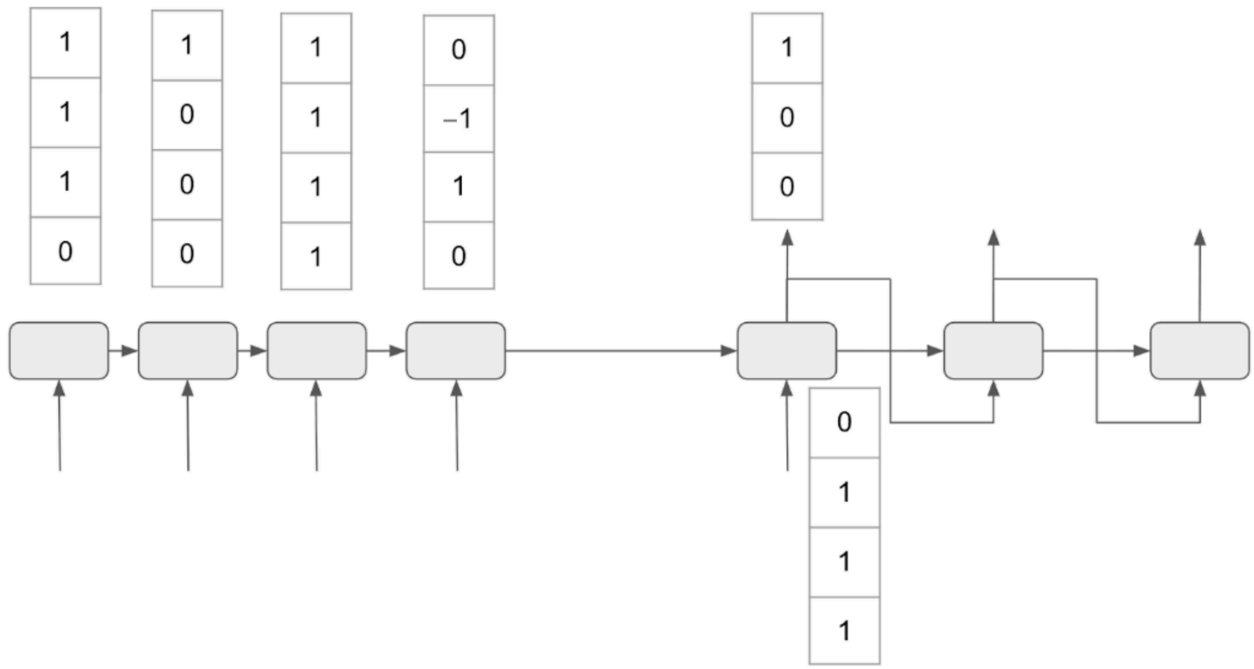
This one math should cover both atteSntion and RNN math issues. Read the explanation carefully.



Here is your RNN attention encoder-decoder. Your job is to generate the output vector for the second (i = 2) hidden unit of the decoder (o_2^d). Here is all the data you have:

$h_1^e = [1\ 1\ 1\ 0]^T$
$h_2^e = [1\ 0\ 0\ 0]^T$
$h_3^e = [1\ 1\ 1\ 1]^T$
$h_4^e = [0\ {-1}\ 1\ 0]^T$
$h_1^d = [0\ 1\ 1\ 1]^T$
$W^d = [0\ 0\ 0,\ 1\ 0\ 0,\ 1\ 1\ 1,\ 1\ 0\ 1]$
$U^d = [1\ 1\ 1\ 1,\ 0\ 0\ 0\ 0,\ 1\ 0\ 1\ 0,\ 0\ 1\ 0\ 1]$
$V^d = [1\ 1\ 0\ 0,\ 0\ 0\ 0\ 0,\ 1\ 1\ 1\ 0]$
$o_1^d = [1\ 0\ 0]^T = \hat{y}_1$

So, how do you do it?

To calculate o_2^d, you need the equation from our basic RNN lecture (slide 6_rnn.pptx, page 7):

$o_2^d = \text{softmax}(Vh_2^d)$ [1]

V is a 3x4 matrix, and when multiplied by h_2^d, which should be a 4x1 matrix (why? Because h_1^d is a 4x1 matrix and all h_i^d has to be the same shape), you should get a 3x1 matrix, which is the shape of o_1^d. Hence, the shapes make sense.

Now, to solve [1], you need V and h_2^d. V is given but h_2^d is not. So let's calculate h_2^d. How? We will use equation 9.35 from the book. But it is a general equation, and let me give a specific version of it. Instead of 9.35, use [2]:

$h_i^d = \tanh(W^d\hat{y}_{i-1} + Uh_{i-1}^d + c_i)$ [2]

What is our i? 2, right? So let's rewrite [2] for our index:

$h_2^d = \tanh(W^d\hat{y}_1 + U^dh_1^d + c_2)$ [3]

W^d has a shape of 4x3, $\hat{y}_1$ has a shape of 3x1, so the multiplication result will have a shape of 4x1. U^d is 4x4, h_1^d is 4x1, so this result will also have a shape of 4x1, and as this will be an elementwise addition, c_2 has to be 4x1 as well, and that means, our h_2^d will also be 4x1– which, again, makes perfect sense.

We now have everything but c_2. How do we calculate c2? Let's look at equation 9.38. For c_2, the equation should look like this:
$c_2 = \sum\alpha_{2j}h_j^e$ [4] for all j.

How many j's are there? j is the index for encoder units, and as we have 4 encoder units, j will be from 1 to 4.

But how do you calculate α_{2j}? Use book's equation 9.37 of course:

$\alpha_{2j} = \text{softmax}(\text{score}(h_1^d, h_j^e))$ [5] for all j.

How do you calculate score() now? Easy, use 9.36 from the book:

$\text{score}(h_1^d, h_j^e) = h_1^d.h_j^e$ [6] for all j. The dot between two h's is a dot product.

Now, dot producing two 4x1 vectors will produce a scalar, right? Right. You are already given h_1^d, h_1^e, h_2^e, h_3^e and h_4^e. You multiply all h^e with h_1^d, so you get 4 scalars. Run softmax on these 4 scalars, then you will have 4 values between 0 and 1 that add up to 1. They will be your $\alpha_{21,}$ $\alpha_{22,}$ α_{23} and α_{24}. Now, multiply each alpha with their corresponding h^e (α_{21} with h_1^e, α_{22} with h_2^e and so on) and you will have four 4x1 vectors.

Add them elementwise, and voila, you have got your c_2 from [4], which is also a 4x1 vector.

Insert this c_2 in [3] and you will get h_2^d, and insert this h_2^d in [1] and you get o_2^d. Done.

If all this looks a little overwhelming, take a deep breath, and go slow. It is not as intimidating as it looks.