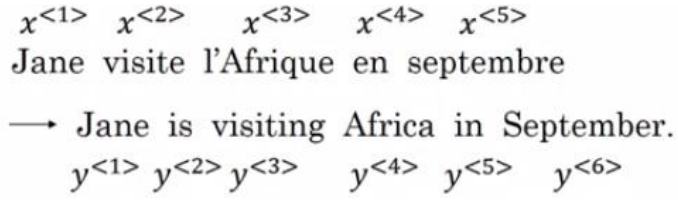
Course 1 Week1

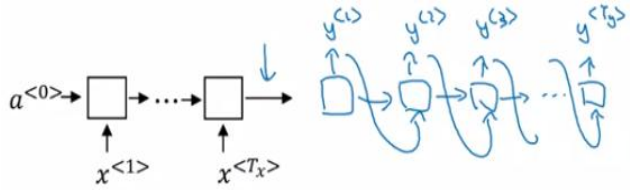
Sequence to Sequence Models:

Basic Model:

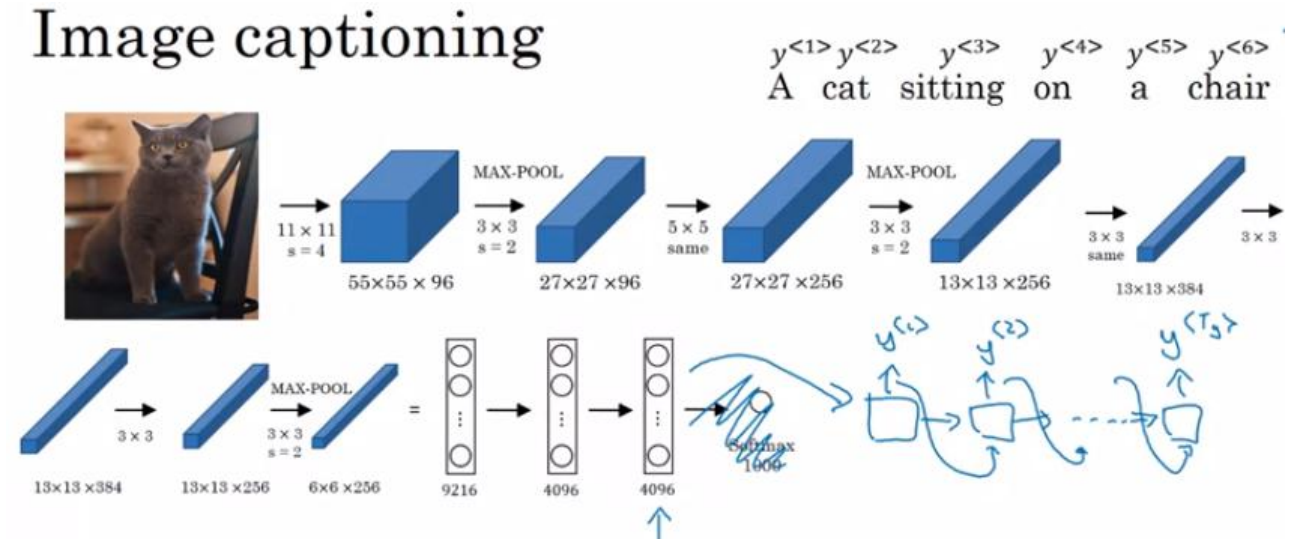
Assume input sequence is a sentence in french and the output sequence is a sentence in English for an example as shwon in figure:



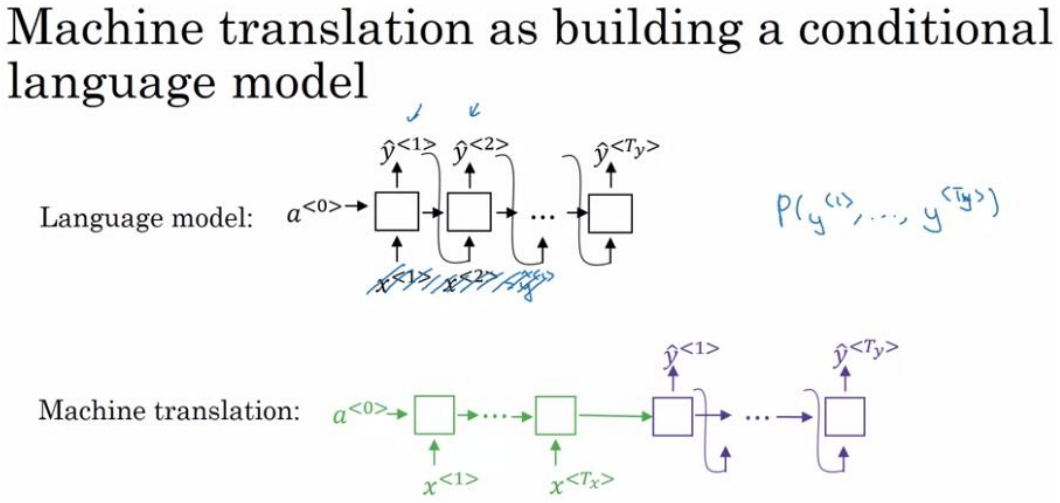
This is an encoding decoding network that the encoder encodes the input sentence and the decoder decodes the output one noting that such networks needs a lot of input output translations to train on them to be able to translate from a language to another.



This example is similar to the Image captioning examples as shown below:



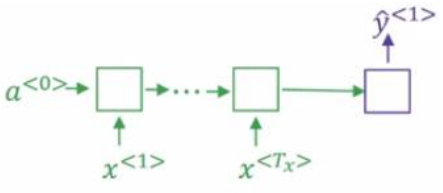
The machine translation is like a conditional language model where $p(y^1 \dots y^{Ty} \mid x^1 \dots x^{Tx})$ as shown:



Beam Search:

Beam search is the most likely algorithm to be used in machine translation. Let us get an example to see how beam search works.

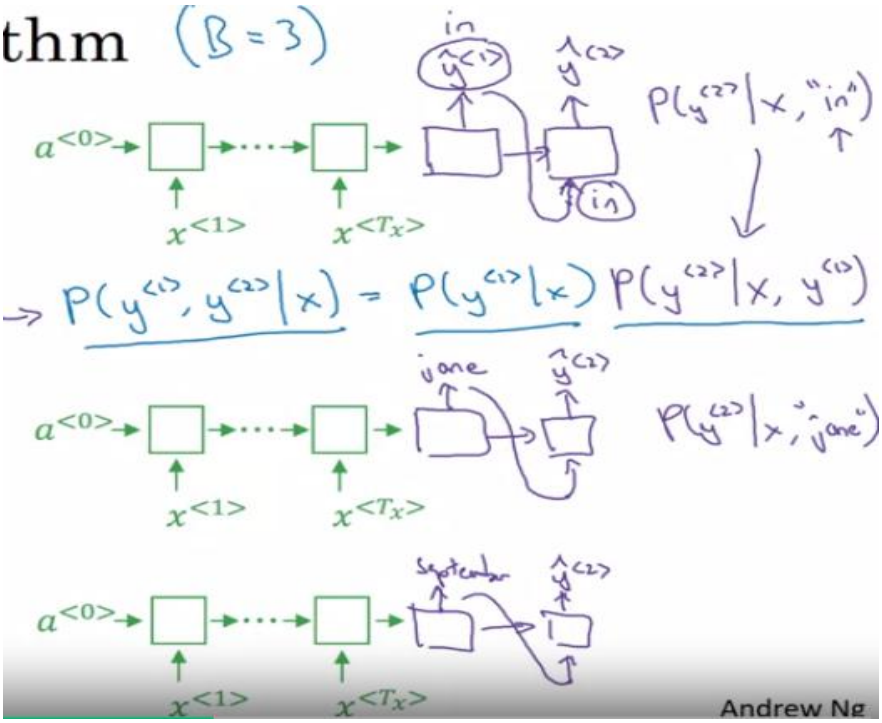
Assuming having English text corpus and you want to translate this text corpus to another language so the first step beam search does is to get the first word in the translated phrase $\hat{y}^{<1>}$ by getting the probability of the first word given the phrase to be translated (English text corpus) $p(\hat{y}^{<1>}| x)$



Beam width: Beam width means that Beam search will cause that not just one possibility but consider three at the time. So for our example given beam width of 3:

Step1: $\hat{y}^{<1>}$ will have the highest probability for 3 values not only looking at the highest one.

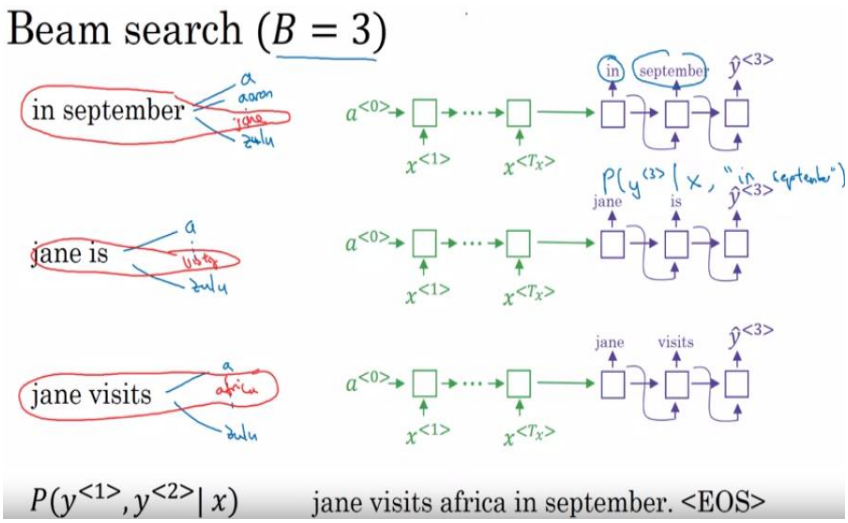
Step2: For each of the 3 given words, consider what should be the second word. This is done by assuming the first word is now known and try to predict the most possible upcoming 3 possiblilties for the second word. $p(\hat{y}^{<1>}, \hat{y}^{<2>}| x) = p(\hat{y}^{<1>}| x) \cdot p(\hat{y}^{<2>}| x, (\hat{y}^{<1>})$



The result will be now 3 candidates of two words that have the highest probability among all the candidates of the possible words after each of the selected 3 words in the first step. Now we iterate again for getting the next words and so on.

The sentence is terminated by <EOS> then the highest probability among the 3 sentences is picked.

Note: B=1 is a greedy algotrithm..



Refinements to Beam Search:

Length normalization is one of the refinements you can do to the beam search algorithm to work better. Beam search is maixmizing this probability shown in the opposite figure.

If we take the summation of logs of same probability it will give us the same maximization results but more stable and less prone to rounding errors. The equation is shown in the opposite figure.

$$\arg\max_{y} \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

Noticing the product form of the probability, we can say that if the sentence was too long knowing that each probability is a number between 0 and 1 then the product of numbers that are less than 1 will give smaller number which makes the objective function tends to give higher accuracies for the shorter sentences rather than the longer one. That is one of the reasons why logarithmic summation is better as the adding of logarithmic of those probabilities between 0 and 1 will give a more negative number.

To overcome the previous problem in a more solid way, we can use normalization by normalizing against the length of the sentence itself. And this significantly reduces the penalty for outputting longer translations.

Note: Normalizing is done by dividing by T_y^{α} where α is a factor that regulates the length such that if α =1 then normalizing is done by whole length of sentence. The default value is 0.7 and can be between 0 and 1.

$$\arg\max_{y} \sum_{y=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

The final objective function now will be as follows:

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

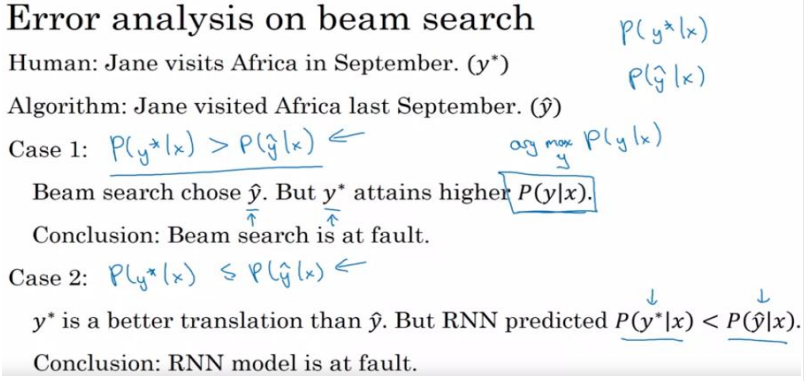
This is called normalized logarithmic likelihood objective function.

How to choose Beam width B ? The larger this number, the more possibilities you're considering, and does the better the sentence you probably find. But the larger B is, the more computationally expensive your algorithm is, because you're also keeping a lot more possibilities around.

Error analysis:

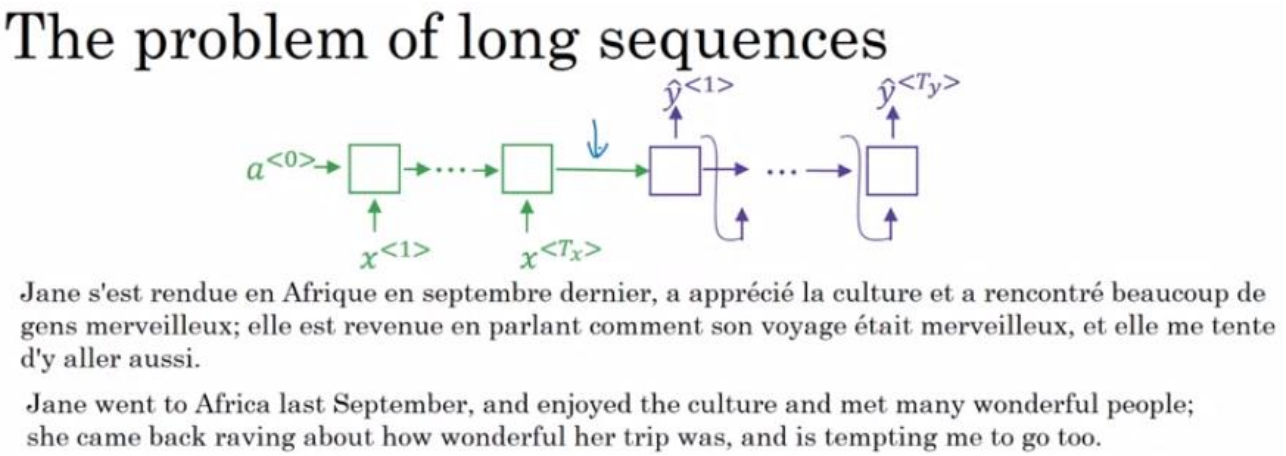
Beam search can make mistakes as it only depends on the n top possibilities and construct its heirarchy of multiplication based on the probabilities.

I didn't understand this part :(



Attention Model:

This a modification on the idea of encoder-decoder network that makes all this work much better. let's take a look at how that works. Get a very long French sentence like this. What we are asking this green encoder in your network to do is, to read in the whole sentence and then memorize the whole sentences and store it in the activations conveyed here. Then for the purple network, the decoder network till then generate the English translation.



What the human translator would do is read the first part of it, maybe generate part of the translation. Look at the second part, generate a few more words, look at a few more words, generate a few more words and so on. You kind of work part by part through the sentence, because it's just really difficult to memorize the whole long sentence like that. What you see for the Encoder Decoder architecture above is that, it works quite well for short sentences, so we might achieve a relatively high score, but for very long sentences, maybe longer than 30 or 40 words, the performance comes down.

The attention models try to mimic how person translates the sentences to be able to cut the long sentences to parts to have short sentences so it doesn't have to memorize.

Implementation:

Let us assume we have the input senetence as shown in the figure and we use bidirectional RNN to compute features on every word. The output sequence is then computed one by one using attention weights such that each word in the input sequence will have an attention weight α with subscript n,m where n represents the position of the word in the output sequence and m represents the position of the word in the input sequence so when we say $\alpha_{1,2}$ for example this means the effect of second word in the input sentence on predicting the first word in the output sentence.

The input to each one of the output sequence will be annotated by c where $c = \sum_{t'} \alpha^{<n,\,t'>} a^{<t'>}$.

But how to compute α? ==>

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$