Lecture 7: Seq2Seq, NMT, Attention

Andrei Arsene Simion

Columbia University

March 8, 2023

Outline

- NMT
- Attention

Outline

- NMT: Neural Machine Translation
- Attention

NMT

- Sometimes we want to go from one input sequence to another output sequence
- These are known as Seq2Seq problems
- Some examples:
  - Machine Translation: Go from one language to another language
  - Summarization: Go from a large piece of text to a smaller piece of text
  - Chatbots: Generate a piece of text which is an answer to some input question
  - Code Generation: Words to Python code
  - Named Entity Recognition: Sentence to Tags
  - Parsing: Sentence to output parse
- We'll focus on Translation here

Neural MT vs Statistical MT

- It would be good to know how this was done before by looking at SMT = Statistical Machine Translation
- Suppose we want to translate from x (French) to y (English)
- Let x have length T_x and y have length T_y
- From 1990 - 2010, people focused on modeling $p(y|x)$
- If you look at this a bit, you'll notice that (Bayes' Rule)

$$argmax_y p(y|x) = argmax_y p(x|y)p(y)$$

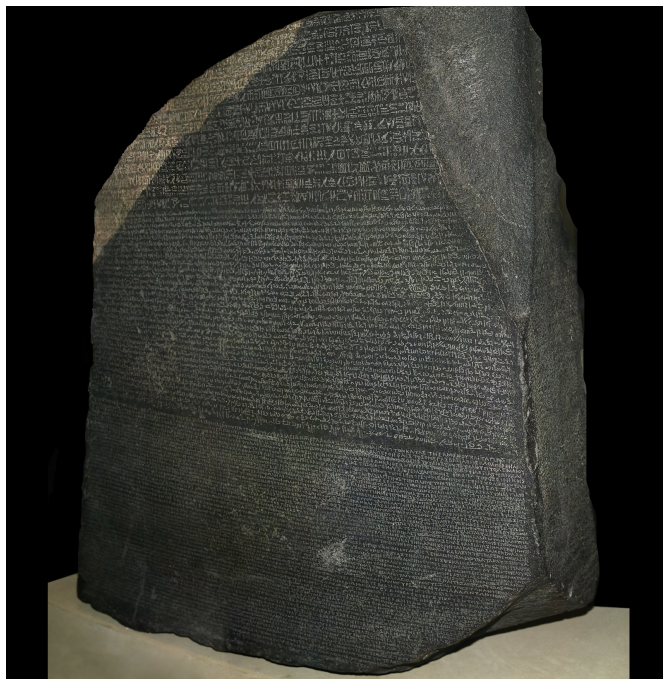
- This is 2 different problems
- Problem 1: Model $p(x|y)$ (Translation)
- Problem 2: Model $p(y)$ (Language Model)

SMT

- Problem 2: Model $p(y)$ (Language Model)
  - Learning $p(y)$ can be done via for example n gram models or otherwise - we saw this
  - This model deals with good fluency; how do we write proper English?
- Problem 1: Model $p(x|y)$ (Translation)
  - Learning $p(x|y)$ can be done if we have data (x, y) which we know has y mapping to x
  - This model deals with good fidelity: how should words and phrases be translated?
  - This model cares more about getting the right words across, not making sense
- At a high level, if we solve both Problem 1 and Problem 2 well, the y that we pick needs to be decent both for $p(x|y)$ and $p(y)$, so it should sense and contain the right words

SMT Data

- We have parallel data from many sources
- For example, human translations (Books, TV, Court Proceedings)
- One of the oldest examples is the Rosetta Stone



Using Data for SMT

- How do we use this data?
- In SMT, what people typically did was they considered models that had structure on $p(x, a|y)$, where

$$p(x|y) = \sum_{a}(p(x, a|y))$$

- Here, "a" was usually labeled "*alignment*" and it was a latent variable that told us for each target (x) word which source (y) word generated it
- Put another way: an alignment is, for each word in x, a word in y that was used to generate
  - Note: some models handled the case when for a certain word x_t there was no word y_{a_t} which generated it

SMT

- Some examples; note that in the third example we are aligning not words but phrases

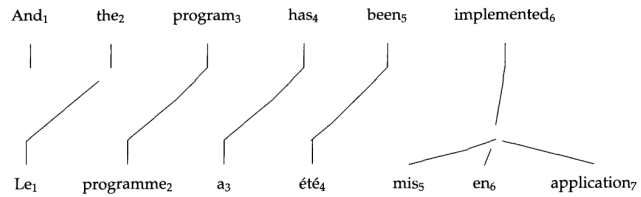


Figure 1
An alignment with independent English words.

An alignment with independent English words.

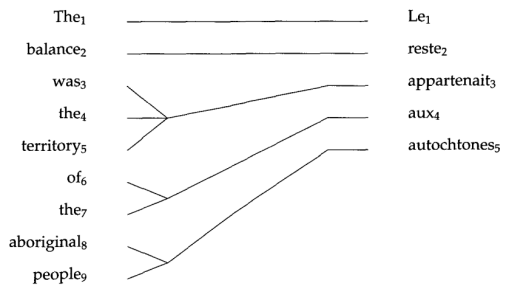


Figure 2
An alignment with independent French words.

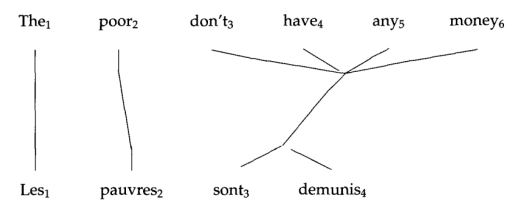


Figure 3
A general alignment.

NMT vs SMT

- There were many SMT models but the most famous ones were called the IBM word-based models as they were built by researchers working at IBM
- For example, a famous model called IBM 2 said that

$$p(x, a|y) = \prod_{t=1}^{T_x} q_w(x_t|y_{a_t}) q_d(a_t|t)$$

- In the above, $a_t \in \{0, 1, \ldots, T_y\}$, q_w and q_d are probabilities
  - $a_t = 0$ meant that a "null" word generated x_t

SMT

- For such a model like IBM 2, you'd want to estimate q_w and q_d which handle the probability of a word generating another word and a position generating another position
- You'd write out the likelihood over data $\{(x_i, y_i)\}_{i=1}^N$ like

$$-\sum_{i=1}^{N} \log p(x_i|y_i)$$

  and estimate q_w and q_d via Expectation Maximization
- Once you have these parameters estimated, you can recover the alignments via

$$a = argmax_{a_1,\ldots,a_{T_x}} \prod_{t=1}^{T_x} q_w(x_t|y_{a_t}) q_d(a_t|t)$$

SMT

- Once you get the alignment from your model, you can measure its quality vs some human generated alignment and recover a metric like Alignment Error Rate (AER)
- AER was loosely negatively correlated with translation quality, so people searched for models with better and better AER
- These models were quite complicated
- Note: to get $argmax_y p(x|y)p(y)$ you can use Beam Search, or another heuristic; Beam Search is a general decoding method

SMT

- SMT was a huge research field
- The best systems were extremely complex
  - There were lots of important details that we skip
  - Systems had many separately designed subcomponents
  - Lots of feature engineering
    - You needed to do particular feature design to capture various language effects
  - SMT required compiling and maintaining extra resources
    - For example, we needed to keep tables of equivalent phrases
  - Lots of human effort to maintain
    - Lots of repeated effort for each language pair!

NMT

- What is Neural Machine Translation?
- At a high level, NMT is doing machine translation but using a single network

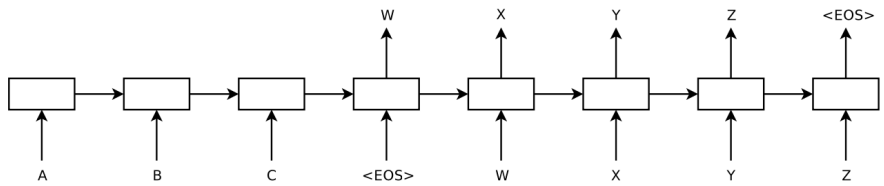


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

- Let's spell out the picture above a bit more
- Here, x is the input, y is the output; T_x, T_y are the number of tokens in each
- NMT was a small effort in 2014 - it blew up!

NMT

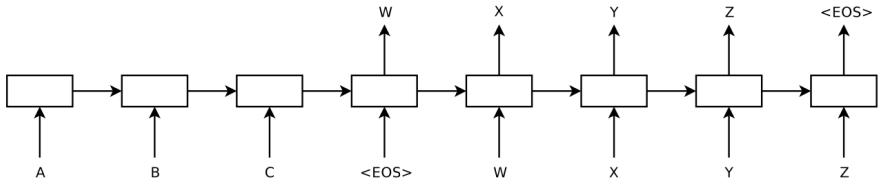


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

- 2 Models: An encoder and a decoder
  - Encoder and decoder were usually 2 separate RNN models (no parameter sharing)
  - Encoder RNN can have a different structure than the decoder RNN (encoder = LSTM, decoder = special GRU (?))
  - The representation of the input would usually be the final encoder hidden states h_{T_x}
  - $h_{T_x} = k_0$, the final encoder hidden layer states are the initial decoder hidden layer states

NMT

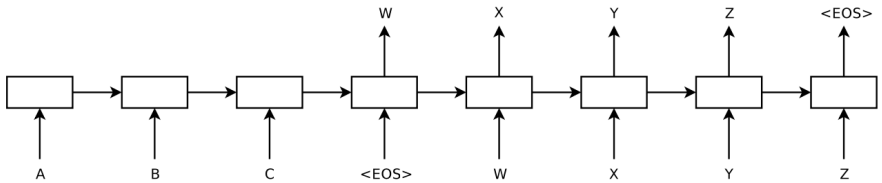


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

- Note also that NMT directly learns a Conditional Language Model
- Specifically, we have
  $p(y|x) = p(y_1|x)p(y_2|y_1, x) \dots p(y_{T_y}|y_{T_y-1}, \dots, y_{T_1}, x)$
- No need to have two different models and components!
- How do we train this system?

NMT: Loss

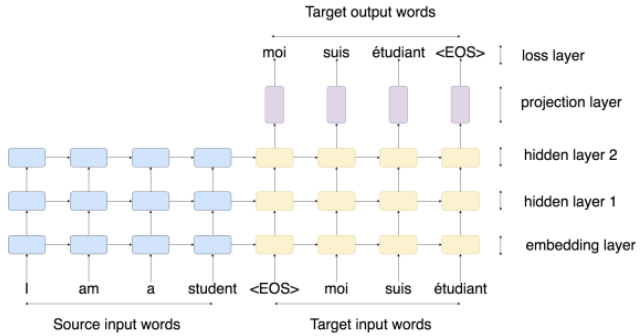


Fig. 1. The training process of RNN based NMT. The symbol $< EOS >$ means end of sequence. The embedding layer is for pre-processing. The two RNN layers are used to represent the sequence.

- The input is fed in and we get the final hidden values per layer
- These final hidden values initialize the corresponding hidden layers in the decoder
- We train by Teacher Forcing on the decoder side: we aim to predict $(y_1, y_2, \ldots, y_{T_y+1})$ from $(y_0, y_2, \ldots, y_{T_y})$
- Note $y_0 =< s >$ and $y_{T_y+1} =< /s >$ are the special start of sentence and end of sentence tokens
- Specifically, it is like we have a language model to train but we initialize it with a special set of values (h_{T_x})

NMT: Layers

- To get the best performance, we can use a (bidirectional) stacked LSTM for the encoder
- The decoder RNN is autoregressive: it can't look into the future (it's a sort of language model!)
- We can use a stacked (one directional) LSTM for the decoder
- The idea is that lower layers compute general features, while higher layers compute more complex interactions: both are crucial
- Seems like 2-4 layers is about the sweet spot in the number of layers needed
  - 2 layers better than 1
  - 3 layers a little better than 2
  - The more layers you have, the more you'll need connections, tricks, etc

NMT vs SMT - Advantages

- Compared to SMT, NMT has many advantages
- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities
- 1 Neural Network, not 12 models in a disjoint "pipeline"
- Much less human engineering effort

NMT vs SMT - Disadvantages

- Disadvantages of NMT?
  - NMT is less interpretable and hard to debug
  - NMT is difficult to control: we can't really put in rules and and get the result we want

BLEU - How do we say a translation is "good"?

- How do we measure the quality of a translation?
  - Source: Le chat est sur le tapis
  - MT System Output: the the the the the the the
  - Reference 1: The cat is on the mat
  - Reference 2: There is a cat on the mat
- How do we know this is a bad translation?

BLEU

- One idea is to measure the precision of the translation
- For a general MT output we can get the precision by
  - Getting all the words (tokens) in the generated translation
  - Counting if each word is or is not in the reference translations
  - Dividing by the number of words in the generated translation
- For the example above we have 1 word 7 times
- "the" is seen in at least one of the reference sentences
- We get Precision $= \frac{7}{7}$
- But this is bad, we clearly have a bad translation

BLEU

- One possible idea is to clip the maximum count for each token in the generated output by the maximum count in any of the references
- For example, the count of "the" can't be more than 2 in either of the reference sentences, so we'll clip this word's count
- Specifically we define Precision $= \frac{\sum_{w \in \hat{y}} Count_{clip}(w)}{\sum_{w \in \hat{y}} Count(w)}$

BLEU

- Specifically, this is
  - For each unique word w in the generated translation
  - Get $Count(w)$, the number of times the word w is in the generated output
  - Get $Count_{clip}(w)$, the minimum of $Count(w)$ and the maximum number of times w is in some reference sentence
  - Return $\frac{\sum_{w \in \hat{y}} Count_{clip}(w)}{\sum_{w \in \hat{y}} Count(w)}$
- With this definition, we get a Precision of 2/7 since "the" is in the generated translation 7 times and in the first reference 2 times

BLEU

- So, clipping solved one problem
- But, what if we have
  - Translation 1: The cat is on the mat
  - Translation 2 (Yoda): Cat on the mat it is
- Clearly, Translation 1 is better but we would not pick this up if we just use tokens w that are unigrams (words)
- To this end, we can use bigrams, trigrams, and 4-grams to capture word order
- For example, for Translation 1, we have bigrams "the cat", "cat is", "is on", "on the", "the mat"
- We can then do the same exact thing as before, but this time use w as a bigram instead

BLEU

- The precisions we calculate for 1, 2, 3 and 4 grams are p_i
- Then, BLEU (Bilingual Evaluation Understudy) is

$$\text{BLEU} = (p_1 p_2 p_3 p_4)^{1/4}$$

- Another problem: short translations can boost BLEU scores; how can we prevent this?
- Introduce a Brevity penalty
- Typically this looks like $\min_r \mathit{penalty}(c, r)$ where c refers to a candidate and r refers to a reference
- Here, $\mathit{penalty}(c, r) = e^{1-|r|/|c|}$ if $|c| \leq |r|$ and otherwise 1
  - The idea is to make sure generated translations c are close (or more) in length to the reference r sentences

BLEU

- The ranges of BLEU indicate empirically how good the translation is

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

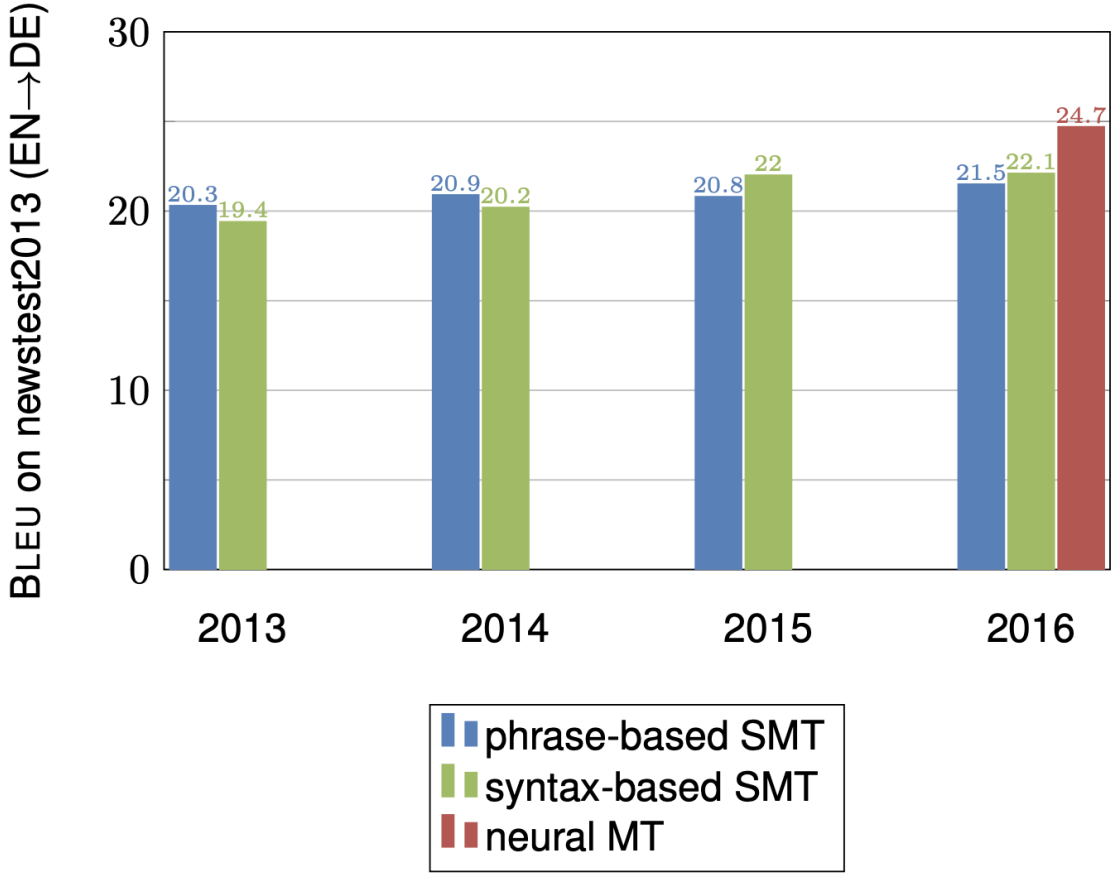
- There might be better metrics, for example "GLEU", ""RIBES", "BEER", etc
- BLEU is still the most standard metric used today

BLEU - More on Positives and Negatives

- Positives
  - It is quick to calculate and corresponds with the way a human would evaluate the same text
  - It is used very widely, which makes it easier to compare your results with other work
  - Importantly, it is language-independent making it straightforward to apply to your NLP models
  - It can be used when you have more than one ground truth sentence
- Negatives
  - It does not consider the meaning of words. It is perfectly acceptable to a human to use a different word with the same meaning eg. Use "watchman" instead of "guard". But BLEU Score considers that an incorrect word
  - It looks only for exact word matches. Sometimes a variant of the same word can be used eg. "rain" and "raining", but BLEU Score counts that as an error
  - It ignores the importance of words. With BLEU Score an incorrect word like "to" or "an" that is less relevant to the sentence is penalized just as heavily as a word that contributes significantly to the meaning of the sentence
  - It does not consider the order of words eg. The sentence "The guard arrived late because of the rain" and "The rain arrived late because of the guard" would get the same (unigram) BLEU Score even though the latter is quite different

BLEU and NMT

- In 2014, there was a fringe attempt to use NMT
- By 2016, NMT took over - Google switched to NMT
- SMT essentially died out as its performance had somewhat plateaued
- By 2019, the BLEU scores went to about 50 (or higher)



NMT Issues

- Currently, problems in NMT focus on still to be solved critical problems
- Low resource languages (what do we do if we have no data?)
- Domain mismatch between train and test data
- Out of vocabulary words
- Bias pick up (gender neutral sentences become gender specific)
- Maintaining context over longer pairs (we'll see a valuable tool for this next)
- Try it: there are lots of examples where Google Translate breaks down for specific languages

Outline

- NMT: Neural Machine Translation
- Attention

Attention: Motivation

- For a general NMT problem we saw that we feed in $(x_1, \ldots, x_{T_x})$ and expect to get $(y_1, \ldots, y_{T_y})$
- Assume we use an encoder GRU over x and that the final hidden state is h_{T_x}
- This final state becomes the initial hidden state of the decoder over y, $h_{T_x} = k_0$
- One big problem with this approach is k_0 now has to carry all the information from the input x

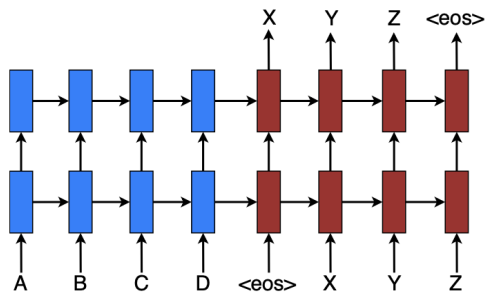


Figure 1: **Neural machine translation** – a stacking recurrent architecture for translating a source sequence `A B C D` into a target sequence `X Y Z`. Here, `<eos>` marks the end of a sentence.

- This was anticipated: to get better results, the authors of the original NMT model fed x in reverse order so that y_t and x_t are "close"

Attention: Motivation

- Imagine we have a sentence $x =$ "I went to the flower market to buy some flowers"
- Imagine the translation is $y =$ "Ich ging zum Blumenmarkt, um Blumen zu kaufen "
- If we compress everything into h_{T_x}, the resulting vector needs to contain
  - Information about the subject ("I")
  - Information about the verbs ("buy" and "went")
  - Information about the objects ("flowers" and "flower market")
  - Interaction of the subjects, verbs, and objects with each other in the sentence
- This is a lot of information!

Attention

- Typically the decoder's hidden state is a function of the input and the previous time step's hidden step,

$$k_t = f(y_t, k_{t-1})$$

- With the attention mechanism, we introduce a new time-dependent context c_t for the t^{th} decoding step
- The c_t vector is a weighted mean of the unrolled encoder hidden states $\{h_1, \ldots, h_{T_x}\}$

Attention

- Specifically, we have

$$k_t = f(y_t, k_{t-1}, c_t)$$

- Here,

$$c_t = \sum_{s=1}^{T_x} \alpha_{ts} h_s$$

- What should c_t be?
- At a high level, α_{ts} is a normalized weight telling us how important encoder state s is to decoder state t
- Idea: Let the model learn this!

Attention

- Introduce
$$e_{ts} = v_a^\intercal tanh(W_a k_{t-1} + U_a h_s)$$
- Here, v_a, W_a, U_a are parameters we want to learn
- We define $\alpha = Softmax(e)$ so that
$$\alpha_{ts} = \frac{\exp(e_{ts})}{\sum_{r=1}^{T_x} \exp e_{tr}}$$
- Given $k_t = f(y_t, k_{t-1}, c_t)$ we can then model the output in several ways
  - $\hat{y}_t = Softmax(W_y k_t + b_y)$
  - $\hat{y}_t = FF(k_t)$ where FF is some generic deep neural architecture

Attention: Benefits

- Attention improves NMT performance
  - It is useful to allow the decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows the decoder to look directly at the source; we bypass the bottleneck h_{T_x}
- Attention helps with vanishing gradient problems
  - We have provided the model with shortcuts to distant states
- Attention offers some interpretability
  - By inspecting the attention distribution, we can see what the decoder was focusing on

Attention: Results

- This is $\alpha_{ts} = \alpha_{ij}$ here, for each row we see the weight on a column (source) word

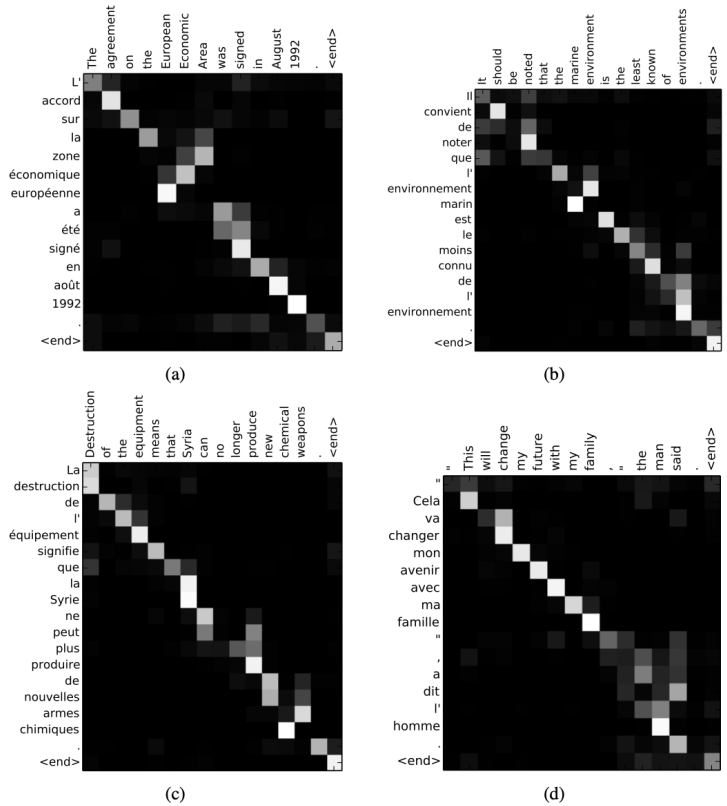


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word (see Eq. (6)), in grayscale (0: black, 1: white). (a) an arbitrary sentence. (b–d) three randomly selected samples among the sentences without any unknown words and of length between 10 and 20 words from the test set.

Attention: Results

- At the time (2014), the attention mechanism beat out all other benchmarks

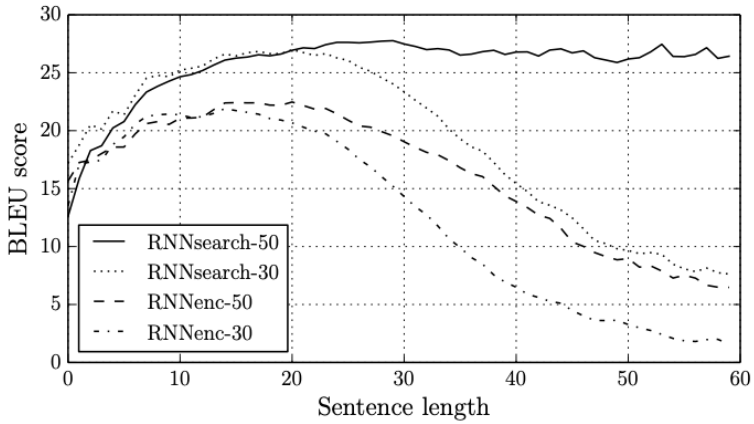


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

Table 1: BLEU scores of the trained models computed on the test set. The second and third columns show respectively the scores on all the sentences and, on the sentences without any unknown word in themselves and in the reference translations. Note that RNNsearch-50* was trained much longer until the performance on the development set stopped improving. (○) We disallowed the models to generate [UNK] tokens when only the sentences having no unknown words were evaluated (last column).

Attention

- Attention is a general idea
- The Transformer which we'll see soon is based on this idea, but the attention defined there is a bit different (slightly different than this, but at a high level the same thing)
- Attention has a few other variants
- Recall we defined $e_{ts} = v_a^T tanh(W_a k_{t-1} + U_a h_s)$ and then $\alpha = Softmax(e)$
- How can we change this?

Attention: Global Variants

- Many variants
- $e_{ts} = score(k_{t-1}, h_s)$ or $e_{ts} = score(k_t, h_s)$
  - Here k, h are respectively the target and source hidden states
  - We can make make α_{ts} depend on k_{t-1} or k_t
- There are many options for *score*
  - $score(k_t, h_s) = k_t^\intercal h_s$ (dot)
  - $score(k_t, h_s) = k_t^\intercal W_a h_s$ (general)
  - $score(k_t, h_s) = v_a^\intercal tanh(W_a k_t + U_a h_s)$ (concat)
  - $score(k_t, h_s) = W_a k_t$ (location)
- This can be called "Global" attention since we always look at the FULL encoder history

Attention: Global Variants

- Global attention focuses on the entire input

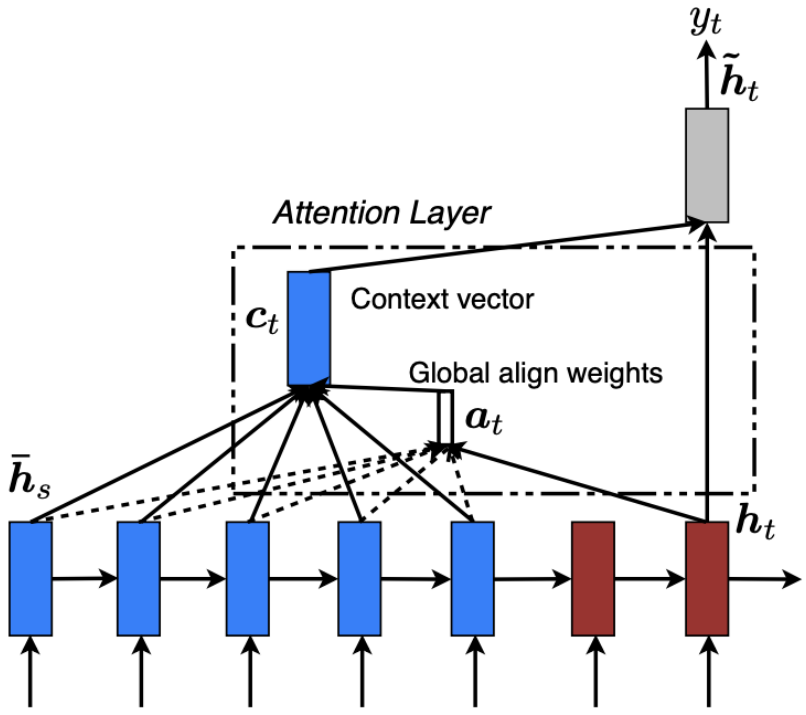


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector a_t based on the current target state h_t and all source states $\bar{h}_s$. A global context vector c_t is then computed as the weighted average, according to a_t, over all the source states.

Attention: Local Variants

- One problem with the Global attention above is that perhaps we will have too many computations for very long source sentences (documents)
- We can define "Local" attention which aims to focus on a specific subset of source sentence words
- To get this working, we can set D to a constant and then define for each target position t an index p_t in the source sentence
- To compute the context vector c_t, we just focus on an interval $[p_t - D, p_t + D]$ of source words
  - $p_t = T_x * (t/T_y)$ (local-m)
  - $p_t = T_x \sigma(v_p^\intercal tanh(W_p k_t))$ (local-p)
- For $p_t = T_x \sigma(v_p^\intercal tanh(W_p k_t))$ we can set as below to focus more on p_t:

$$\alpha_{ts} \sim score(k_t, h_s) \exp -\frac{(s - p_t)^2}{2(D/2)^2}$$

Attention: Local Variants

- Local attention focuses on a small subset of the input

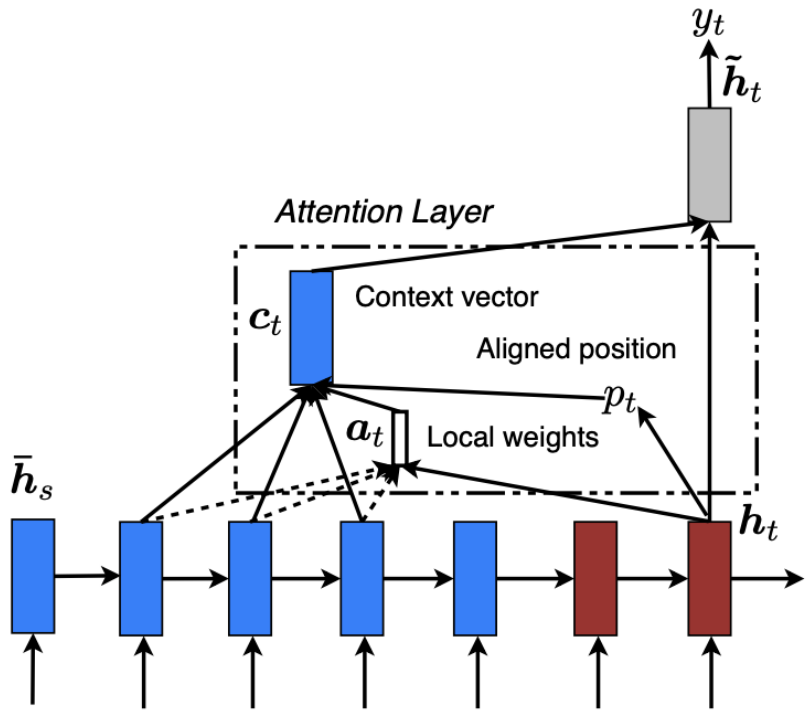


Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights $\boldsymbol{a_t}$ are inferred from the current target state $\boldsymbol{h}_t$ and those source states $\bar{\boldsymbol{h}}_s$ in the window.

Attention: Results

- Authors get some neat results, and had SOTA at the time on NMT (2015)
- Attention is THE concept the last few years, we'll build out on it in the second half of the class

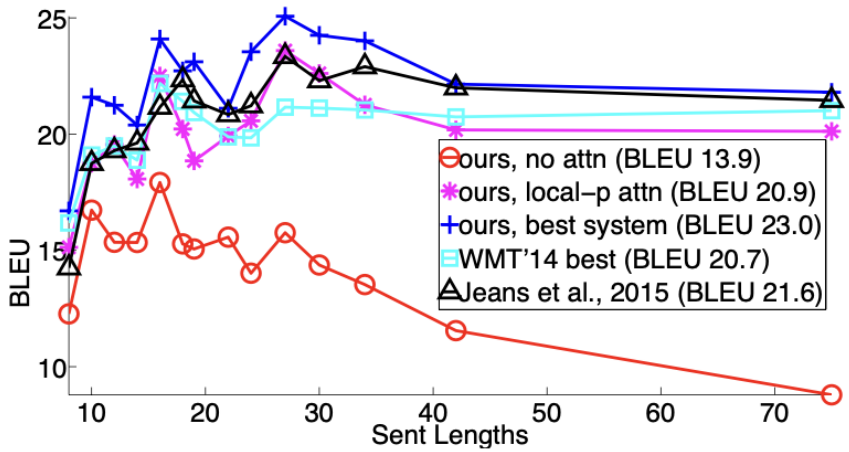


Figure 6: **Length Analysis** – translation qualities of different systems as sentences become longer.

References

- NMT original paper - 2014
- Hugging Face video on BLEU
- BLEU at Google
- Original Attention Paper - 2014
- Great Second Paper to read on Attention - 2015
- SMT Paper - A really great paper from 1993
- SMT Book - Koehn has a newer boom on NMT
- Good NMT Survey Paper
- How many layers does your NMT system need?
- SMT vs NMT performance