# Week 3

## Various Sequence to Sequence Architectures

### Basic Models:

Sequence to sequence model:

This is used in translators. For example,

French:

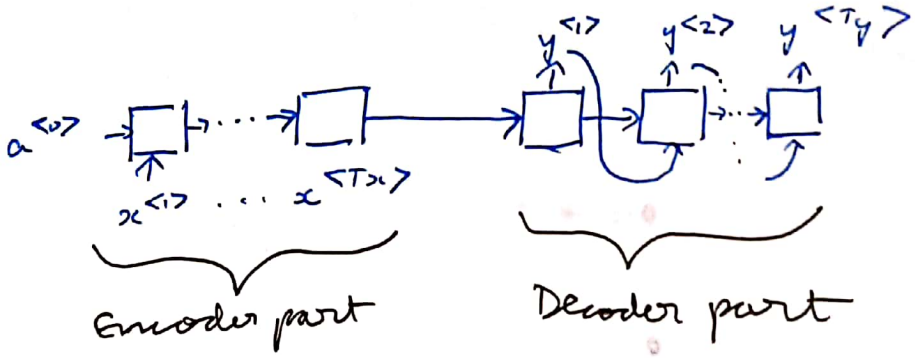
Jane visite l'Afrique en septembre

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$

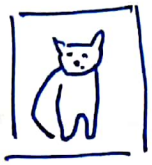
English:

Jane is visiting Africa in September

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$



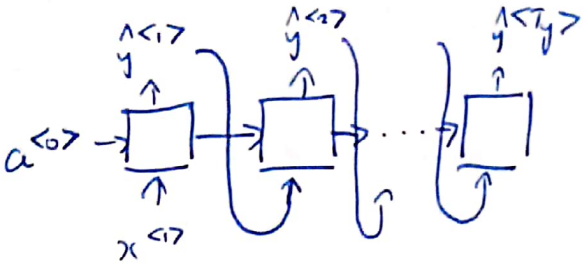

You can even use a similar method for image captioning:

 → A cat sitting on a chair

We do this by replacing the encoder part with an AlexNet (the final softmax layer is removed and is instead connected to the decoder part)

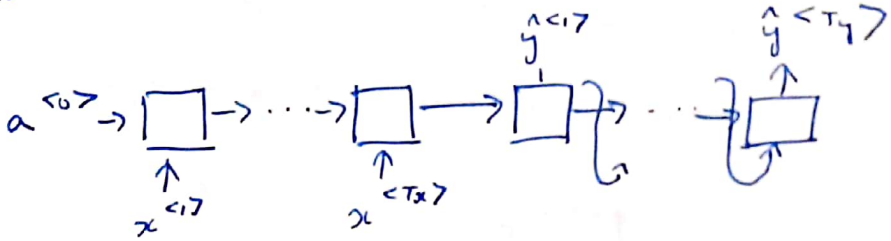
# Picking the most likely sentence:

Language model (that we learnt in week 2):



$P(y^{<1>}, \ldots, y^{<T_y>})$

Machine translation:



→ Similar to language model, just that rather than inputting a $a^{<0>}$ in the decoder, in machine translation we input the output of the encoder.

→ $P(y^{<1>}, \ldots, y^{<T_y>} \mid x^{<1>}, \ldots, x^{<T_x>})$

Instead of modelling any sentence (like in language model), it is now modelling the probability of the output English translation, conditions on some input French sentence.

→ In language model, we would select the next word at random (from the list of possible word predictions), but here we pick the best.

$$\arg\max_{y^{<1>}, \ldots, y^{<T_y>}} P(y^{<1>}, \ldots, y^{<T_y>} \mid x)$$

→ Why don't we use greedy search algorithm?

Greedy Search algorithm is a CS algorithm that finds the 1st best word, 2nd best word, 3rd best word and so on. By best I mean $P(\hat{y}^{<1>}, \hat{y}^{<2>}, \ldots, \hat{y}^{<T_y>} | x)$ is the highest. But consider the following:

- Jane is visiting Africa in September
- Jane is going to be visiting Africa in September

Here $P(\text{Jane is going} | x) > P(\text{Jane is visiting} | x)$ So it'll pick 2 rather than 1, even though 1 is better (since going is a more probable word than visiting, it'll pick that). So this doesn't work.

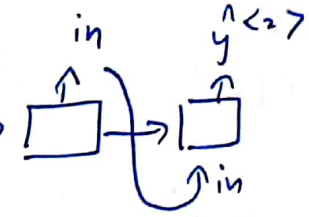
Beam Search Algorithm:

Here we use a parameter B.

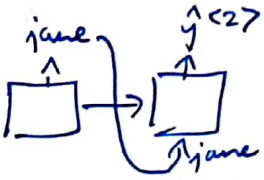
If $B = 3$, rather than picking the best word like in greedy search, we pick the top 3 words. Then we create 3 parallel networks where we hardcode and feed these 3 words seperately for the second cell of the decoder RNN. The output of the 2nd cell will be 10K words for each RNN (So total $3 \times 10K = 30K$). Out of 30K we pick the best 3 two words (1st word + 2nd word), and this goes on.

Step 1: choose 3 best

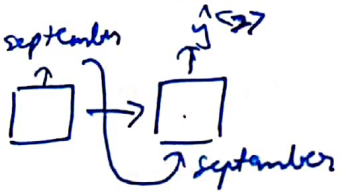

1) Encoder → $P(y^{<2>} | x, \text{"in"})$



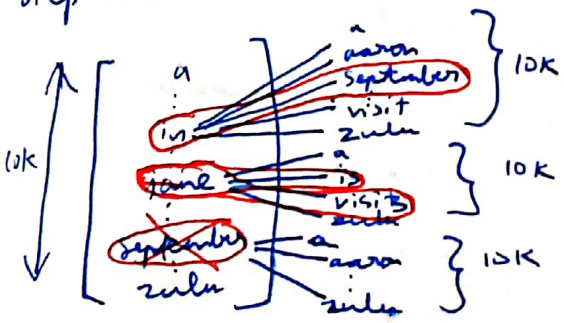

2) Encoder → $P(y^{<2>} | x, \text{"jane"})$




3) Encoder → $P(y^{<2>} | x, \text{"september"})$




Step 2:




Top 3 ~~$P(y^{<1>}, y^{<2>} | x)$~~

$P(y^{<1>}, y^{<2>} | x)$

So next we find $P(y^{<3>} | x, \text{"in september"})$

$P(y^{<3>} | x, \text{"jane is"})$

$P(y^{<3>} | x, \text{"jane visits"})$

Notice how "september ___" wasn't in the top 3 so it was relegated.

If we do this for further words we will get:

jane visits africa in september. <EOS> ✓

Note: $B = 1$ is greedy search algorithm.

## Refinements to beam search:

Length normalization:

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

This is nothing but $P(y^{(1)} \dots y^{<T_y>} | x) =$

$$P(y^{(1)}|x)\, P(y^{(2)}|x, y^{(1)}) \dots P(y^{<T_y>} | x, y^{(1)} \dots y^{(T_y-1)})$$

~~But when we~~

~~Here number hit~~

-> These terms like $P(y^{(1)}|x)$ will be between 0 and 1, so what happens when we multiply many of these is that we get a very small number that our floating value can't hold (because of less precision). So instead we use this formula:

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

-> But since the more words we have, the lesser the probability becomes, this also tends to choose shorter sentences. So we normalise it.

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

where $\alpha = 0.7$ usually

Since we are like taking the average, it won't penalise longer sentences.

before we discussed that we will choose the 3 best translations (for B = 3). we will score them using this formula and choose the best as final output.

How to choose Beam Width B?

→ large B: better results, slower
small B: worse results, faster

considered big for this application

→ In production systems you would use 10 - 100
In research, where best result is required, 1000 - 3000

→ The gain in results from 1→10 will be larger than 1000→3000 (higher the number, the less performance improvements we will see).

→ Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$

# Error Analysis in Beam Search:

Beam search is an approximation algo, so it might not give the most accurate result always. If an error occurs, how do you find out if the problem is with beam search or your RNN?

Example:

Jane visite l'Afrique en septembre

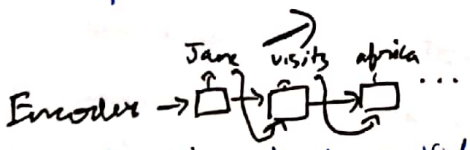
Human: Jane visits Africa in September. ($y^*$)

Algorithm: Jane visited Africa last September. ($\hat{y}$)

Was the RNN or the beam search algo to blame for this bad translation?

You can increase the beam width, so it gets better results, but this is like getting more data - its not the most effective way. There is a better way.

1. Compute $P(y^*|x)$ and $P(\hat{y}|x)$




By plugging in these values.

2. Case 1: $P(y^*|x) > P(\hat{y}|x)$

Beam search chose $\hat{y}$. But $y^*$ attains higher $P(y|x)$

Conclusion: Beam search is at fault

↑

It's job was to find the highest $P(y|x)$ and it failed at that by choosing $\hat{y}$ over $y^*$

Case 2: $P(y^*|x) \leq P(\hat{y}|x)$

$y^*$ is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) < P(\hat{y}|x)$

Conclusion: RNN model is at fault

Error analysis Boxes:

For all the translations it got wrong,

| Human | Algorithm | $P(y^*\|x)$ | $P(\hat{y}\|x)$ | At fault? |
|---|---|---|---|---|
| Jane visits Africa in September | Jane visited Africa last September | $2 \times 10^{-10}$ | $1 \times 10^{-10}$ | B |
| – | – | – | – | R |
| – | – | – | – | B |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |

Find the fraction of errors due to beam search vs RNN model.

-> Only if a majority of the errors are due to beam search, then only then its a good idea to increase beam width B.

-> If a majority of the errors are due to RNN, you can perform a deeper analysis on whether to add regularization, more complex network, etc.

## Bleu score:

One of the challenges of machine translation is that given a French sentence there can be multiple equally good english translations. The BLEU score is a measure of accuracy to sort this.

Bleu -> Bilingual Evaluation Understudy

Take an example,

French: Le chat est sur le tapis

~~And it has 2 equally good translations~~

Reference 1: The cat is on the mat

Reference 2: There is a cat on the mat

These are two references given by a human

The intuition is that we are going to look at the machine generated output and see if the types of words it generates appear in atleast one of the human generated references.

reactive Translation

MT output: the the the the the the the

Precision: $\frac{7}{7} \leq$ no of words in MT output that appear in either ref1 or ref2

$7 \leq$ no. of words in MT output

So this isn't a good way of measuring precision, so we use a modified precision

Modified Precision: $\frac{2}{7}$ ← no. of times the word appears in a reference (Count clip ("the"))

The appears 2 times in ref 1 and 1 time in ref 2

2 ? 1 ∴ It is 2

← no. of ~~words in MT output~~ time the word appear in MT output

Count ("the")

Bleu score on bigrams:

Example,

Reference 1: The cat is on the mat

Reference 2: There is a cat on the mat

MT output: The cat the cat on the mat

| | Count | Count clip |
|---|---|---|
| the cat | 2 | 1 |
| cat the | 1 | 0 |
| cat on | 1 | 1 |
| on the | 1 | 1 |
| the mat | 1 | 1 |
| | 6 | 4 |

∴ Score = $\frac{4}{6}$

Formulas:

For unigram,

$$P_1 = \frac{\sum_{\text{unigrams} \in \hat{y}} \text{Count}_{clip}(\text{unigram})}{\sum_{\text{unigrams} \in \hat{y}} \text{Count}(\text{unigram})}$$

$P_1$ → 1 means unigram

$\hat{y}$ is MT output

For n-gram,

$$P_n = \frac{\sum_{\text{n-grams} \in \hat{y}} \text{Count}_{clip}(\text{n-gram})}{\sum_{\text{n-gram} \in \hat{y}} \text{Count}(\text{n-gram})}$$

Bleu details:

$P_n$ = Bleu score on n-gram only

Suppose we get $P_1$, $P_2$, $P_3$, $P_4$

The the combined Bleu Score is,

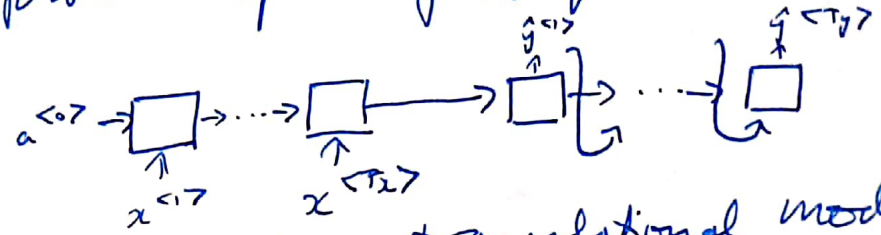
$$BP \exp\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)$$

Here BP is Brevity Penalty. If you output short translations, it's easy to get high precision. But we don't want translations that are very short, so BP is an adjustment factor that penalizes translation systems that output short translations.

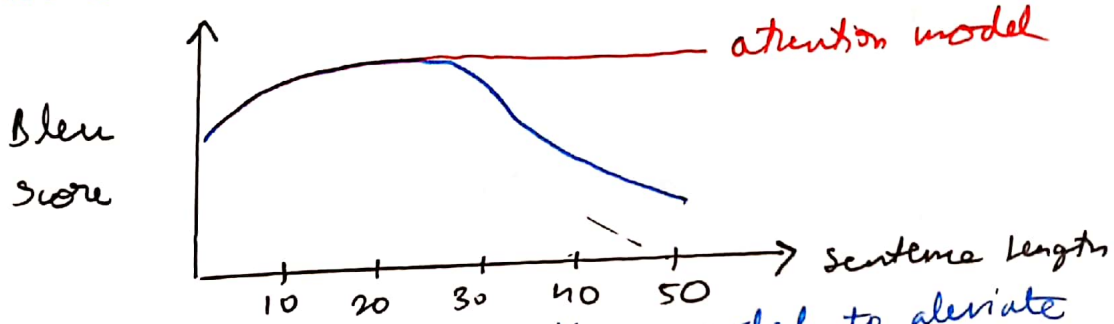
$$BP = \begin{cases} 1 & \text{if MT output length} > \text{reference output length} \\ \exp(1 - \text{reference output length} / \text{MT output length}) & \text{otherwise} \end{cases}$$

# Attention Model:

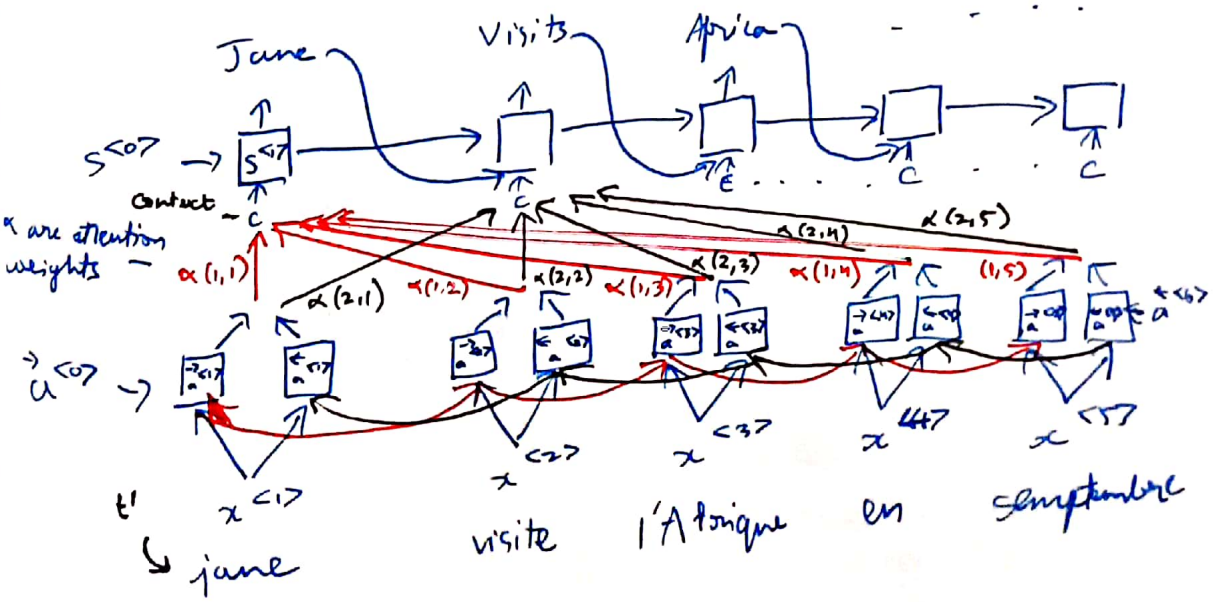
## The problem of long sequences:



In the RNN machine translational model, it memorises the entire large sentence and translates it in one shot. However a real human translator looks at a few words and translates these smaller parts one at a time. Therefore the performance of the encoder-decoder model drops as word count increases.




So we use an attention model to alleviate this.



→ $a^{\langle t' \rangle} = \left( \overrightarrow{a}^{\langle t' \rangle}, \overleftarrow{a}^{\langle t' \rangle} \right)$

→ $\sum_{t'} \alpha^{\langle 1, t' \rangle} = 1$

→ $c^{\langle 1 \rangle} = \sum_{t'} \alpha^{\langle 1, t' \rangle} a^{\langle t' \rangle}$

→ $\alpha^{\langle t, t' \rangle}$ = amount of attention $y^{\langle t \rangle}$ should pay to $a^{\langle t' \rangle}$

$$= \frac{\exp(e^{\langle t, t' \rangle})}{\sum_{t'=1}^{Tx} \exp(e^{\langle t, t' \rangle})}$$

→ To get $e^{\langle t, t' \rangle}$:

$s^{\langle t-1 \rangle}$ →

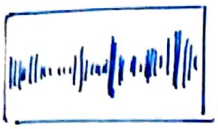
$a^{\langle t' \rangle}$ → [0 0 0 0] → $e^{\langle t, t' \rangle}$

# Speech Recognition - Audio Data

## Speech Recognition:

The ~~problem~~ speech recognition problem is:

x (audio clip) → y (transcript)



"the quick brown fox"

→ A common pre-processing step for audio data is to convert the raw audio clip and generate a spectrogram (plotting the frequencies).

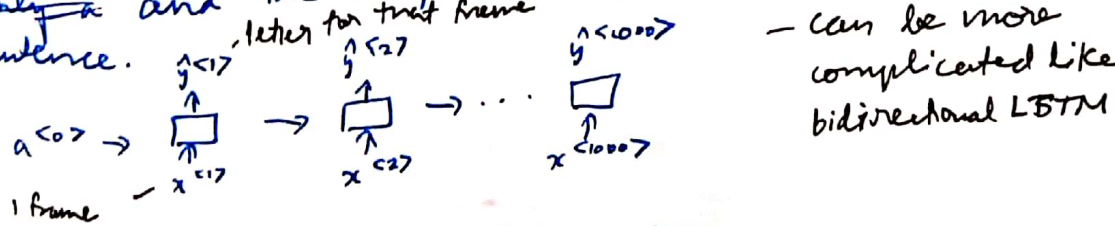
→ Before speech recognition systems used to be built using phonemes (like the is de, quick is k-wic, etc - written as sound) but due to deep NN, this is no longer required.

→ 300h, 3000h (for research), 100,000h (commercial) of training data have been used.

CTC (Connectionist temporal classification)

For speech recognition:

Usually we will have many inputs (since a 1 hr audio clip for example is split into 100 Hz frames) and the outputs we get collapse to form a sentence.




- can be more complicated like bidirectional LSTM

We get ttt_h_eee ___ space ___ qqq ___

Collapses to the_ q

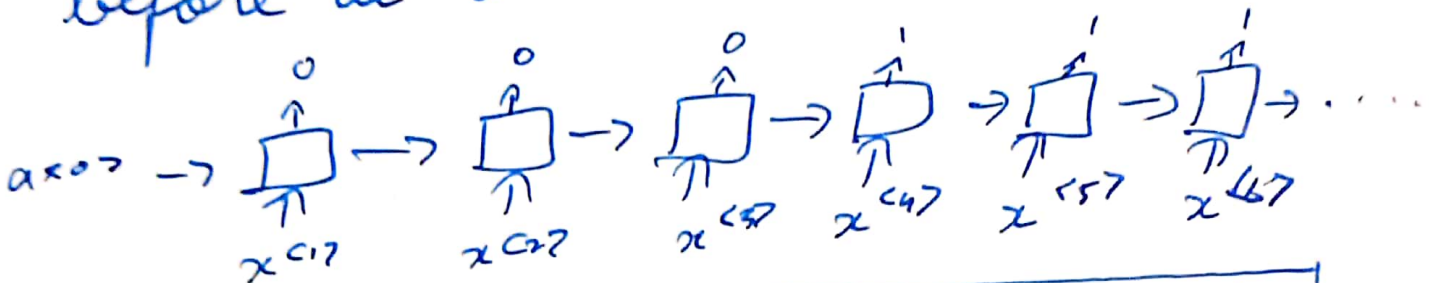
The letters that aren't divided by a _ are combined (eg ttt → t)

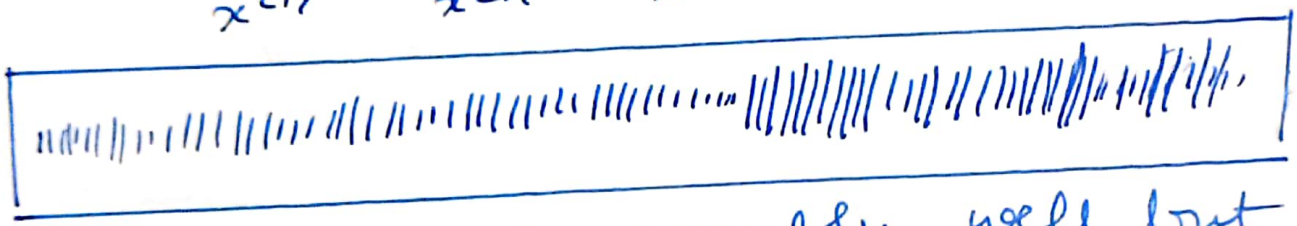
# Trigger Word Detection :

Examples:

Amazon Echo woken up with Alexa

→ When the trigger word is said, mark that frame with a 1, while the ones before as 0



This works reasonably well but it results in a very imbalanced training set where most examples are 0.

Hack: Instead of setting only the example with the trigger word, make subsequent 3-4 examples also 1.