"Attention"

* Attention is based on Encoder-Decoder network structure or sequence-to-sequence (sec2sec) modelling. Before delving into sec2sec, it is a good idea to briefly talk about word embedding and word2Vec.

* in NLP, if we want to plug in words into a deep network we need a way to turn words into numbers

* We can use random numbers or one-hot vectors to represent words → But this way doesn't capture relationship between words: good ↔ bad, perfect ↔ awesome, and etc.

* Assigning a long vector to a number helps NN to capture different meaning of a word in different contexts:

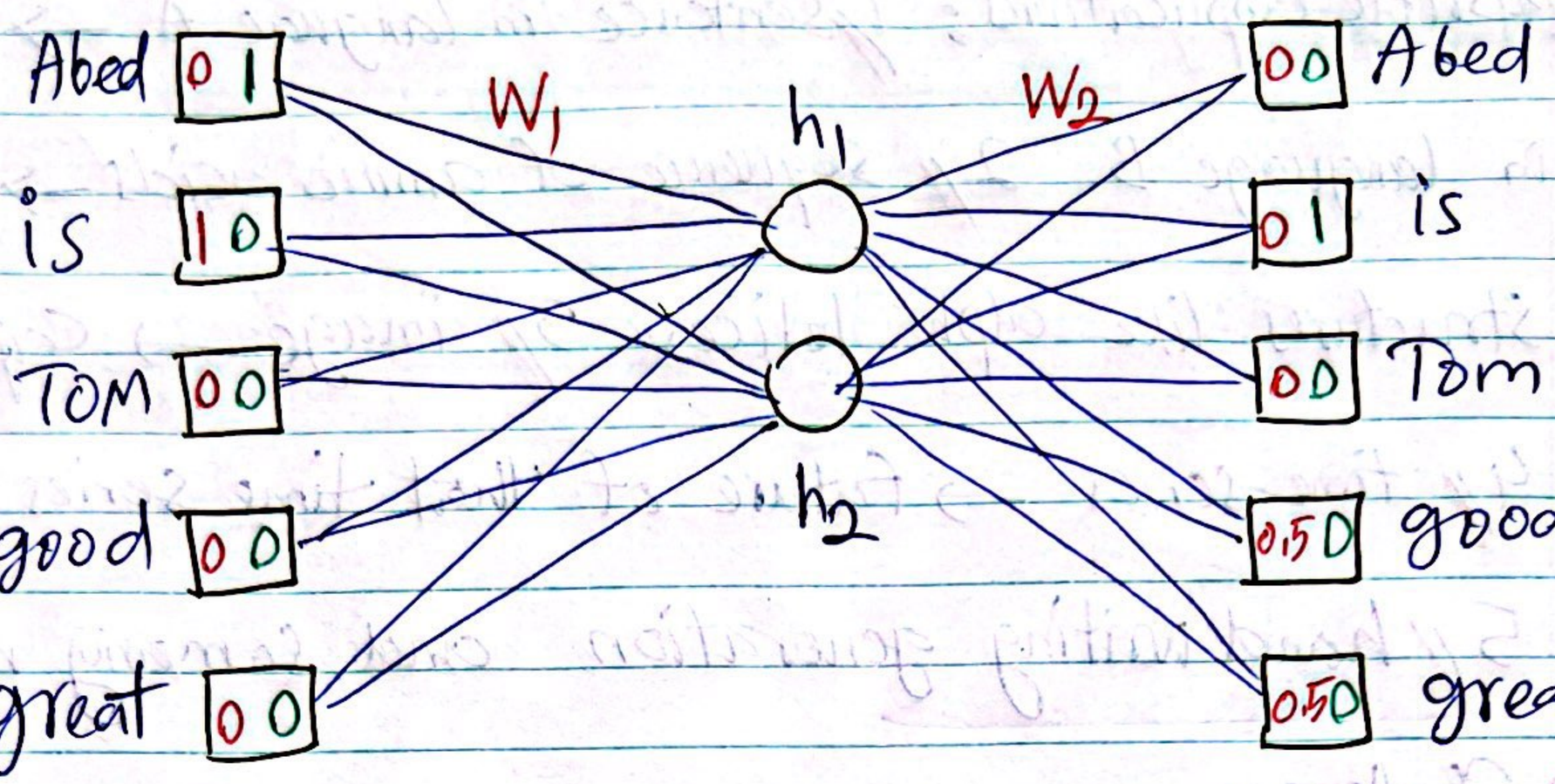
This phone is great! ↔ my phone is broken, great!

* consider that we have this dataset:

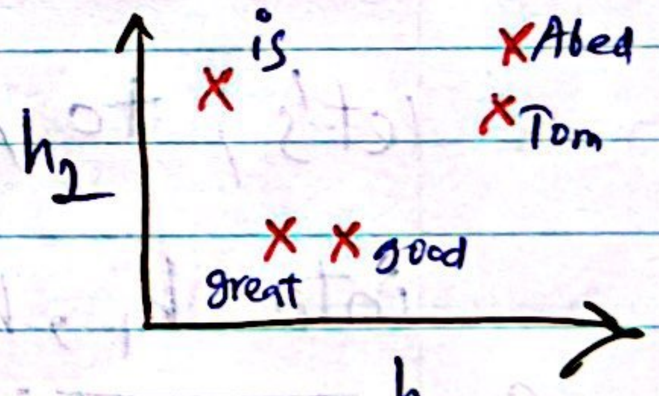
1. Abed is great
2. Tom is good

we want to adjust weighs w_1 and w_2 in a

in



way that for each input word, the network predicts the next word. Using backpropagation, we optimize W_1 and W_2 and the result seems like this:



- Although, similar words are embedded near to each other, this is not a perfect way to embedd words. We have these methods:

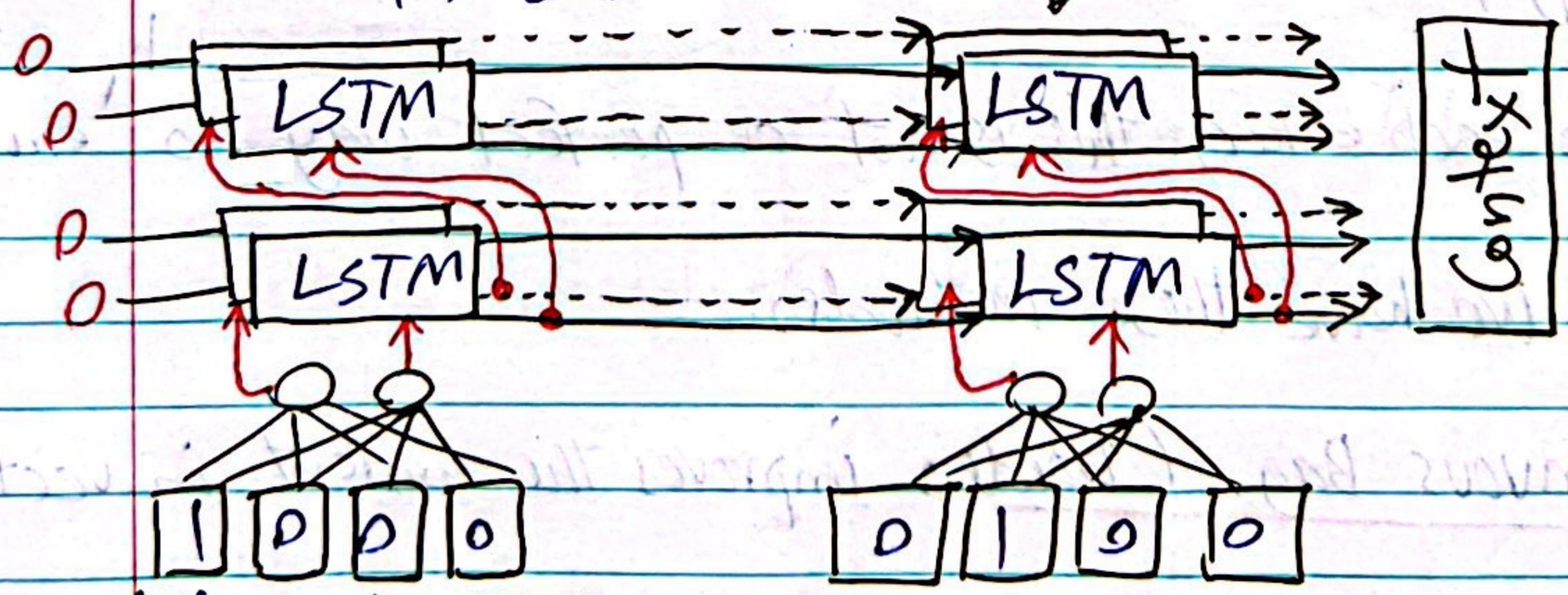
1/ Continuous Bag of Words: improves the context in vector h by using the surrounding words to predict what occurs in the middle

2/ Skip Gram: improves the context in h by using the word in the middle to predict its surroundings

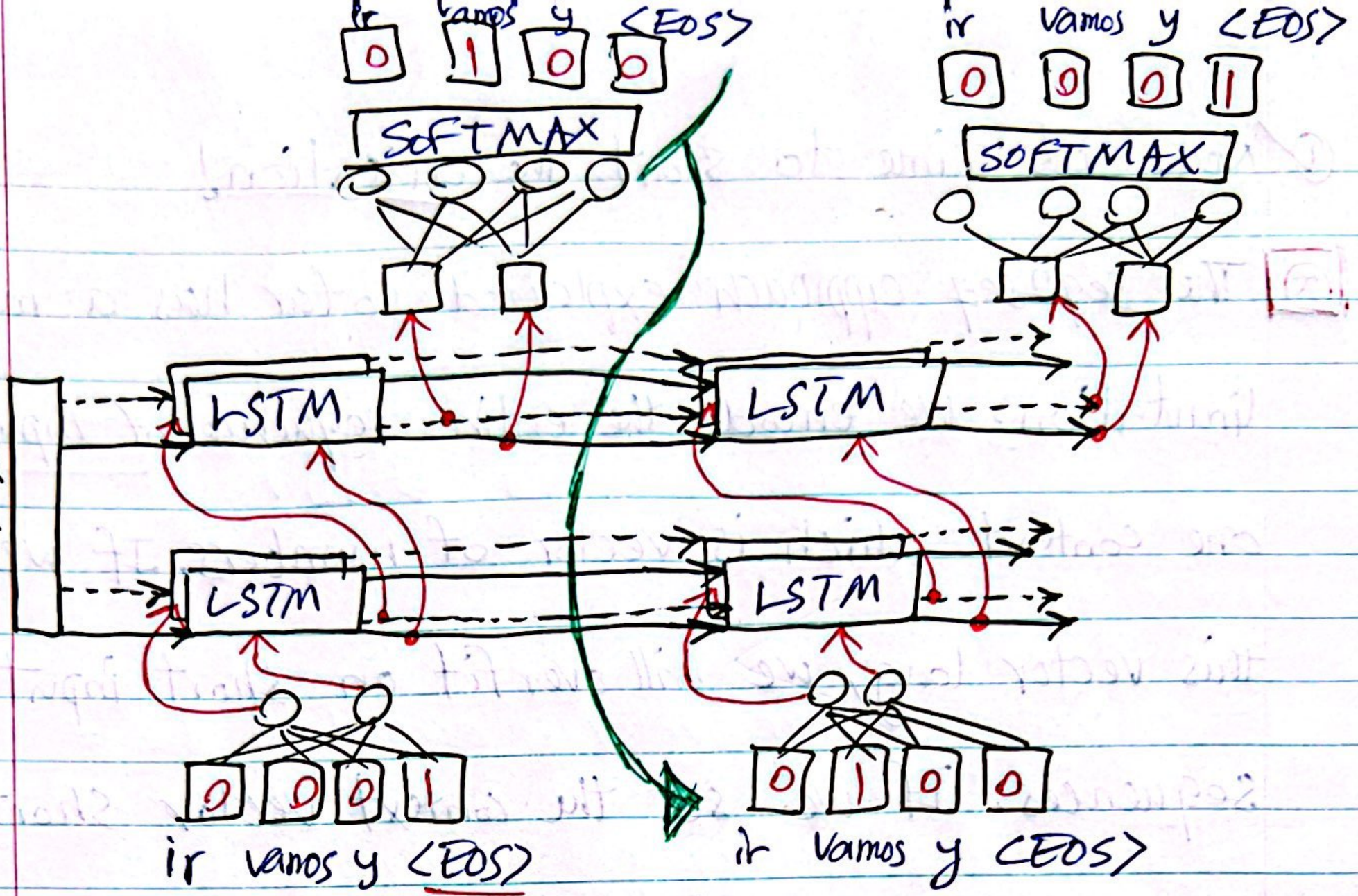
* Now, we will move to seq2seq : these are possible applications: 1// sentence in language A → sentence in language B 2// sequence of amino acids → 3D structures like alpha helices. 3// image → caption 4// time-series → future of that time series 5// hand writing generation and so many more!

* Challenge 3

Different lenght of inputs/outputs → RNN/LSTM can handle this! Let's say, we have 4 tokens let's, to, go, <EOS> and we want to embedd into h_1, h_2!



Now we feed context to decoder: Decoder has the structure of LSTMs with different weights.

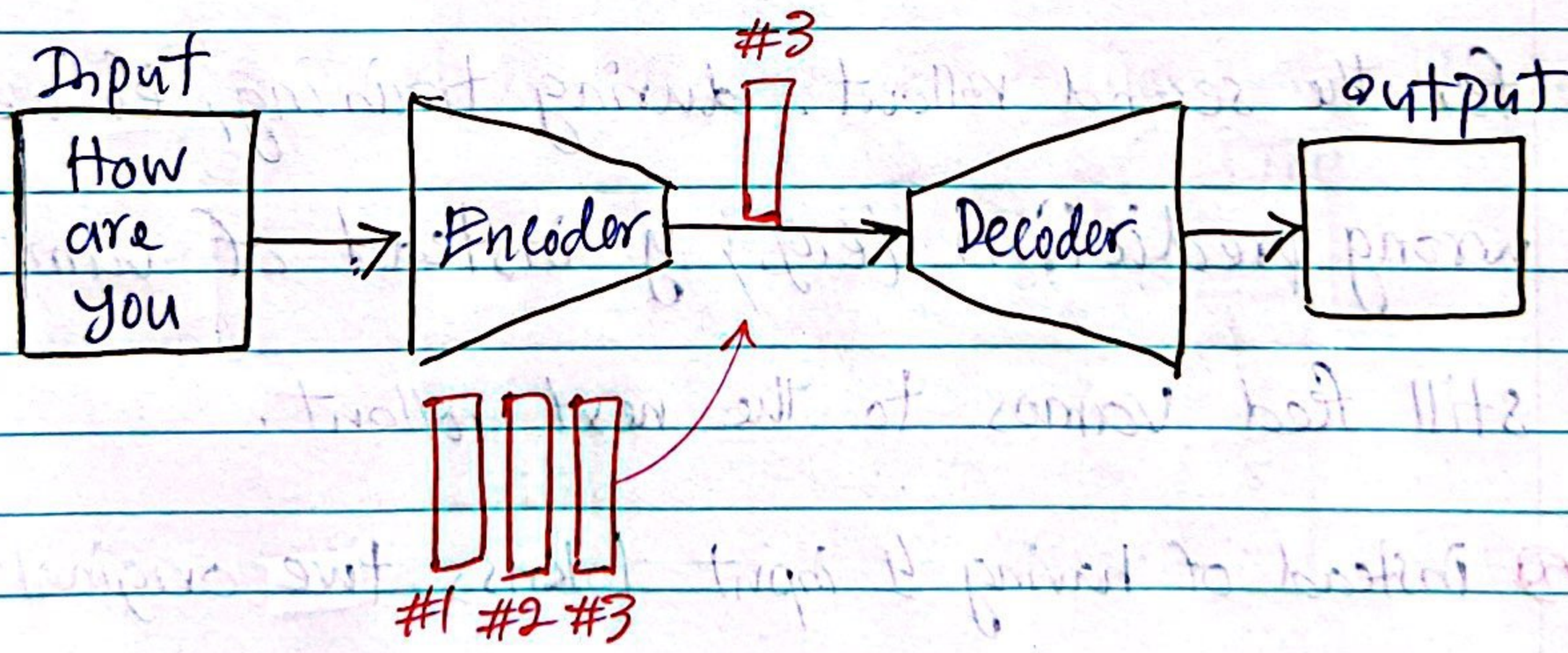


always we start with Eos to start the decoder

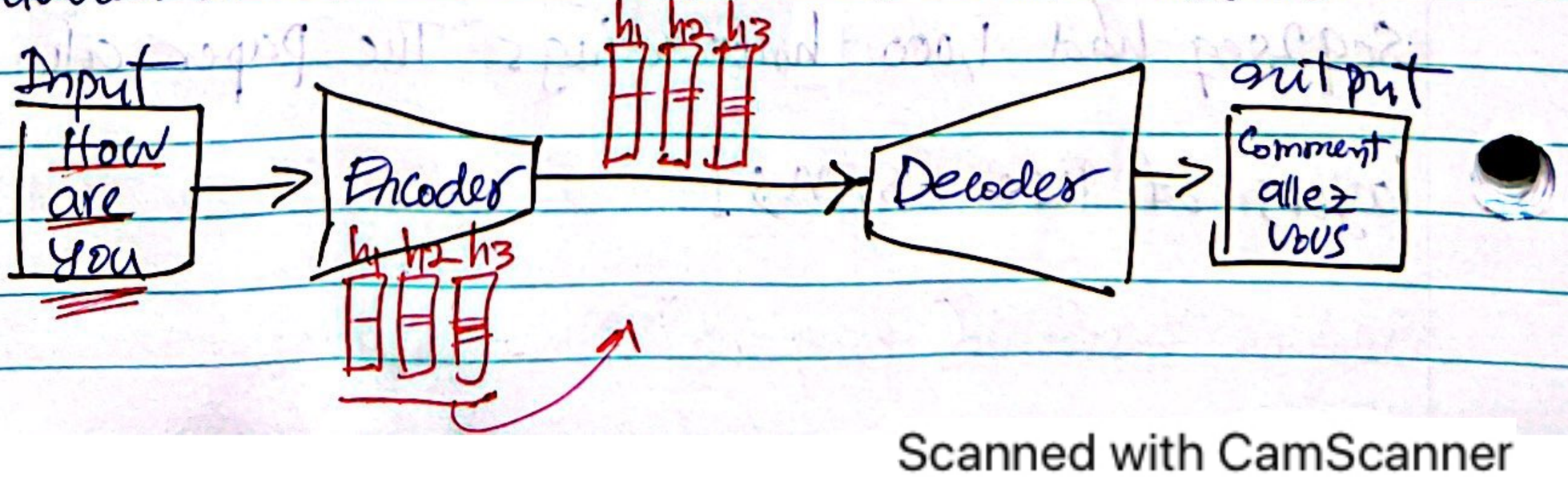
- As you see, the predicted word vamos is used as input for the second rollout. during training, if we have a wrong prediction (e.g., y instead of vamos) we still feed vamos to the next rollout.
- instead of having 4 input tokens, the original paper has 160,000 tokens! The output vocabulary had 80,000 tokens. instead of having two embedding values, the original seq2seq had 1,000 embeddings. The paper also had 4 layers of 1000 LSTMs!

① Now, it's time to start the attention!

② The seq2seq approach explained so far has a major limitation. We encode the entire sequence of input into one context which is vector of numbers. If we set this vector long, we will overfit on short input sequences. If we set the context vector short, we cannot encode large sentences and we may forget concepts at the start of the input when we reach the end of sentence!



However in attention-based models we feed all of the hidden states into decoder as context vectors



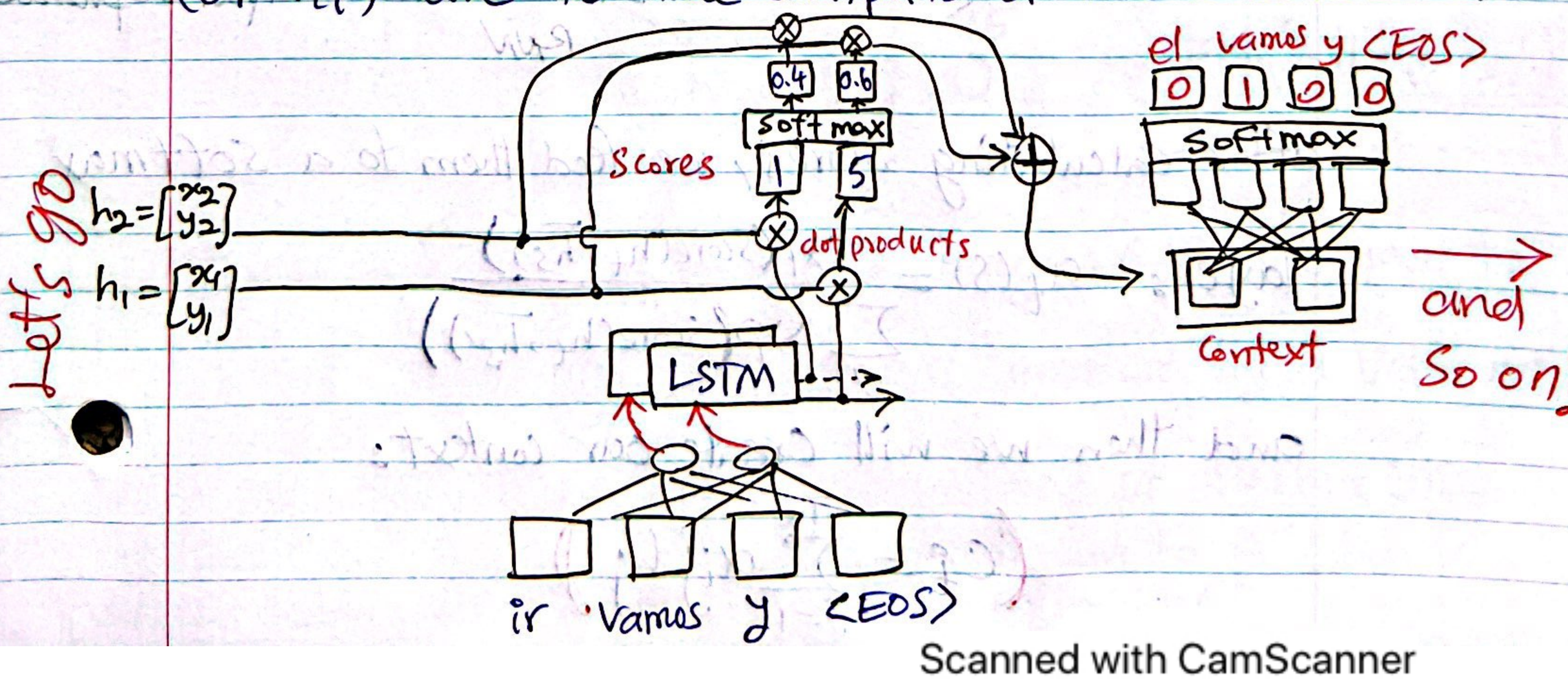
→ In this way, longer inputs have larger contexts!

another benefit is that h_i captures the essense of i^{th} element of the input sequence → we have track of all elements of input and can use them selectively!

✓ In the decoder side, we calculate a weighted sum of h_i to create context for each step → The decoder will be initialized by <EOS> input and generates the first context and then first output → The first output will be used as the input for the second step and generates new scores for h_i and new context! ...

* In a lay language, the thing that attention does is to determine how similar the outputs of Encoder LSTMs (or h_i) are to the outputs of the Decoder LSTM

(cosine loss!) ↔ dot product!!



So, the whole problem is to calculate score. function:

$$score(h_t, \bar{h}_s) = score\left(\square, \square\square\square\right) = \boxed{s_1 | s_2 | s_3}$$

(h_t; h_1 h_2 h_3)

h_t: target (decoder); $\bar{h}_s$: source (encoder)

the length of score vector is equal to the # of h_i in $\bar{h}_s$. a simple way to calculate s_i is dot product of h_t to all h_i. in other words:

$$\boxed{score(h_t, \bar{h}_s) = h_t^T \bar{h}_s}$$

another method is the general method is

$$\boxed{score(h_t, \bar{h}_s) = h_t^T W_a \bar{h}_s}$$

W_a: lxm; h_t^T: mxn; $\bar{h}_s$: nxt

- l: # of hidden units in decoder RNN
- m: # of hidden units in encoder RNN
- n: number of elements in the input sequence

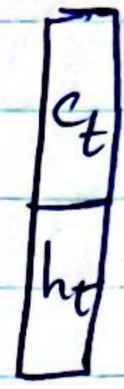
after calculating scores, we feed them to a softmax layer:

$$a_t(s) = \frac{\exp(score(h_t, \bar{h}_s))}{\sum_{s'} \exp(score(h_t, \bar{h}_{s'}))}$$

and then we will create our context:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

- and then we concatenate our current context vector c_t with current hidden state of the decoder h_t: $[c_t; h_t]$ or



context + hidden vector

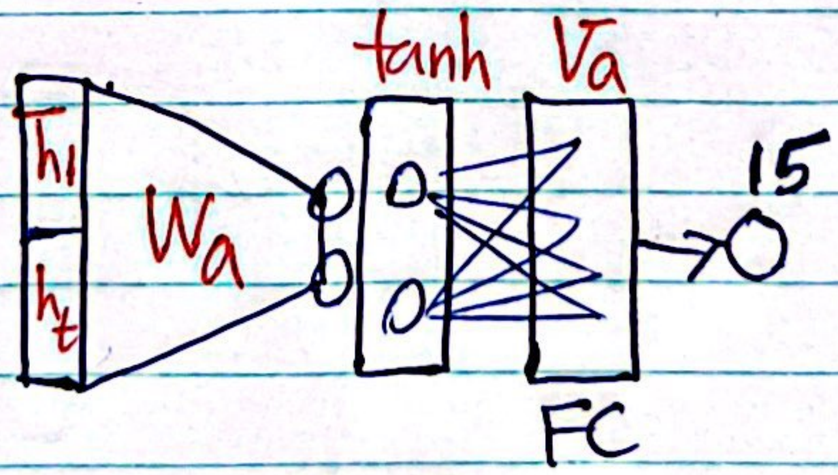
- then we feed this to a fully connected layer with weights W_c followed by tanh to creat $\tilde{h}_t$:

$$\tilde{h}_t = \tanh(W_c[c_t; h_t])$$

- the output of this network ($\tilde{h}_t$) determines the output of our decoder sequence!

- Another approach to calculate score between h_t and $\bar{h}_i$ is concat method! For instance, we want to calculate the score between h_t (e.g., h_4) of decoder and $\bar{h}_1$ in encoder:

$$\text{score}(h_t, \bar{h}_1) = v_a^T \tanh(W_a[h_t, \bar{h}_s])$$



- Computer Vision applications: imag to caption (image to context to text!) The concept is created by a VGG net.

in one layer, it produces feature map of 14x14x512