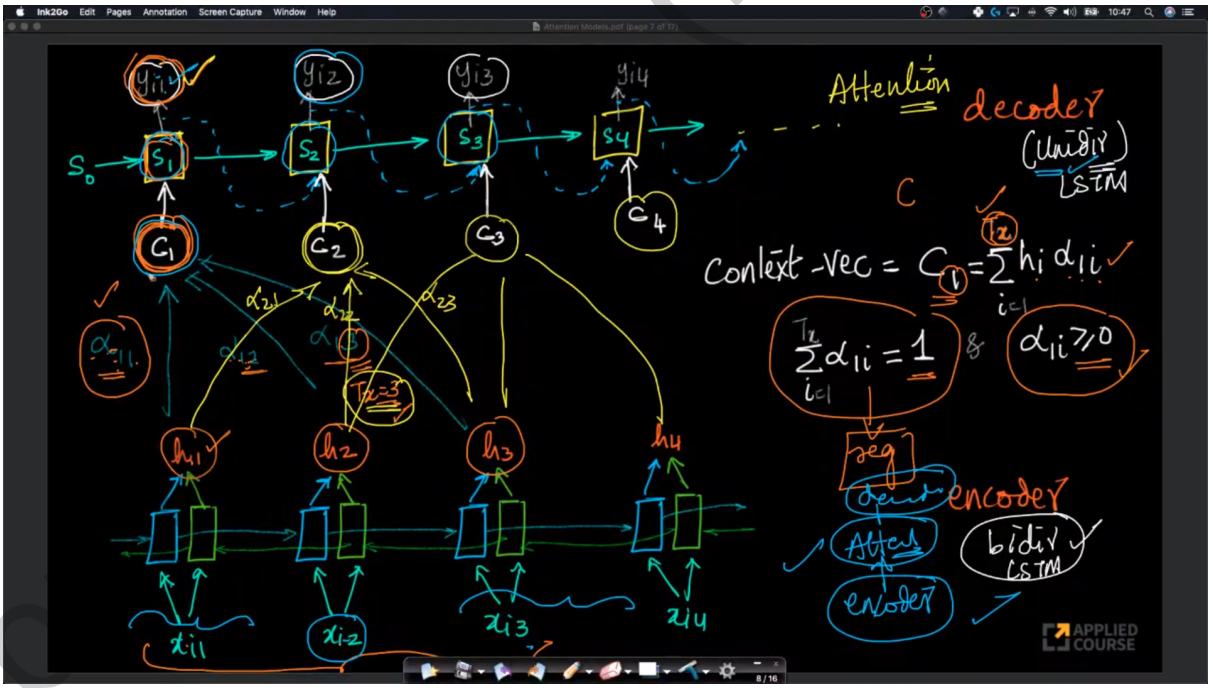
65.1 Attention Models in Deep Learning

Encoder- decoder were popular till 2015,then attention based models were popular followed by BERT.

Simple encoder-decoder architecture does not work for lengthy sentences.
The latent vector w(encoder state vector at last timestep) is meant to give the essence of the input sentence to the decoder. This idea will not work so well when the input is long.

Humans translate by attending to a few words in the input, translate them and comeback to the original sentence then translate them, and this process repeats. This forms the basis for the attention mechanism in encoder-decoder.

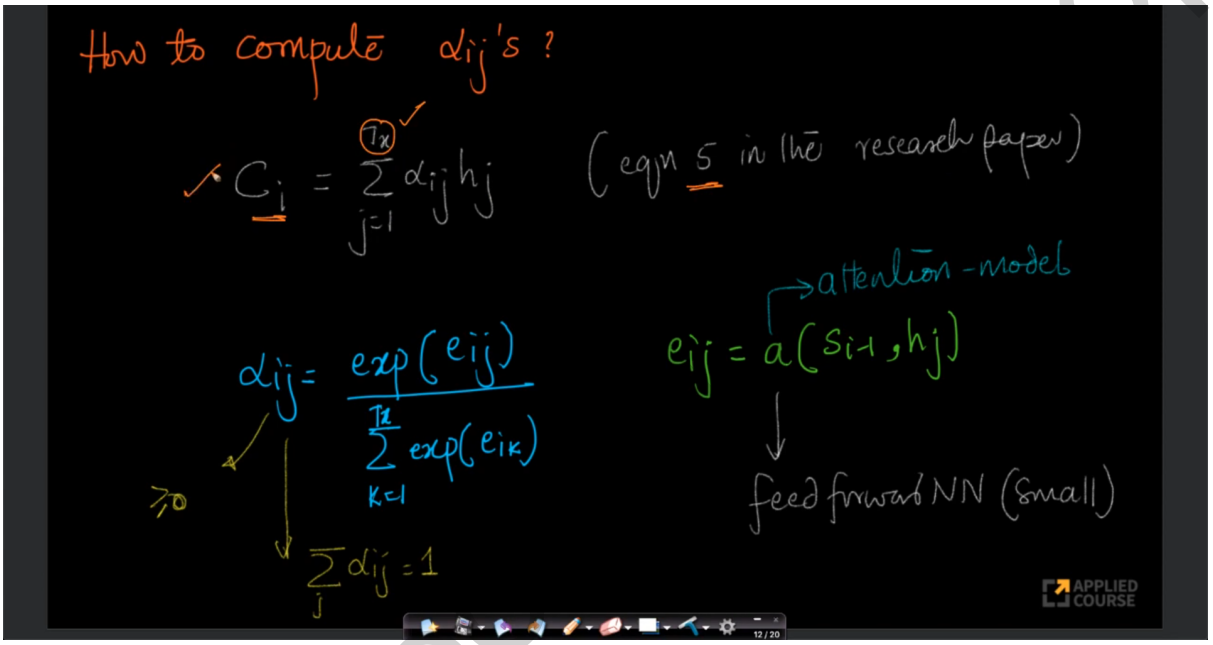


Context vector C here will be a weighted sum of encoder outputs h1,h2,h3,h4. And alpha$_{12}$ will depend on S_0 and h_2.

Decoder is Unidirectional RNN whereas encoder is a Bidirectional RNN. Bidirectional encoder works best in practise, so it is used.

It can also be noticed that encoder,decoder were similar to those of the previous lecture, but the only thing that's changed is the connections between encoder and decoder. So the Attention layer in practice(code) is just an extra layer to add to the encoder-decoder architecture.

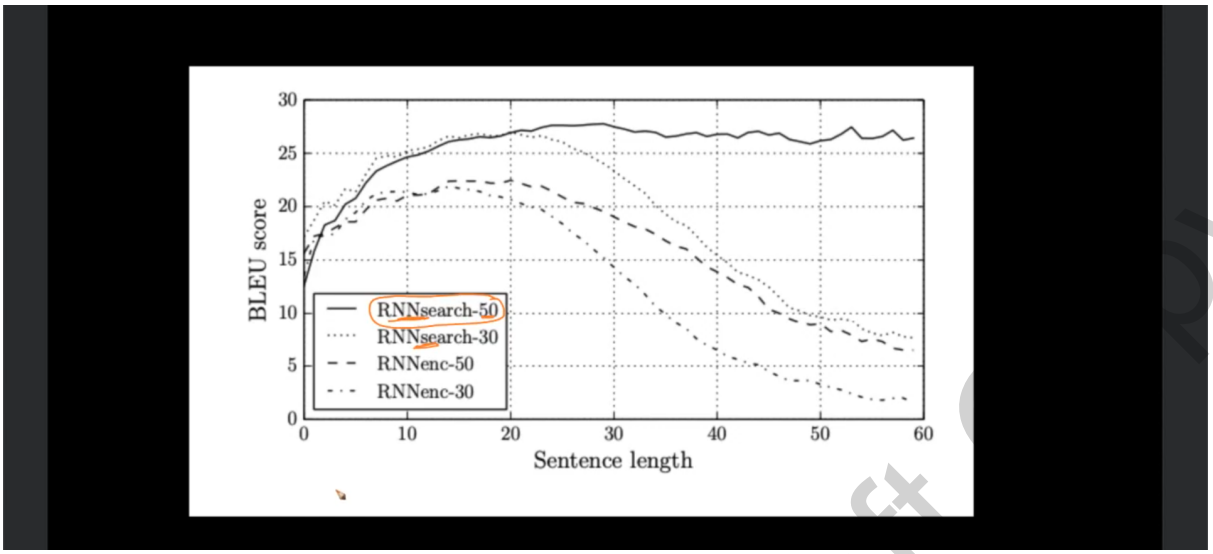
Attention models are also called as models with prior, as we are using the prior knowledge that to translate a word we not only require a particular word but a window of words in the input to encoder.



Here we can find that $alpha_{ij}$ is formulated like a softmax activation on e_{ij}'s. This makes the sum of alpha's for a j to be equal to 1, By this we are getting a weighted sum of encoder outputs as context vector just as discussed above and weights for this sum up to 1, and every weight is positive.

We use a feed forward NN to approximate the function 'a', whose job is to map S_{i-1} and h_j to an e_{ij}.
Although this may look really complex, backprop works in this case as well, as every operation done here is differentiable.

RNNsearch-50 which is the above discussed architecture can be seen to perform well above any other model due to the innovation of attention.

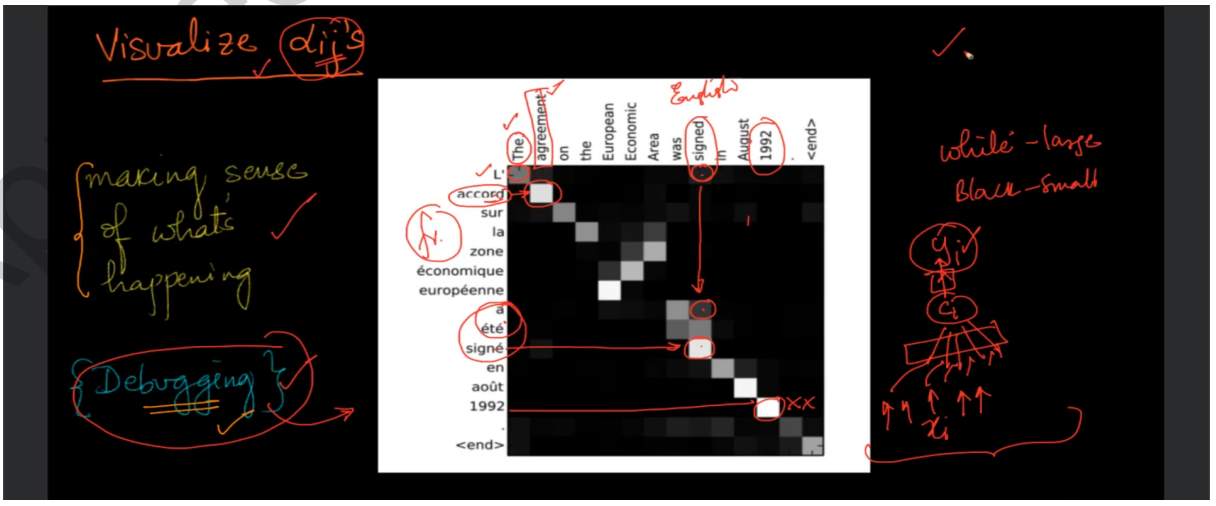


Drawback

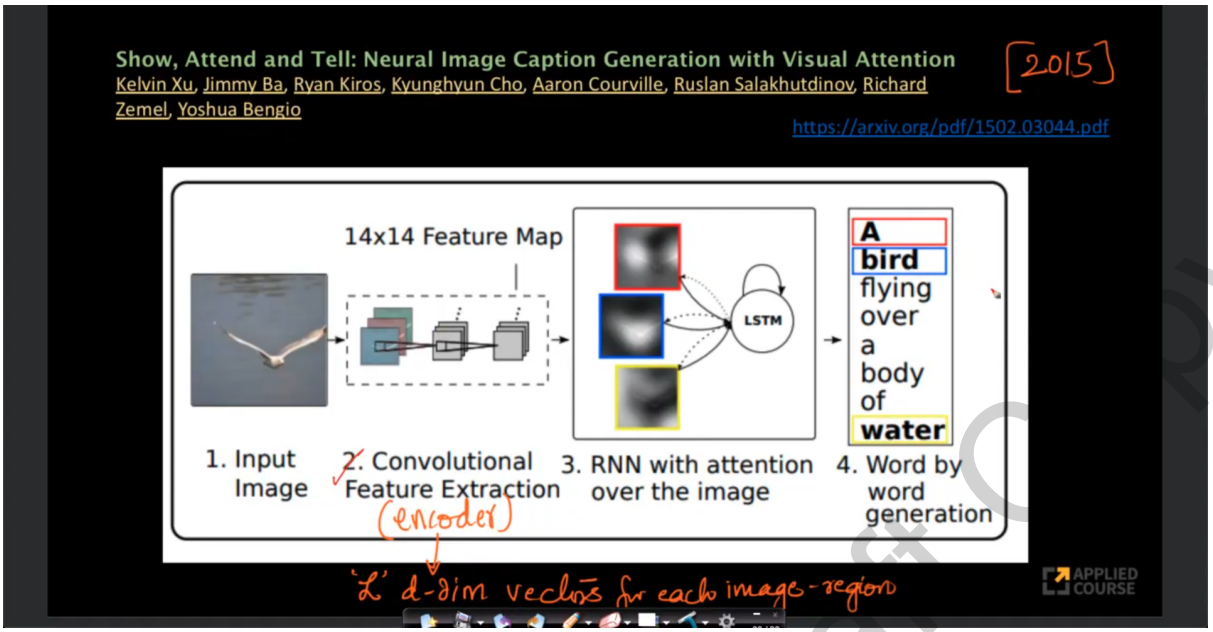
Time Complexity = $O(K_1 * K_2)$
K_1, K_2 are the length of input and output sentences respectively.
This can be explained as, for every decoder output(S) we try to find the weights(alpha's) for every encoder output(h).

Transformers and BERT are based on Attention models.

Attention also helps in debugging the model.Below is a plot which shows what words does the model depend upon when a certain decoder timestep output needs to be given.

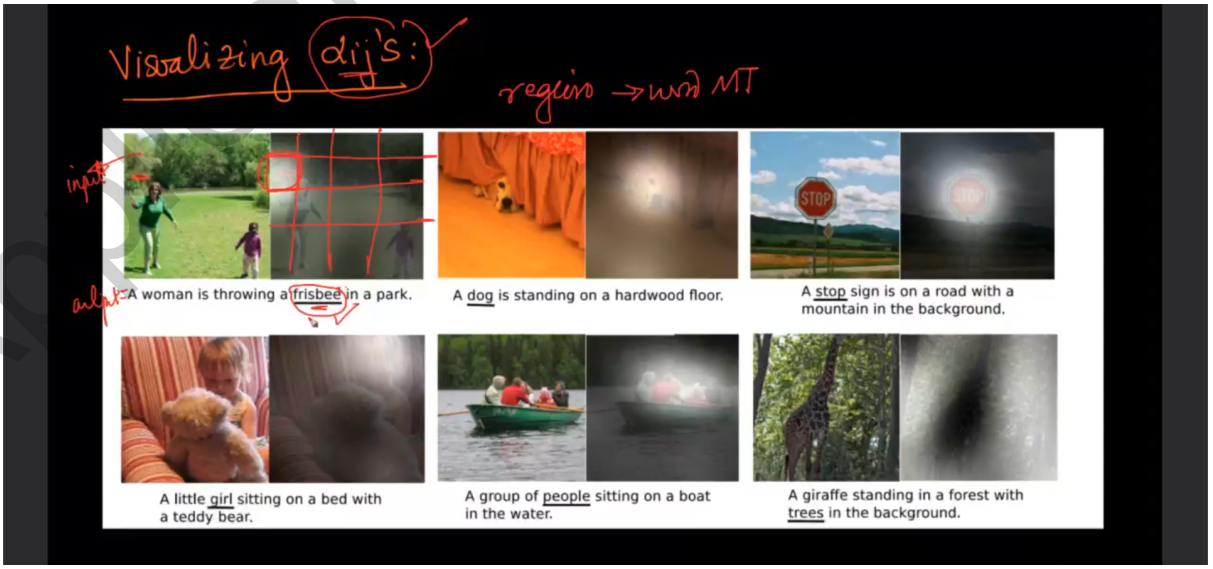


This concept can be extended to **Images** like below.



Here instead of using encoder to generate vectors, we are using a Convnet to generate vectors, and image regions can be thought of as the words in the encoder input. So we are dividing the image into regions and finding features for them from thereon we treat the features like the encoder output(h) and apply attention on them exactly like the way we discussed previously.

Now alpha$_{ij}$ here corresponds to image regions, so a decoder output can be traced back to an image region which was most responsible for the decoder output. This way we can interpret the decisions of the model.



Example Implementation of Attention in tf.keras:

```
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_out, decoder_out])
```

encoder_out, decoder_out are outputs of encoder and decoder in code respectively.