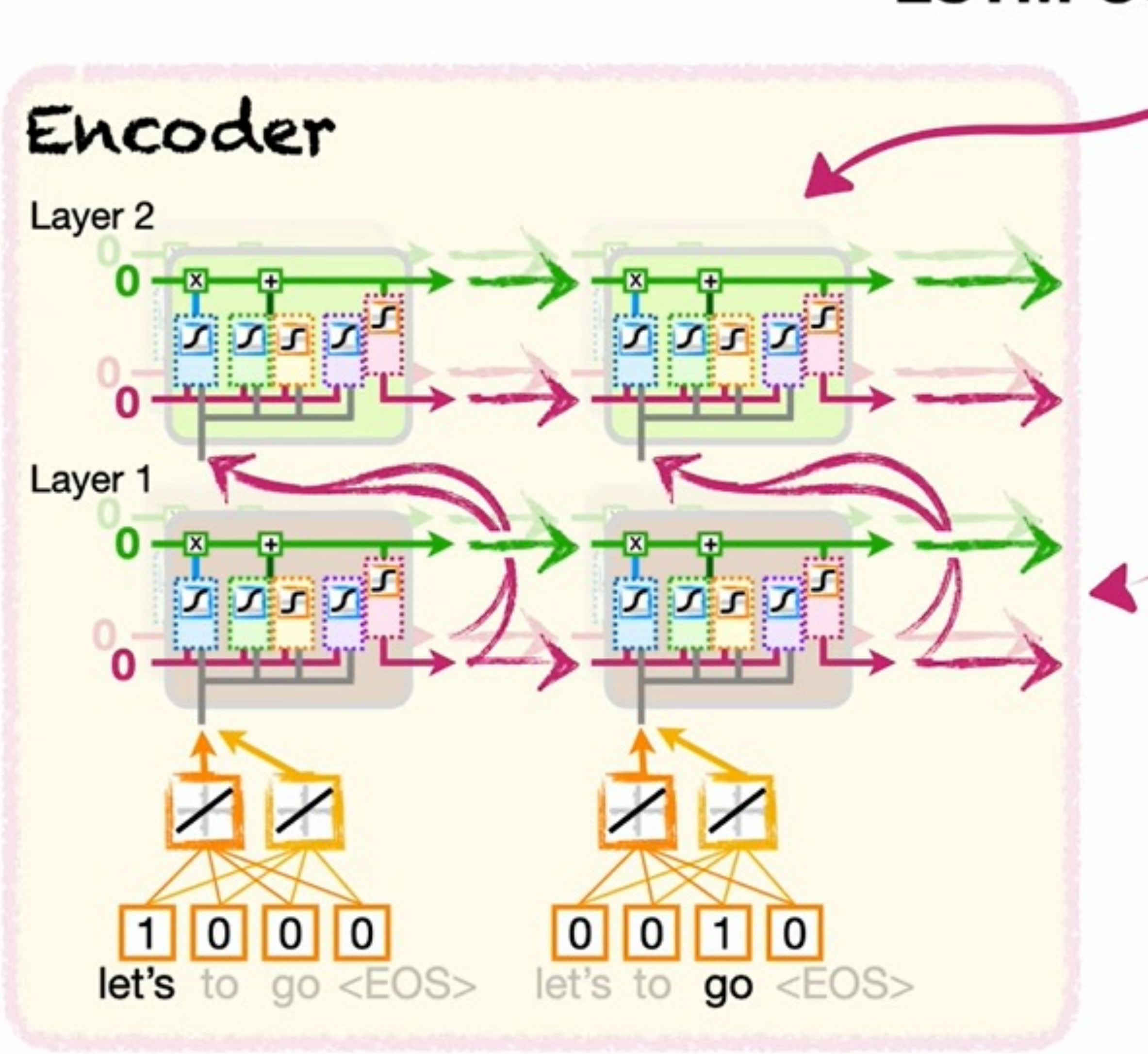


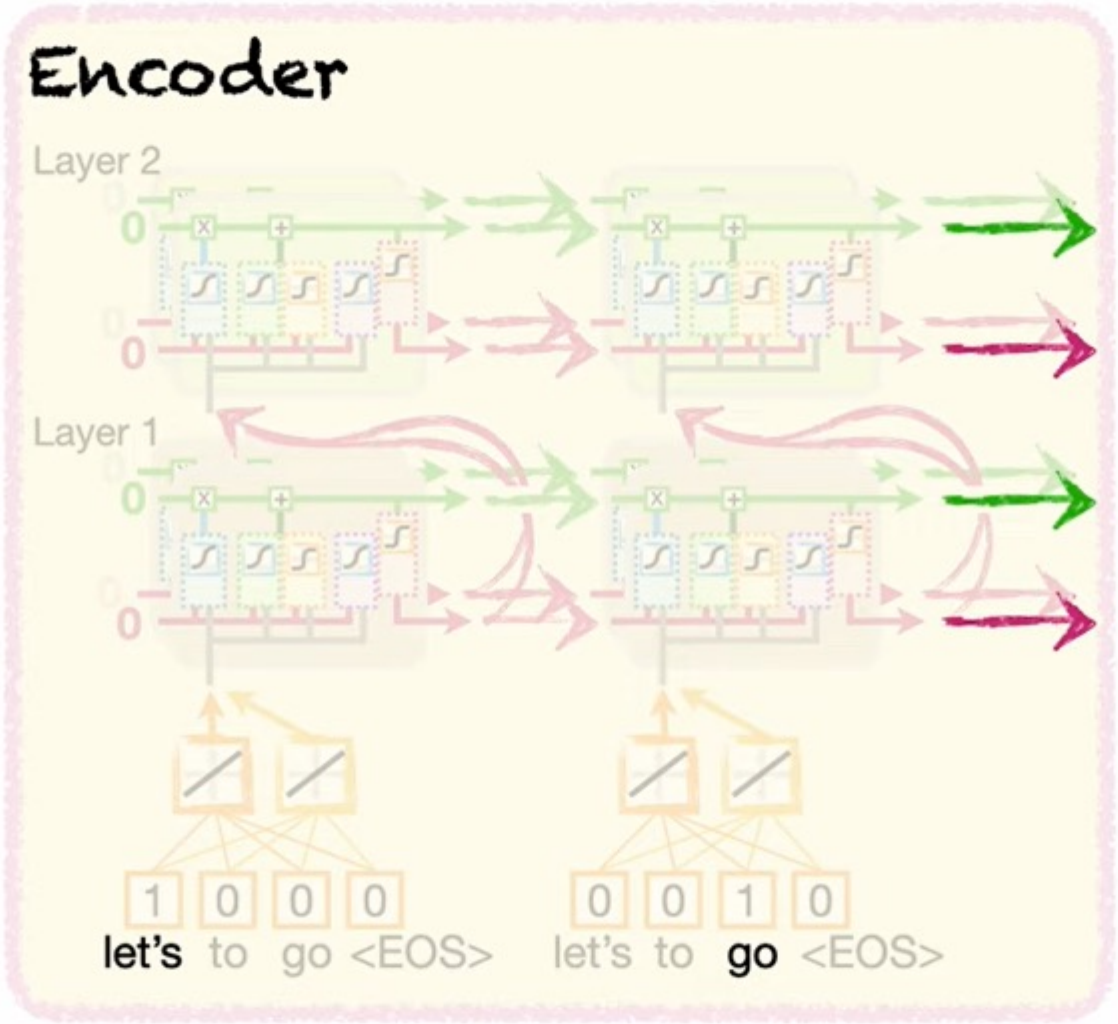


NOTE: Just like how both embedding values are used as inputs to both **LSTM** cells in the first layer…

…both outputs (the short-term memories, or hidden states) from the each cell in the first layer are used as inputs to both **LSTM** cells in the second layer.

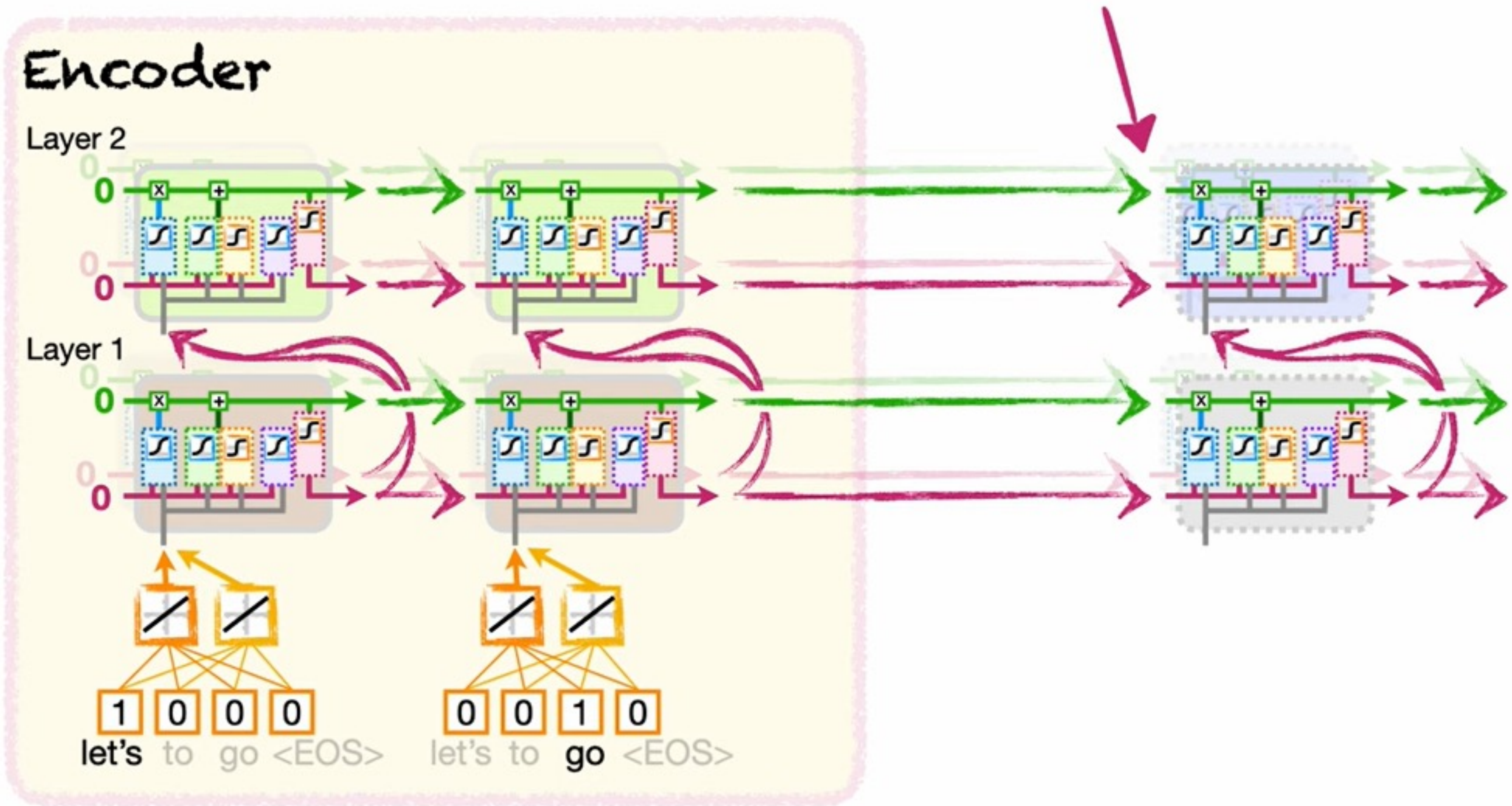
In this example, we have **2 Layers** of **LSTMs**, with **2 LSTM Cells** per Layer.



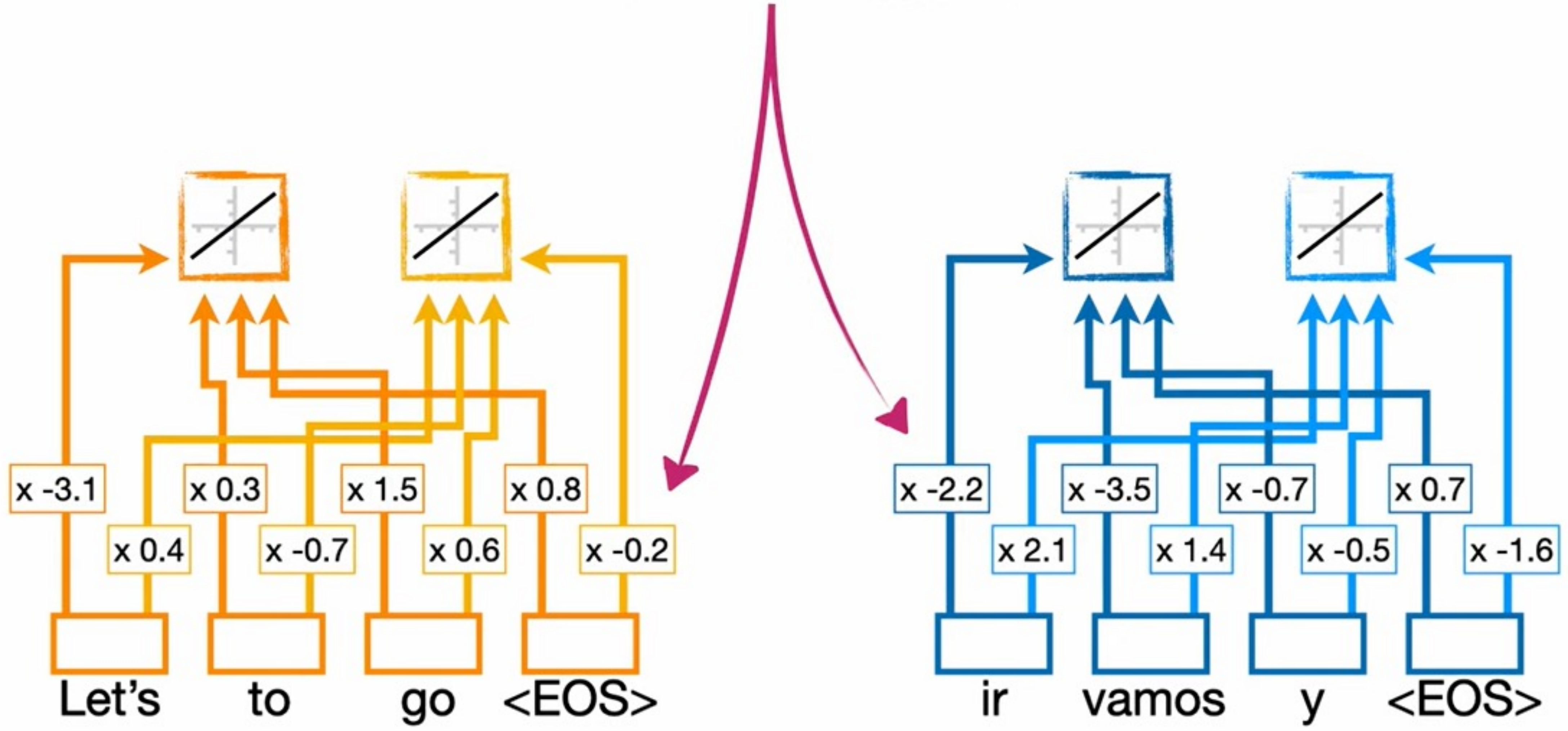


Encoder
Layer 2
0
0
Layer 1
0
0
1 0 0 0
let's to go <EOS>
0 0 1 0
let's to go <EOS>

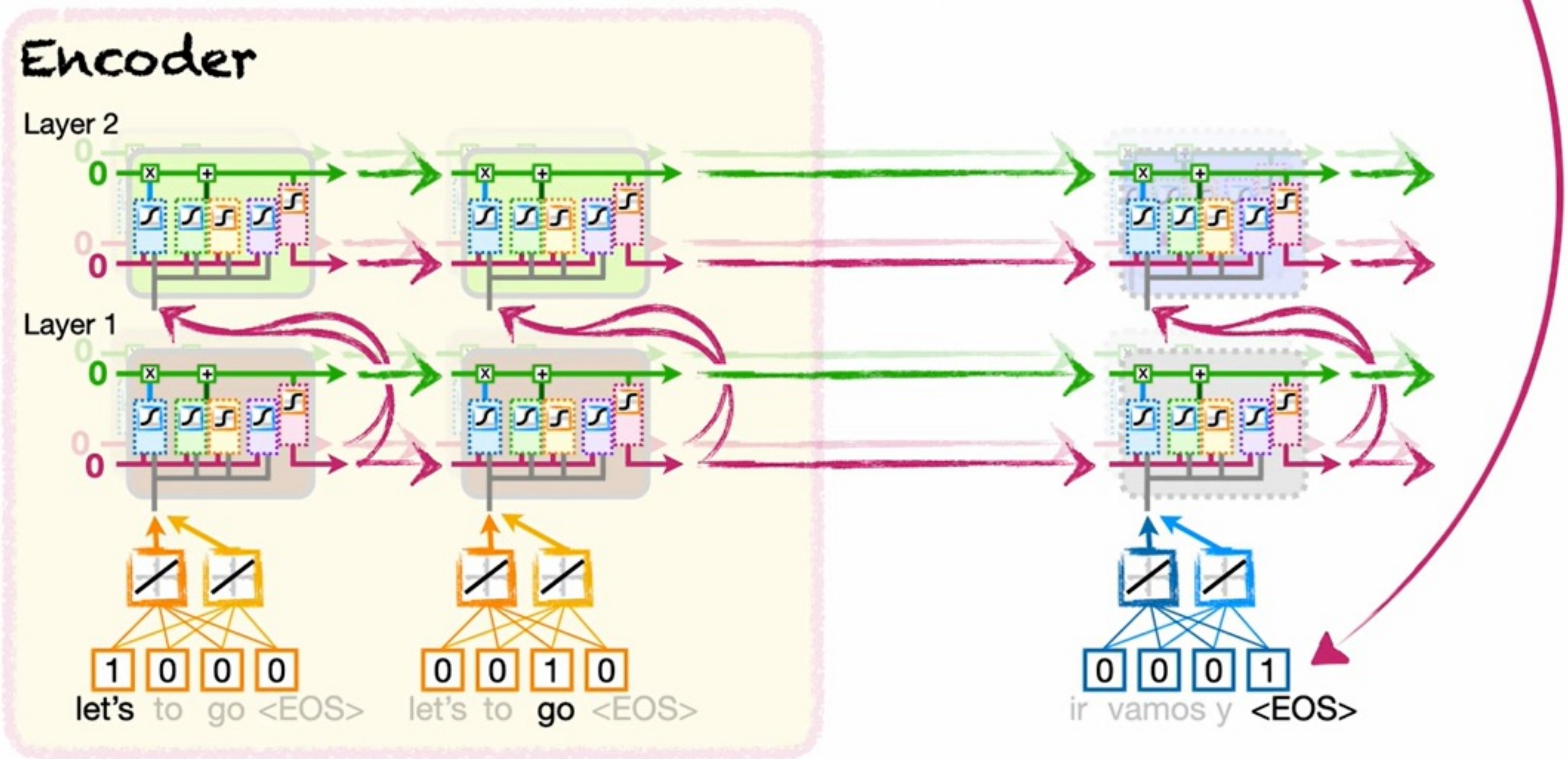
Anyway, the **Context Vector** is used to initialize the long and short-term memories (the cell and hidden states) in the **LSTMs** in the **Decoder**.



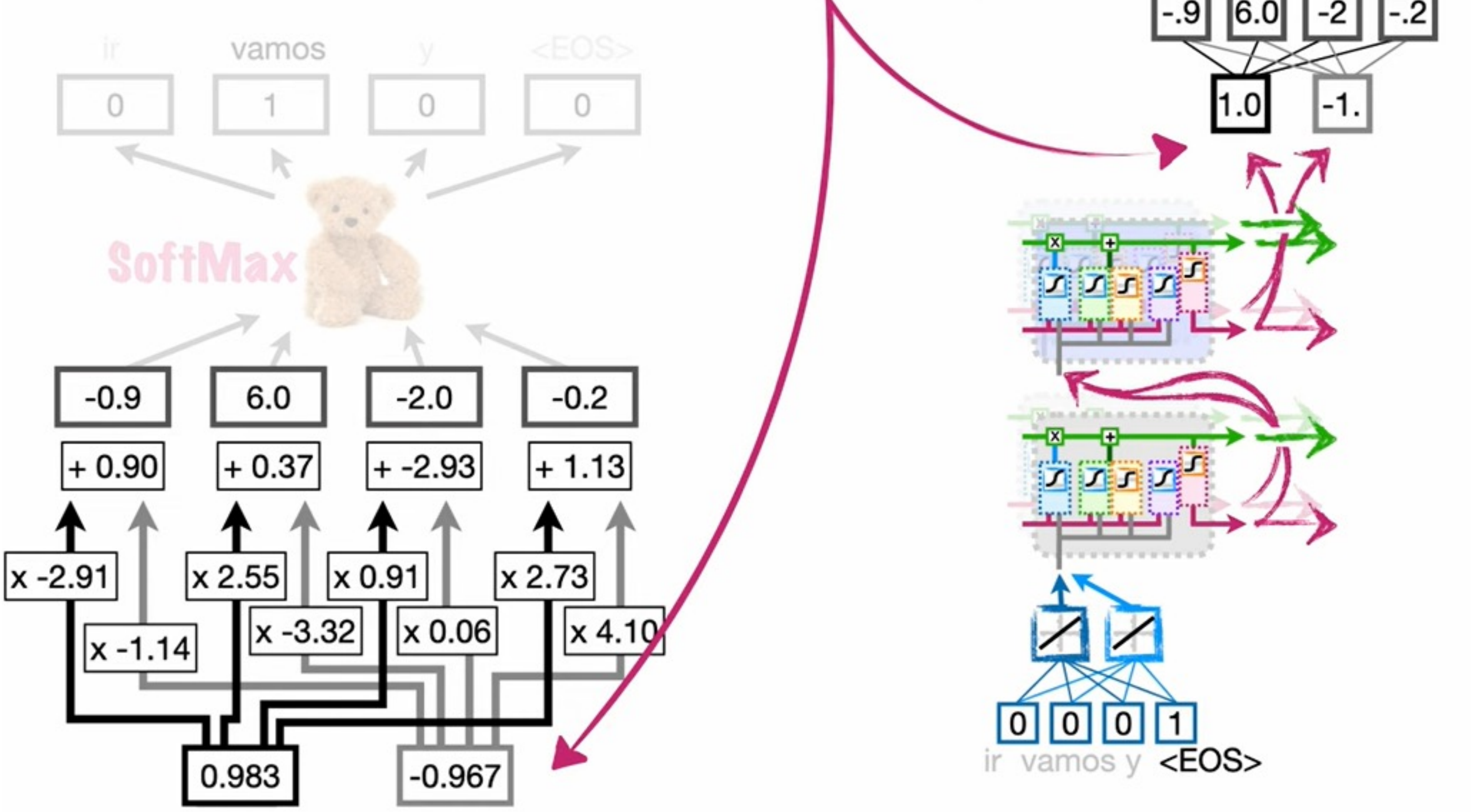
…and different **Weights**, which result in different embedding values for each **Token**.



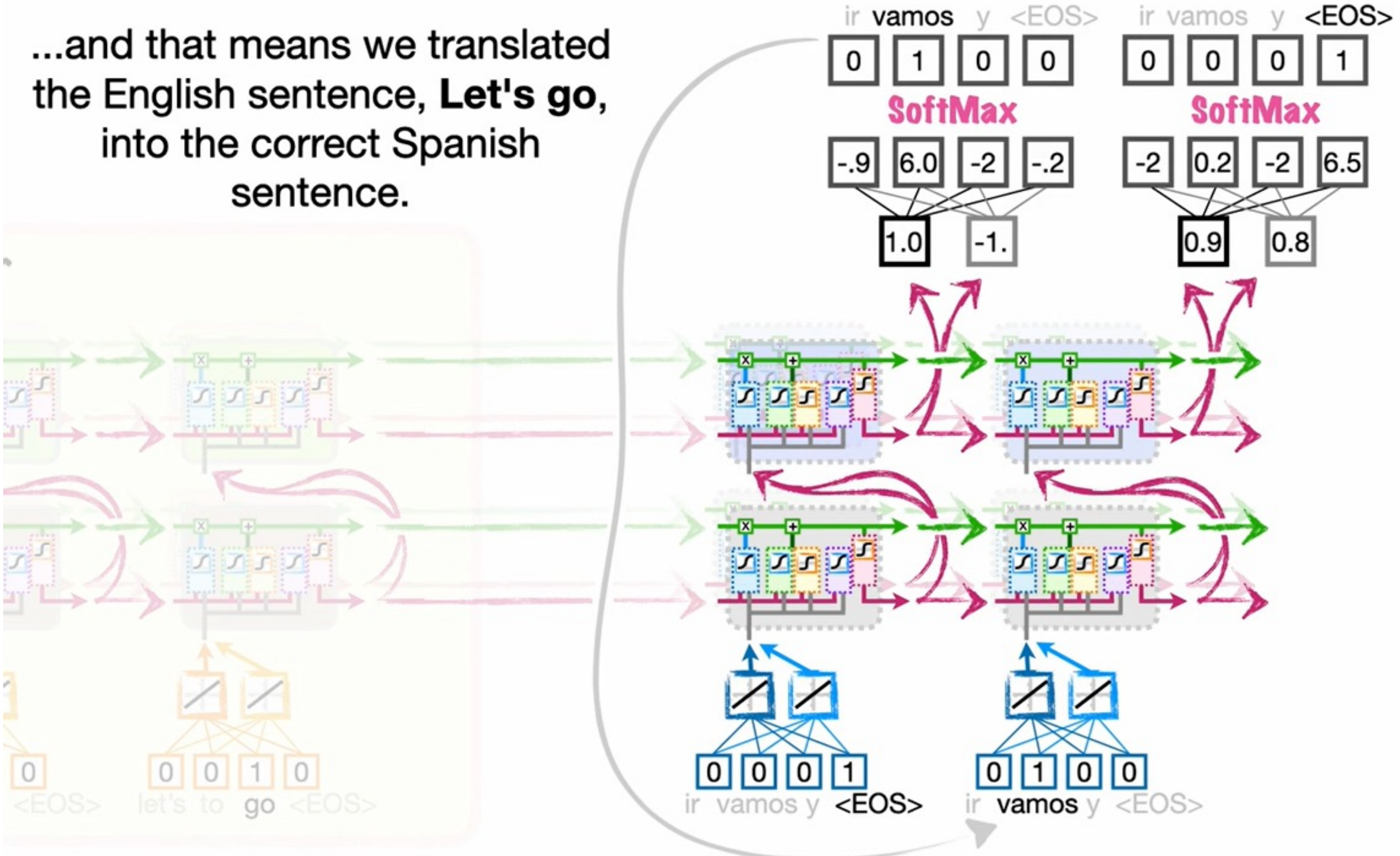
…the **Decoder** starts with the embedding values for the **<EOS>** (end of sentence) **Token**.



This **Fully Connected Layer** has **2** inputs for the **2** values that come from the **LSTM** cells in the top layer...



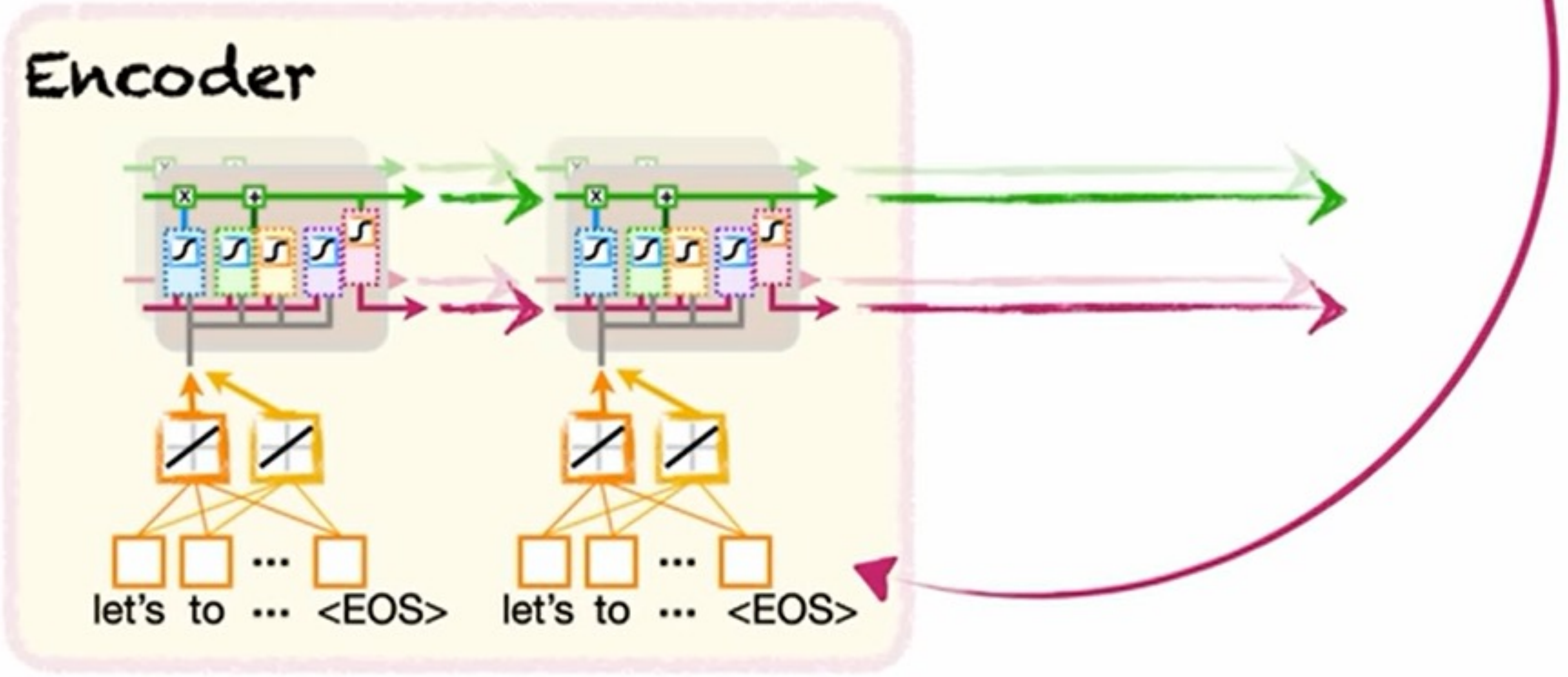
...and that means we translated the English sentence, **Let's go**, into the correct Spanish sentence.



Attention Mechanism

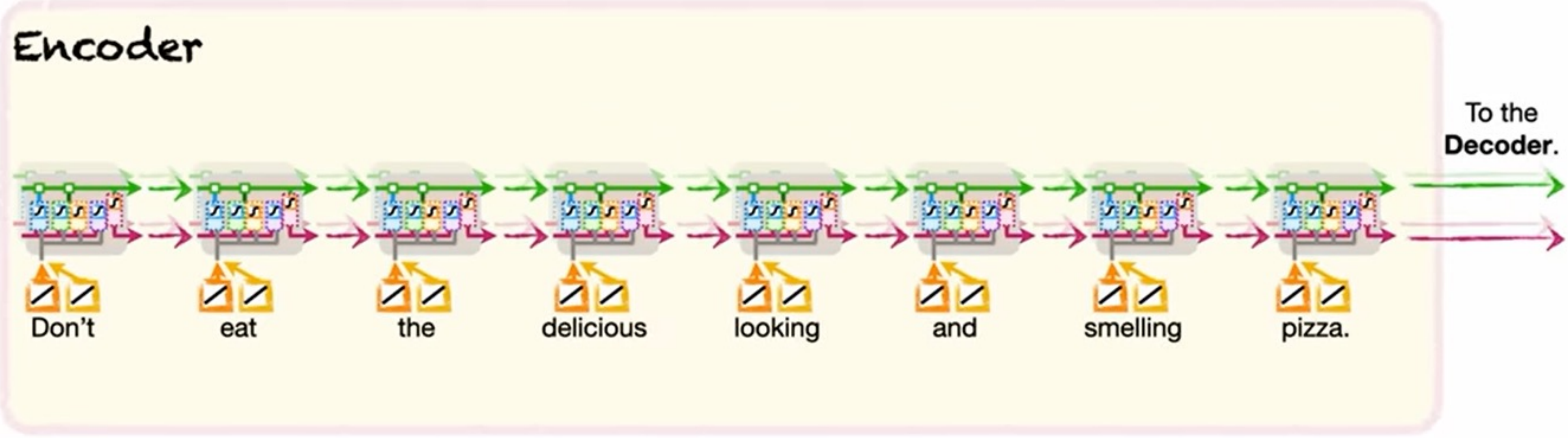
...but if we had a bigger input vocabulary with **1000**'s of words...

...unrolling the **LSTMs** compresses the entire input sentence into a single context vector.

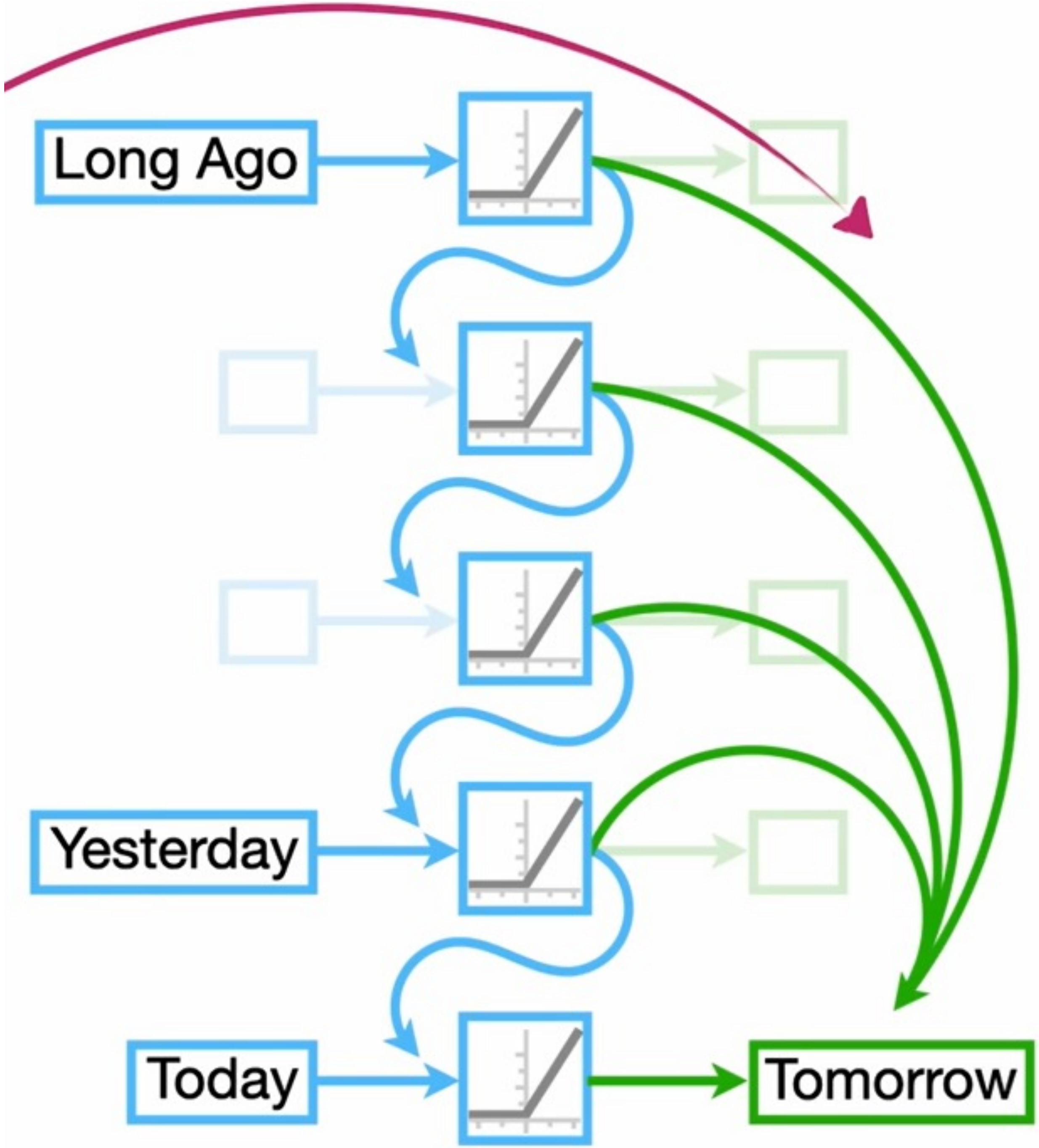


...then **Don't eat the delicious looking and smelling pizza** turns into...

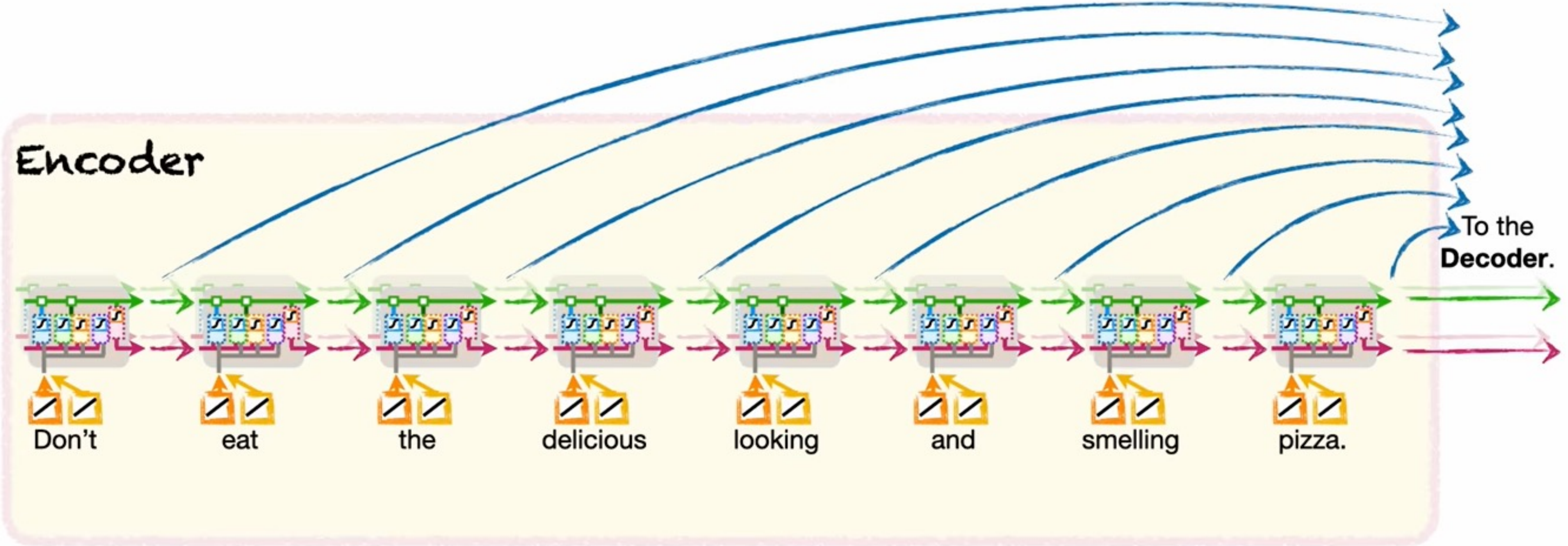
...**Eat the delicious looking and smelling pizza**...



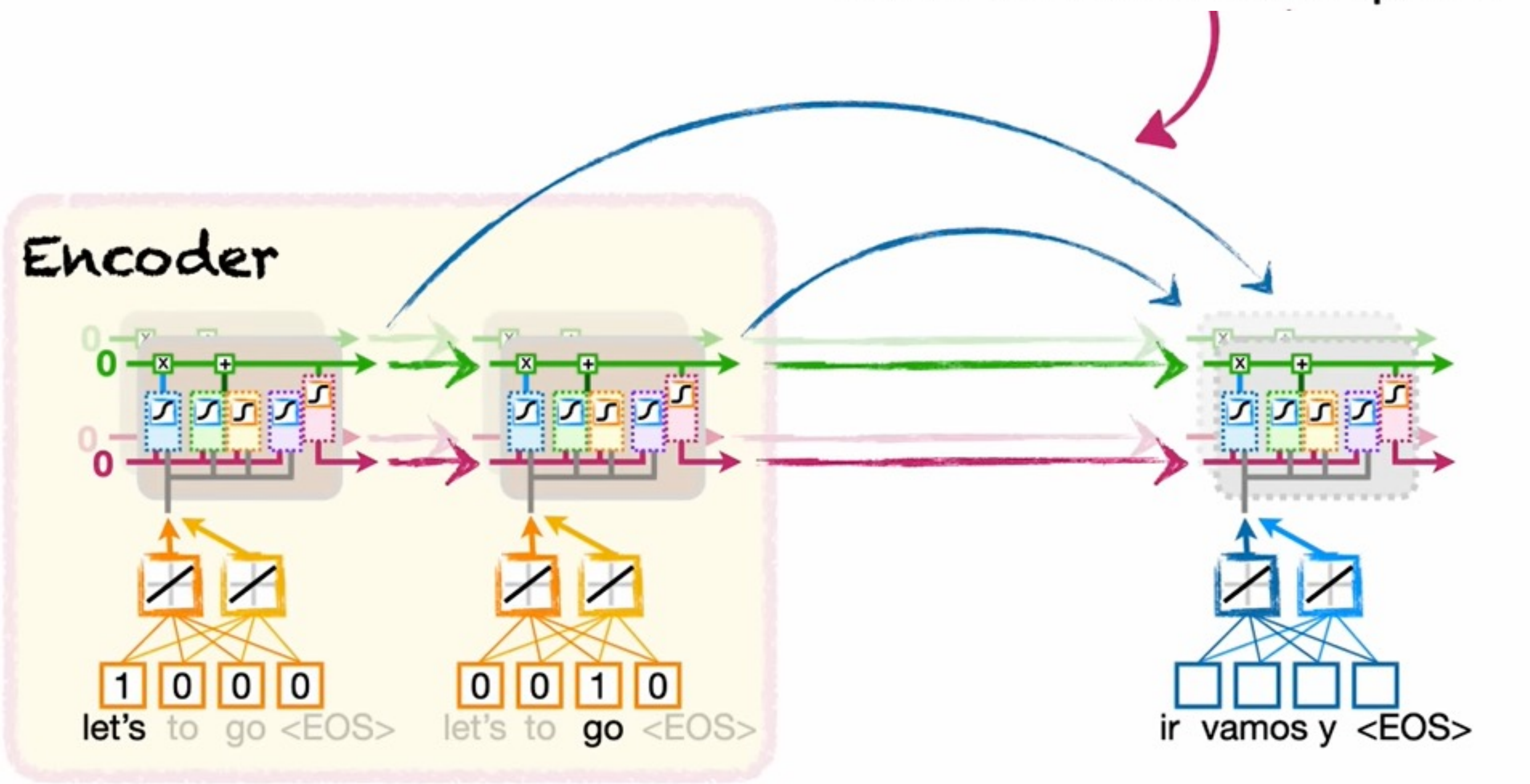
…and that **The Main Idea** of **Long, Short-Term Memory** units is that they solve this problem by providing separate paths for long and short term memories.



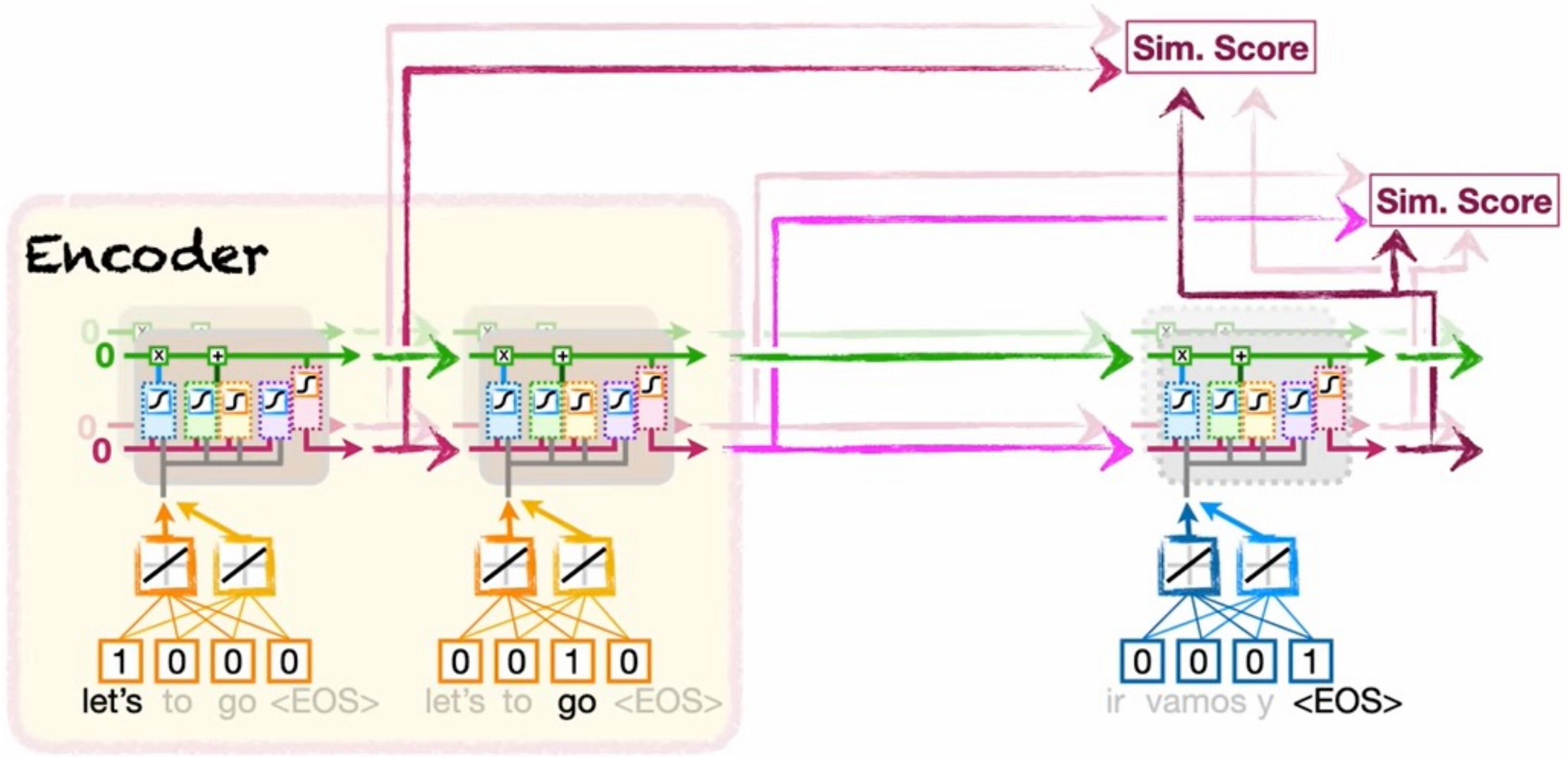
So the **Main Idea** of **Attention** is to add a bunch of new paths from the **Encoder** to the **Decoder**, one per input value, so that each step of the **Decoder** can directly access input values.



...however, the idea of **Attention** is for each step in the **Decoder** to have direct access to the inputs.



$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

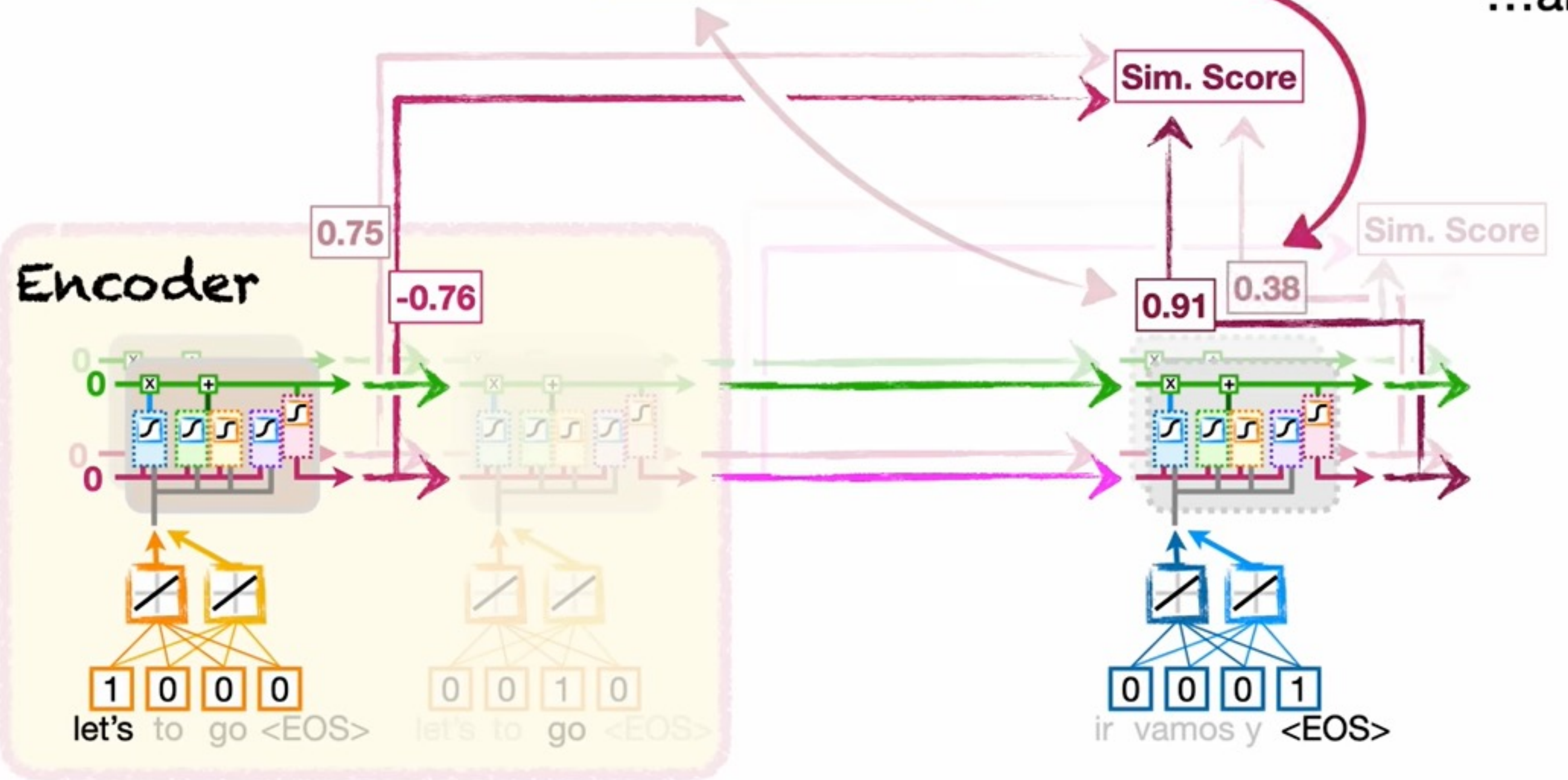


LSTMs

	Cell #1	Cell #2
A = Encoder = Let's	-0.76	0.75
B = Decoder = <EOS>	0.91	0.38

And the output values from the **2 LSTM** cells in the **Decoder** for the **<EOS>** token, are **0.91**…

…and **0.38.**



LSTMs

	Cell #1	Cell #2
A = Encoder = Let's →	-0.76	0.75
B = Decoder = <EOS> →	0.91	0.38

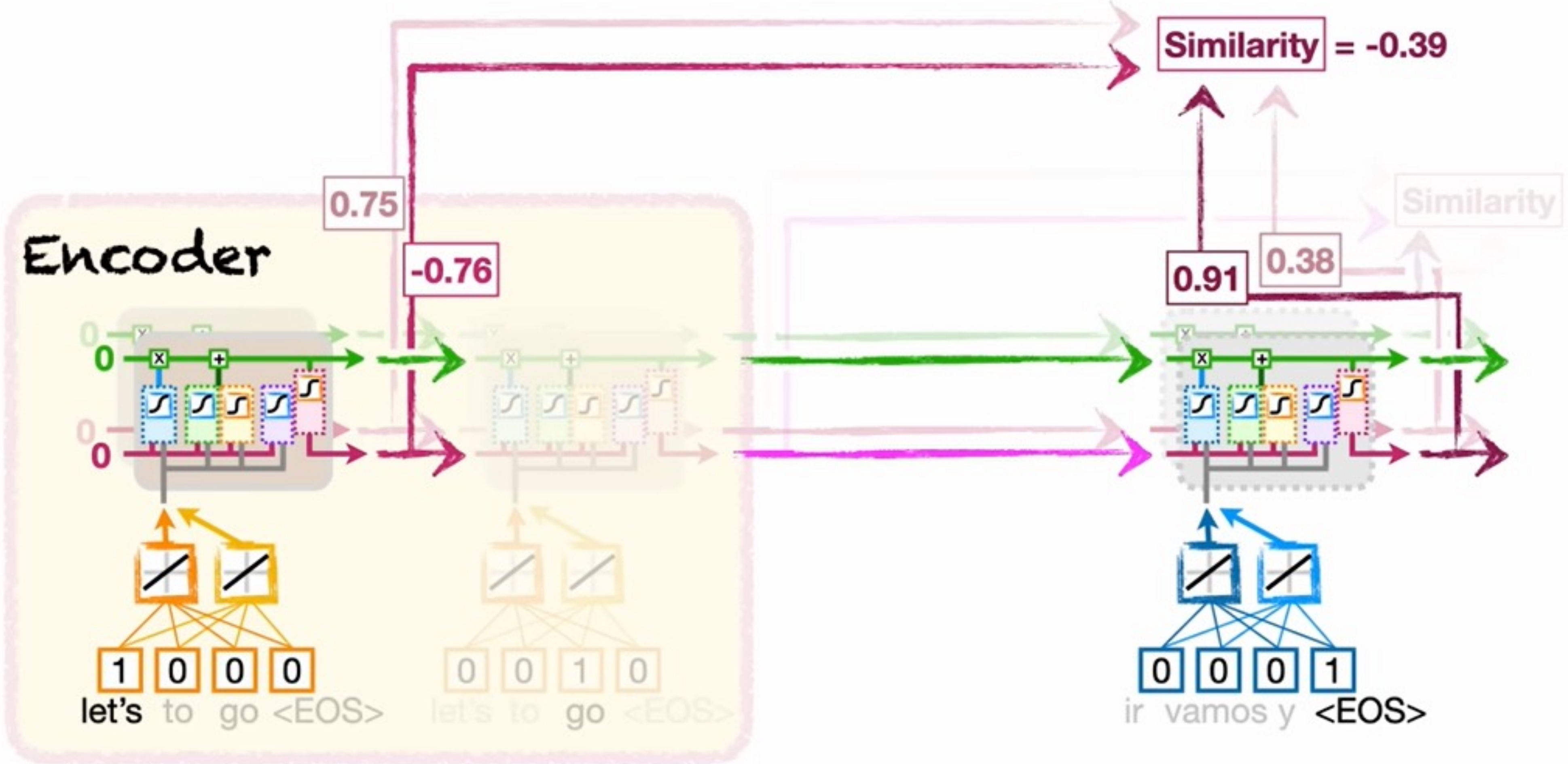
…and we get **-0.39**.

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} = \frac{(-0.76 \times 0.91) + (0.75 \times 0.38)}{\sqrt{-0.76^2 + 0.75^2}\sqrt{0.91^2 + 0.38^2}} = -0.39$$

That being said, a more common way to calculate similarity for **Attention**...

LSTMs

	Cell #1	Cell #2
A = Encoder = Let's →	-0.76	0.75
B = Decoder = <EOS> →	0.91	0.38



LSTMs

	Cell #1	Cell #2
A = Encoder = Let's →	-0.76	0.75
B = Decoder = <EOS> →	0.91	0.38

Anyway, calculating the **Dot Product** is more common than the **Cosine Similarity** for **Attention** because…

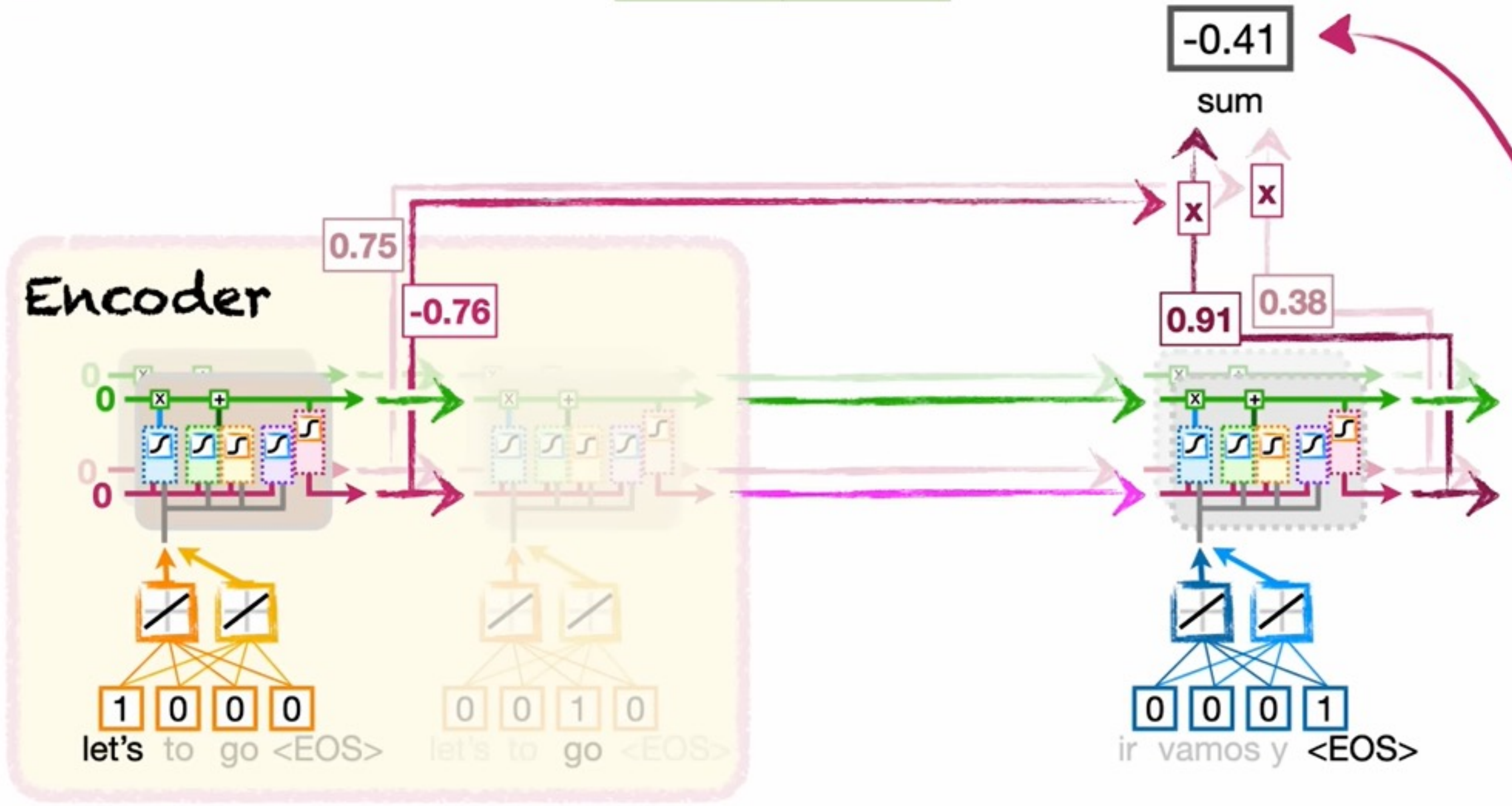
Dot Product $= \sum_{i=1}^{n} A_i B_i = (-0.76 \times 0.91) + (0.75 \times 0.38) = -0.41$

~~Cosine Similarity~~ $= \dfrac{\sum_{i=1}^{n} A_i B_i}{\cancel{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}}$

LSTMs

	Cell #1	Cell #2
A = Encoder = Let's	-0.76	0.75
B = Decoder = <EOS>	0.91	0.38

$$\text{Dot Product} = \sum_{i=1}^{n} A_i B_i = (-0.76 \times 0.91) + (0.75 \times 0.38) = -0.41$$



…and we get **-0.41**.

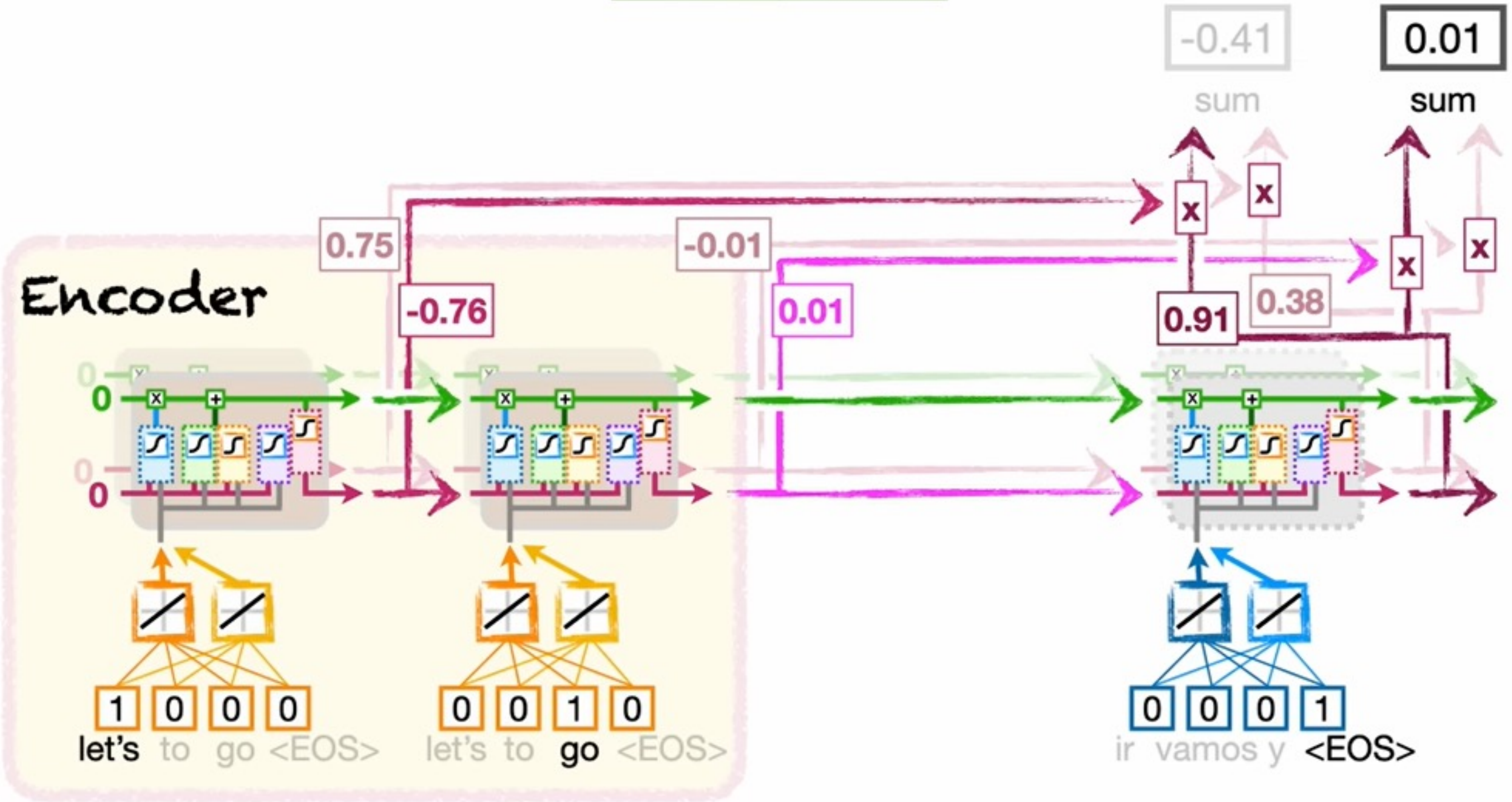
A = Encoder = go

B = Decoder = <EOS>

LSTMs

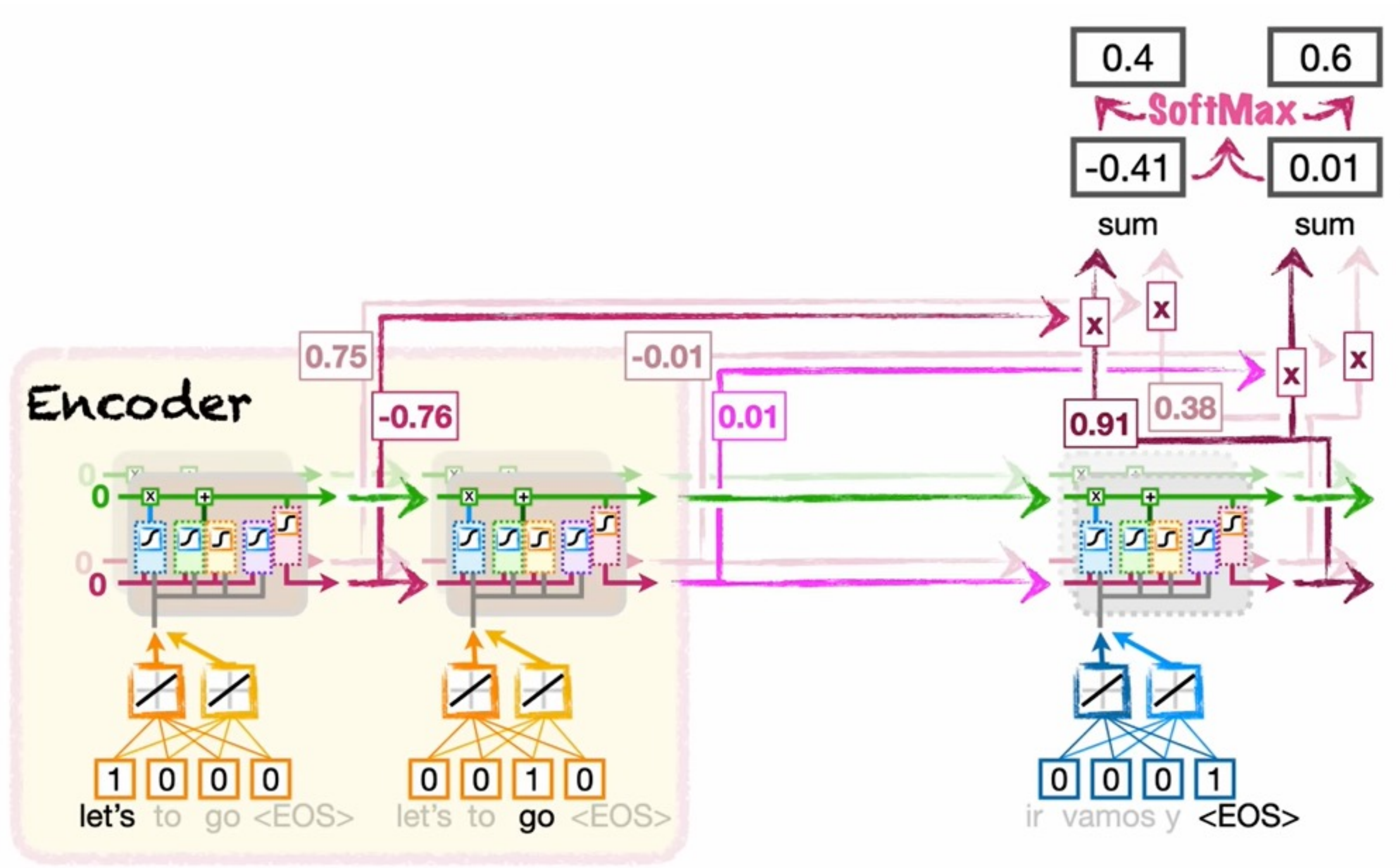
	Cell #1	Cell #2
A = Encoder = go	0.01	-0.01
B = Decoder = <EOS>	0.91	0.38

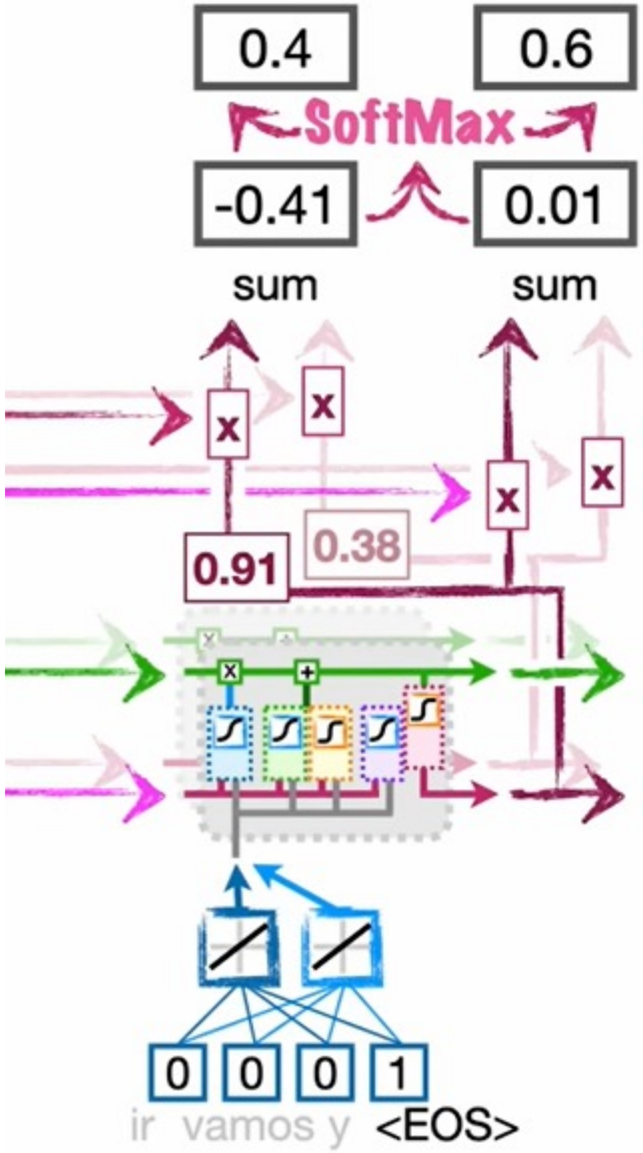
$$\text{Dot Product} = \sum_{i=1}^{n} A_i B_i = (0.01 \times 0.91) + (-0.01 \times 0.38) = 0.01$$



…and we get **0.01**.

So we can think of the output of the **SoftMax** function as a way to determine what percentage of each encoded input word we should use when decoding.

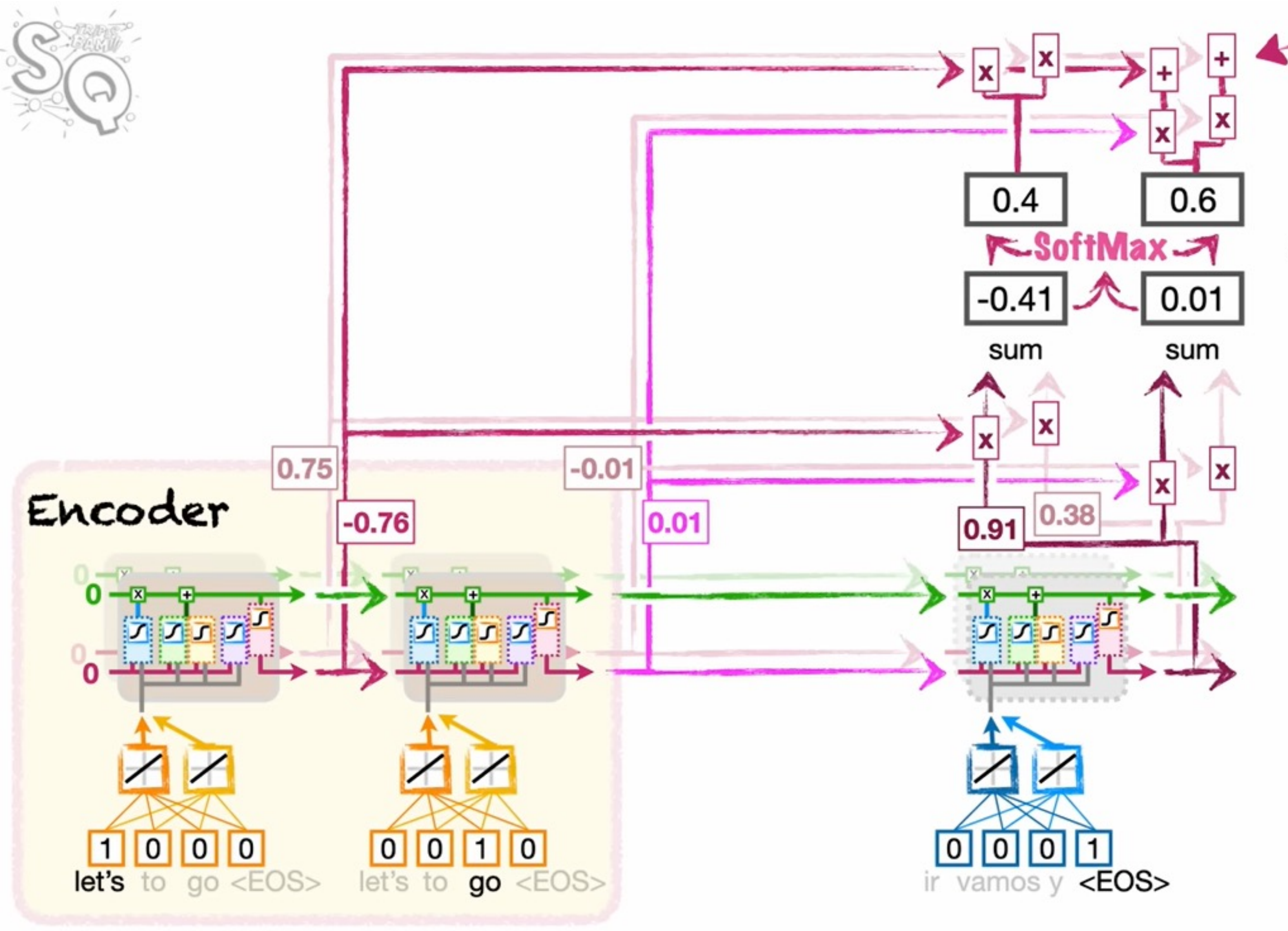




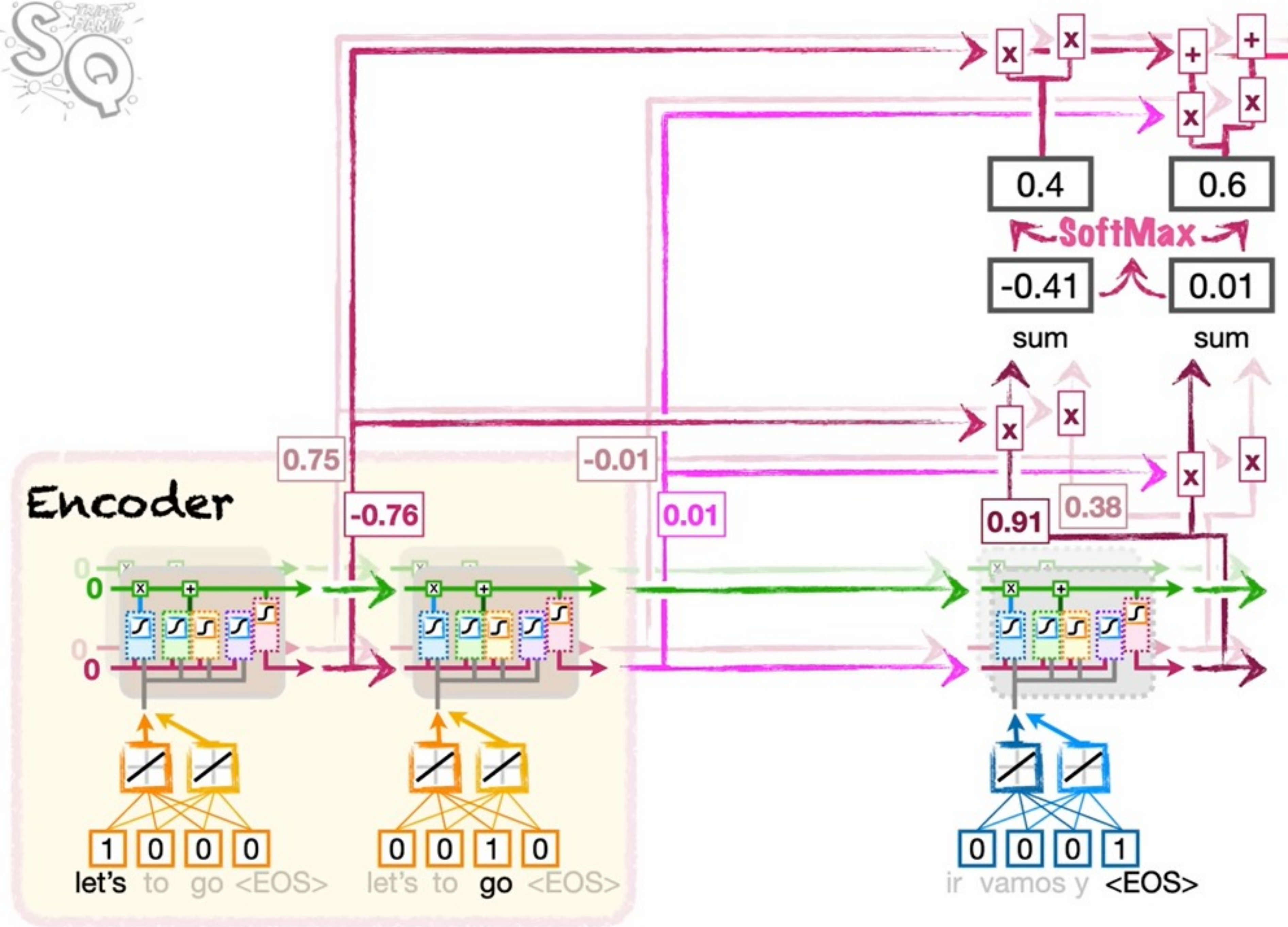
In this case, we'll use **40%** of the first encoded word, **Let's**…

…and **60%** of the second encoded word, **go**…

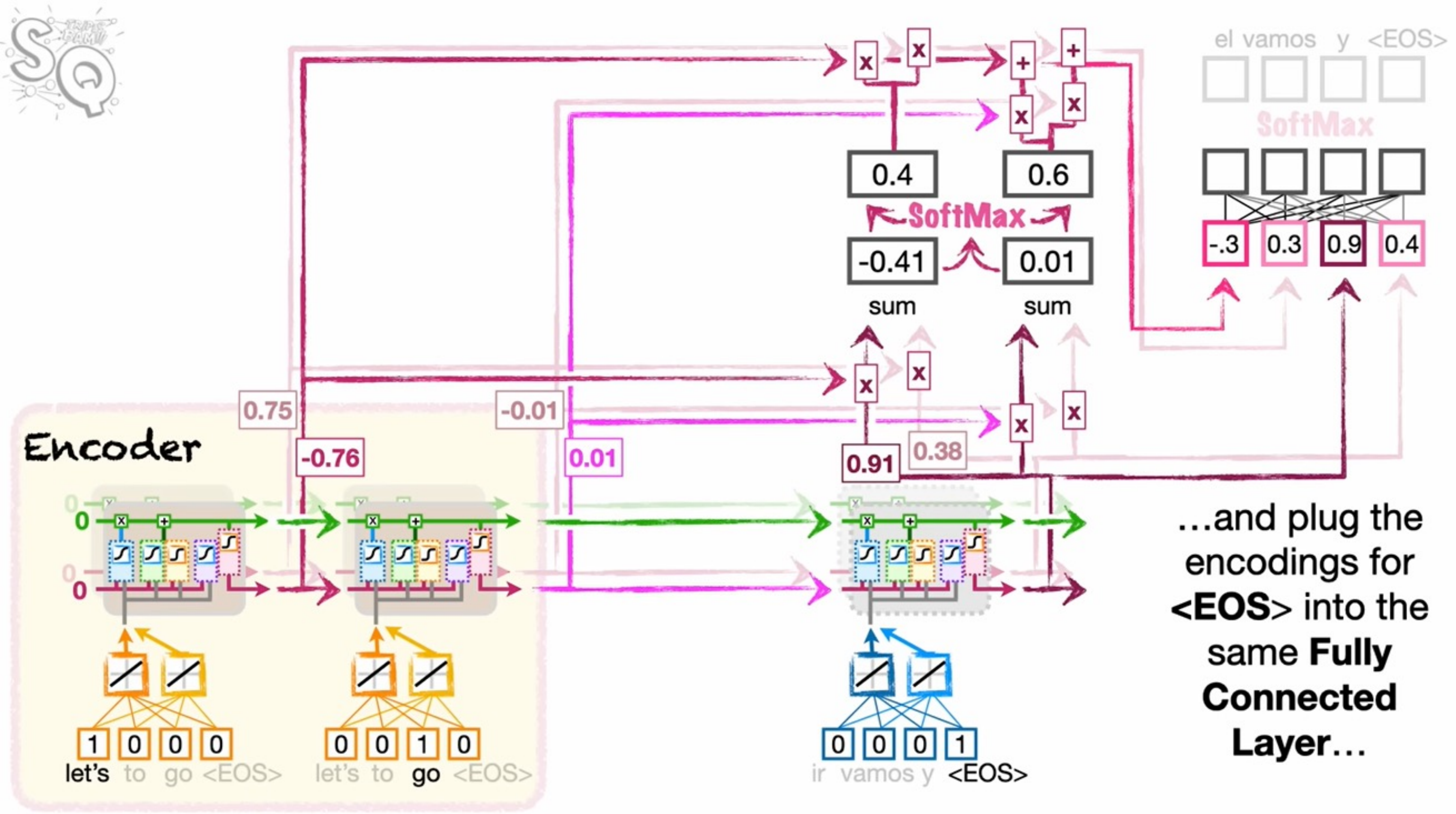
…when the **Decoder** determines what should be the first translated word.



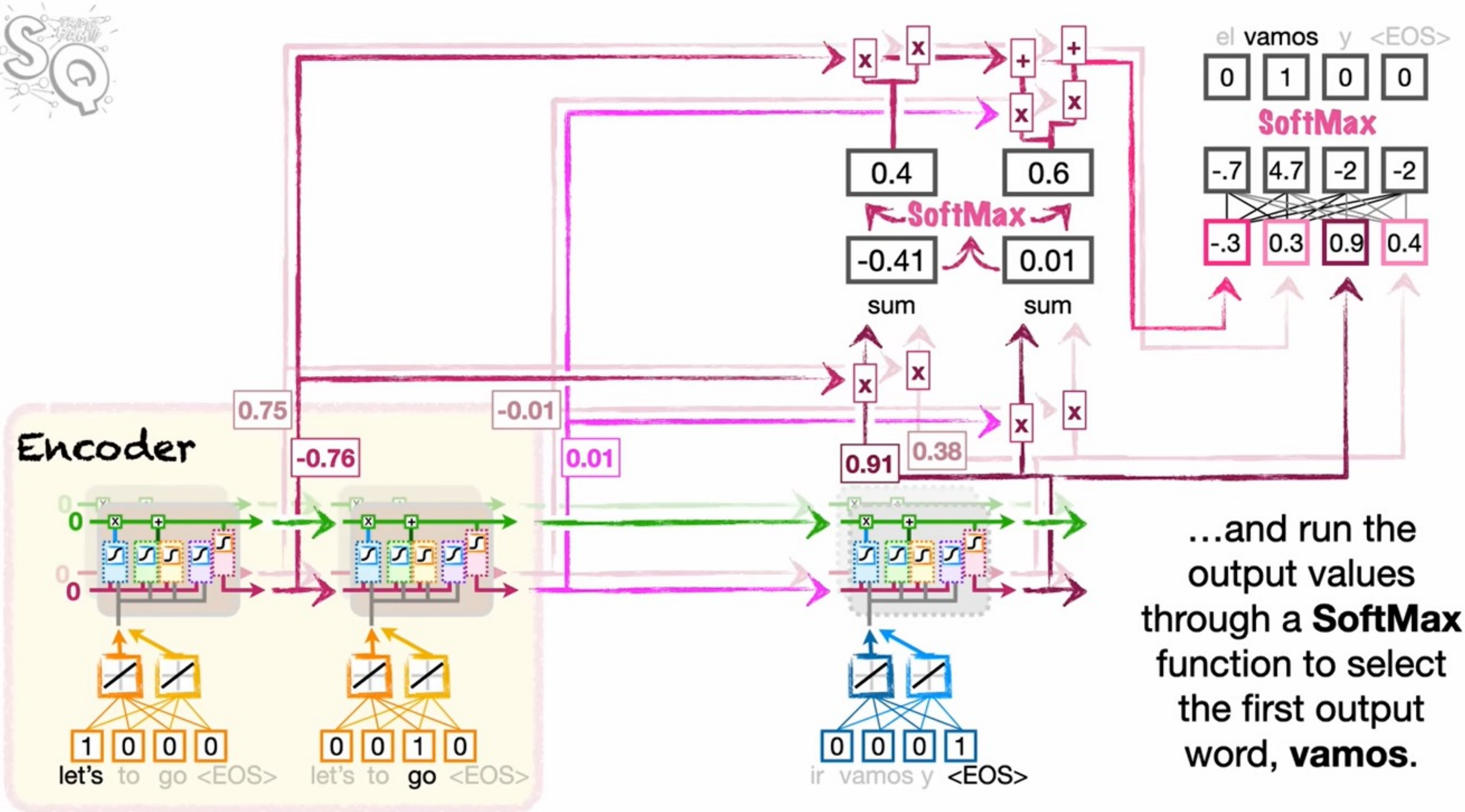
These sums, which combine the separate encodings for both input words, **Let's** and **go**, relative to their similarity to **<EOS>**, are the **Attention** values for **<EOS>**.



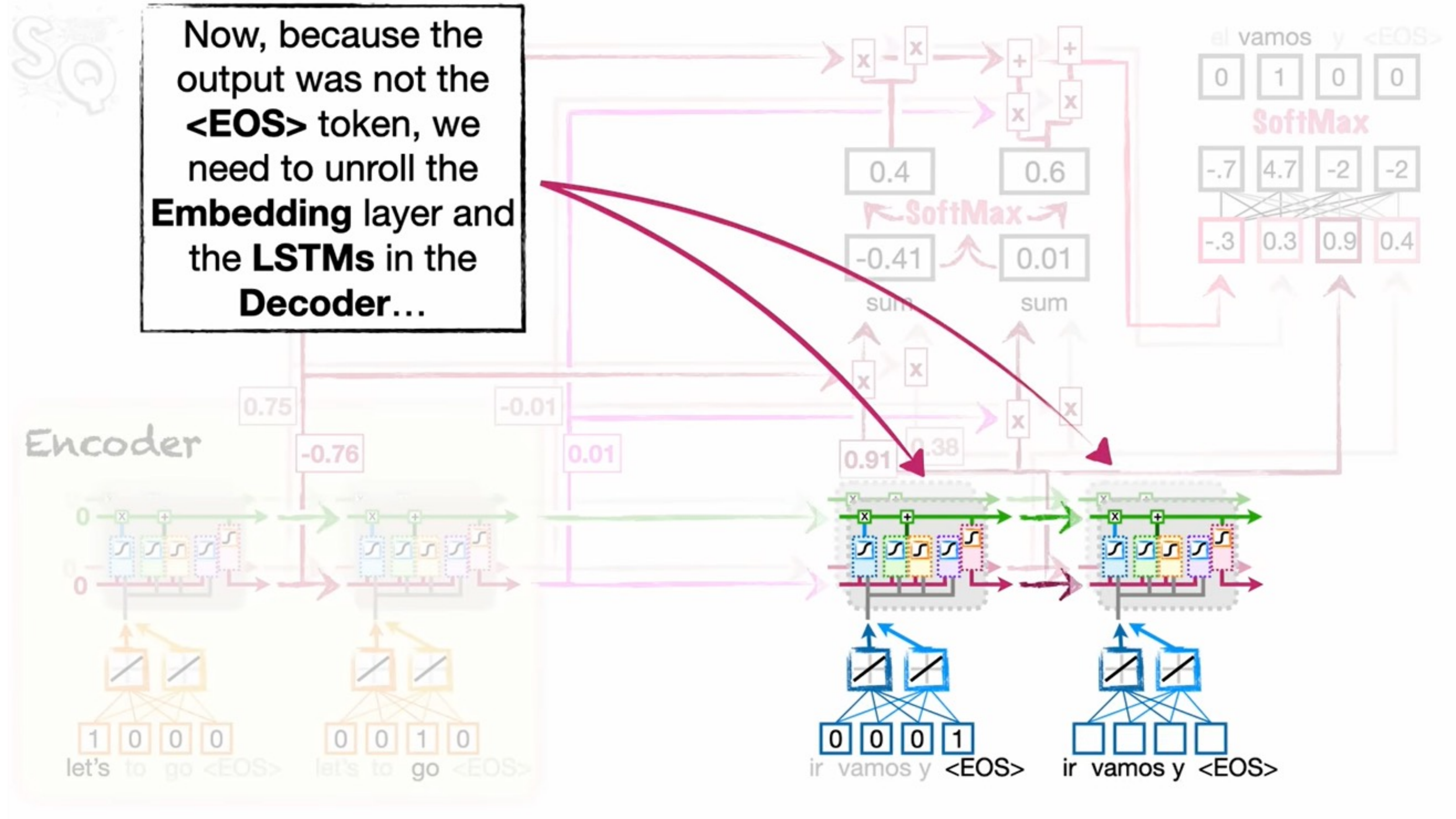
Now, all we need to do to determine the first output word is plug the **Attention** values into a **Fully Connected Layer**…



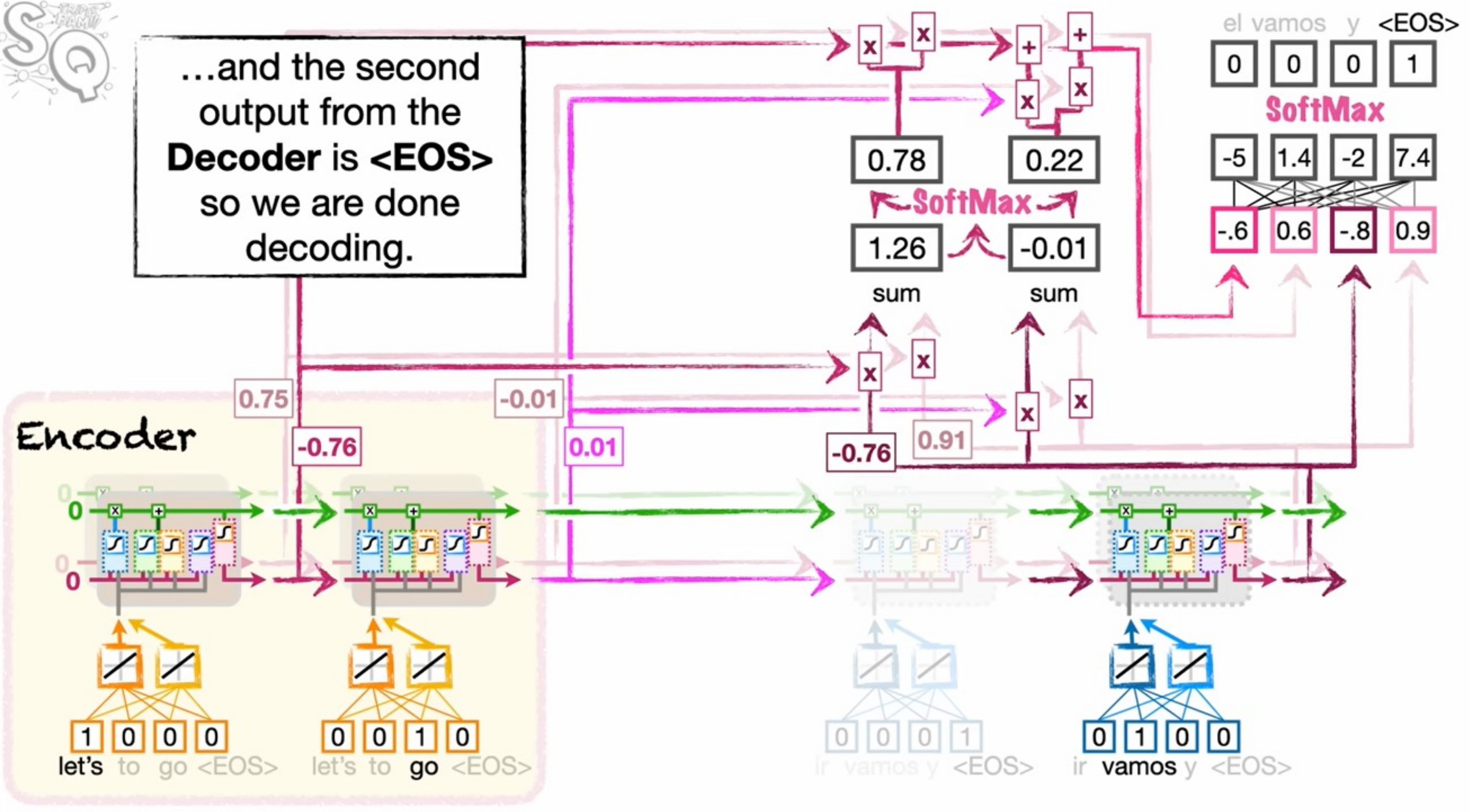
…and plug the encodings for **<EOS>** into the same **Fully Connected Layer**…



…and run the output values through a **SoftMax** function to select the first output word, **vamos**.


Now, because the output was not the <EOS> token, we need to unroll the Embedding layer and the LSTMs in the Decoder…
vamos
0 1 0 0
SoftMax
-.7 4.7 -2 -2
-.3 0.3 0.9 0.4
0.4 0.6
SoftMax
-0.41 0.01
sum sum
0.75 -0.01
-0.76 0.01
0.91 0.38
Encoder
1 0 0 0
let's
0 0 1 0
go
0 0 0 1
ir vamos y <EOS>
ir vamos y <EOS>

...and the second output from the **Decoder** is **<EOS>** so we are done decoding.



In summary, when we add **Attention** to a basic **Encoder-Decoder** model…

…the **Encoder** pretty much stays the same…

…but now, each step of decoding has access to the individual encodings for each input word…

…and we use similarity scores and the **SoftMax** function to determine what percentage of each encoded input word should be used to help predict the next output word.