Neural Machine Translation

Ramaseshan Ramachandran

NEURAL MACHINE TRANSLATION

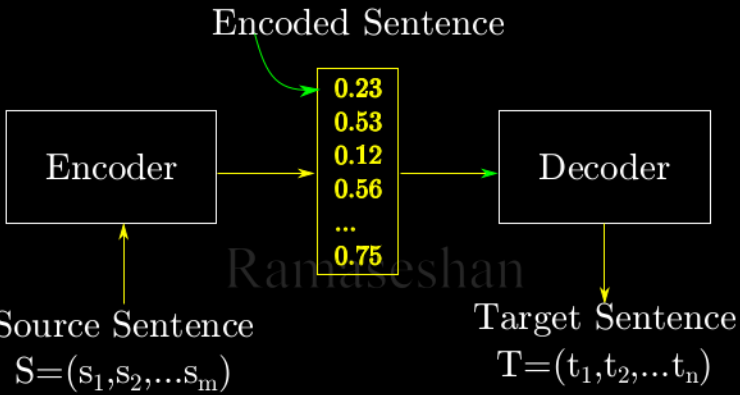
Neural Machine Translation (NMT) is the mechanism of modeling the Machine translation process using artificial neural network

We could consider translations as a sequence with the source and the destination sentences $((E_{t1}, F_{t2}))$ appearing in a time series. The words within E, F appear in different time $(t_{11}, t_{12}, t_{13}, ..., t_{1n})$ and $(t_{21}, t_{22}, t_{23}, ..., t_{2m})$, respectively

Unlike the phrase-based SMT models, NMT attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation[1]

[1] Bahdanau, D., Cho, K. & Bengio, Y., Neural machine translation by jointly learning to align and translate,

ENCODER-DECODER MODEL



- All sentences (of varying length) are encoded into fixed sized vector
- Uses fraction of the memory needed by traditional SMT models[2]
- Performance of this model decreases as the length of a source sentence increase

[2] Cho et al, On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, 2014

- Uses RNN for both encoding and decoding
- Encoder maps the variable length sentence into a fixed-length vector
- Decoder translates the vector representation back to a variable-length target sequence
- Two networks are trained jointly to maximize the conditional probability of the target sentence, given the source sentence - $P(f|e)$
- This model learns a continuous space representation of a phrase that preserves both the semantic and syntactic structure of the phrase[3]

[3] cho-et al-2014, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

RECURRENT NEURAL NETWORK

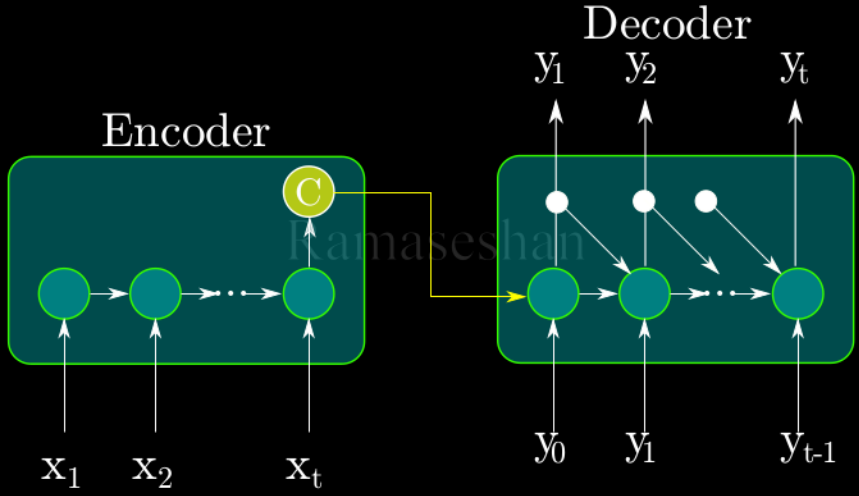
- Input units - variable length source sequence $x = (x_1, x_2, \ldots, x_T)$
- Output units - variable length target sequence $y = (y_1, y_2, \ldots, y'_T)$
- Hidden units for each input state,

$$h_t = f(h_{(t-1)}, x) \tag{1}$$

where f is a simple non-linear activation function (sigmoid or tanh) or a complex LSTM/GRU cell

- RNN is trained to predict the next word in the sequence or RNN learns a probability distribution over a sequence
- The output at each time step $t = p(x_t|x_{t-1}, \ldots x_1)$
- The output distribution (Softmax layer) size is equal to the size of the vocabulary V at every unit
- Then, $\mathrm{p}(\mathbf{x}) = \prod_{t=1}^{T} p(x_t|x_{t-1}, \ldots x_1)$

RNN-BASED ENCODER-DECODER



ENCODER - RNN FOR MT

- RNN learns to map an input sentence of variable length into a fixed-dimensional vector representation.
- It learns to decode a fixed length vector representation back into a variable length sequence
- This model learns to predict a sequence given a sequence $p(y_1, y_2, \ldots, y'_T | x_1, x_2, \ldots, x_T)$. T and T' may differ
- Encoder reads every symbol in $\mathbf{x}$, sequentially
- Hidden state changes according to 1
- C is the summary of the hidden states at time T and has encoded all the symbols in the sequence

DECODER

- This is another RNN
- This is trained to predict the next symbol y_t and generate the output sequence, given the previous state h_t
- y_t and $\mathbf{h}_t$ are conditioned on the summary from the encoder, C and its previous hidden state
- Decoder's hidden state is given by
- Conditional distribution for the next symbol is

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, y_{t-1}, \mathbf{C}) \tag{2}$$

$$P(y_t | y_{t-1}, y_{t-2} \ldots, y_1, \mathbf{C}) = g(h_{t-1}, y_{t-1}, \mathbf{C}) \tag{3}$$

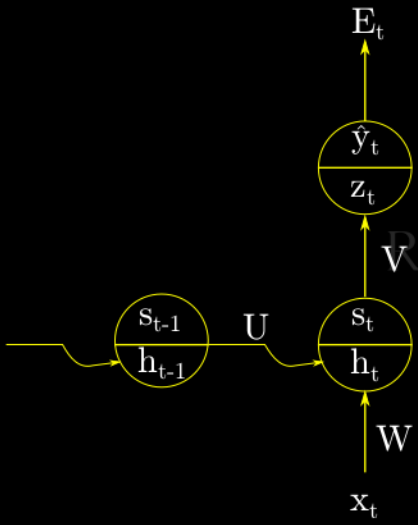
ESTIMATING MODEL PARAMETERS

Both encoder and decoder are jointly trained to maximize the conditional likelihood

$$J(\theta) = \max_{\theta} \frac{1}{N} \log p_\theta(\mathbf{y}_n|\mathbf{x}_n) \tag{4}$$

where θ is the set of model parameters that will be learned during the BPTT and $(\mathbf{x}_n, \mathbf{y}_n)$ is the source sentence sequence and target sequence pair

BPTT - DERIVATIVE FOR V



$$h_t = Wx_t + Uh_{t-1}$$

$$s_t = \tanh(h_t)$$

$$z_t = Vs_t$$

$$\hat{y}_t = softmax(z_t)$$

$$E_t = -y_t \log(\hat{y}_t)$$

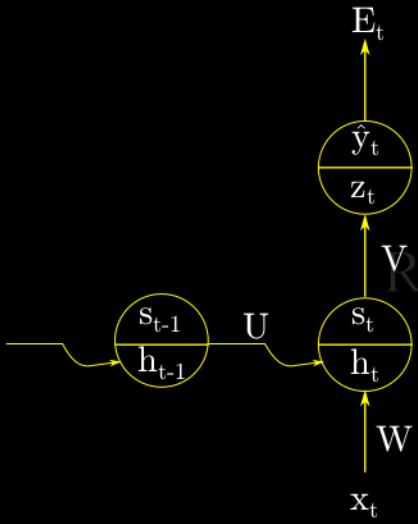
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial V} \tag{5}$$

Let $\delta^t_{out} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}$

$$\frac{\partial E_t}{\partial V} = \delta^t_{out} s_t \tag{6}$$

Here δ^t_{out} is the loss for each of the units in the output layer

BPTT - DERIVATIVE FOR W

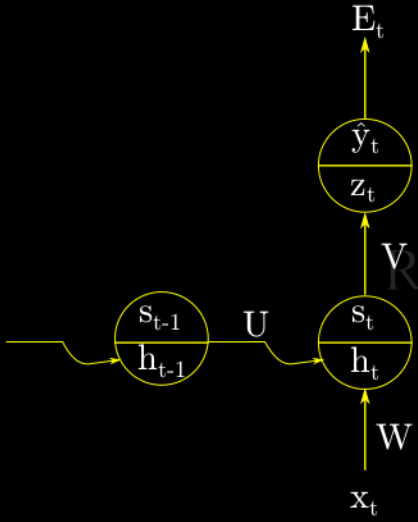


$$\frac{\partial E_t}{\partial W} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}\frac{\partial h_t}{\partial W} \tag{7}$$

$$= \delta^t_{out} V \sigma'(h_t) x_t \tag{8}$$

Since the hidden layer activation depends on the previous time state, we have another similar term δ_{t-1} that get added to (8)

BPTT - DERIVATIVE FOR U

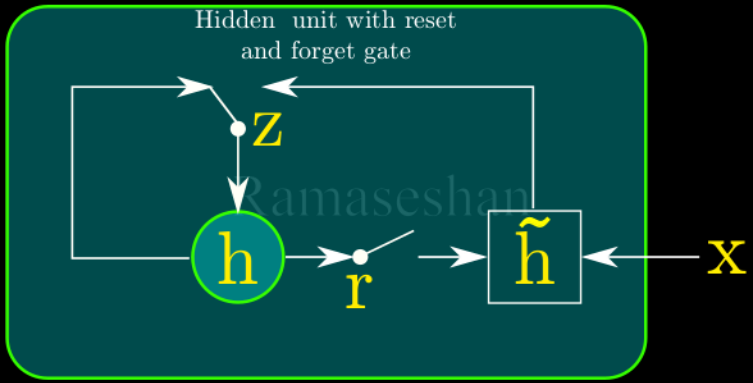


$$\frac{\partial E_t}{\partial U} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}}\frac{\partial h_t}{\partial U} \tag{9}$$

$$= \delta^t_{out} V \sigma^{'}(h_t) h_{t-1} \tag{10}$$

Since we are back propagating the error from the current state to the previous state, $\delta_{next} = \sigma(h_t) U \delta^t_{out} V \sigma^{'}(h_t)$ needs to be added

NEW HIDDEN ACTIVATION CELL



NEW HIDDEN ACTIVATION CELL

A reset and update gates are added to make this a continuously differentiable[4]

$$\text{Reset Gate } , r_j = \sigma\left([\mathbf{W}_r\mathbf{x}]_j + [\mathbf{U}_r\mathbf{h}_{t-1}]_j\right) \tag{11}$$

- σ is a logistic function and $[.]$ is the j^{th} element vector
- W_r and U_r are the input-hidden weight matrix and hidden-hidden weight matrix, respectively. These parameters will be learned during BPTT

$$\text{Update Gate } , z_j = \sigma\left([\mathbf{W}_z\mathbf{x}]_j + [\mathbf{U}_r\mathbf{h}_{t-1}]_j\right) \tag{12}$$

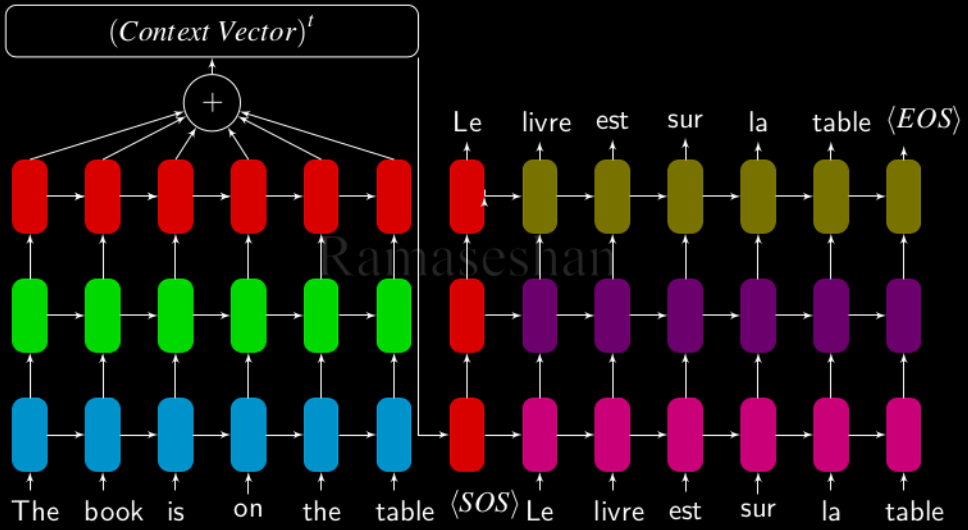
$$\tilde{h}_{t_j} = \phi\left([\mathbf{W}\mathbf{x}]_j + [U(r \odot h_{t-1})]\right) \tag{13}$$

$$h^t_j = z_j h^{t-1}_j + (1 - z_j)\tilde{h}_{t_j} \tag{14}$$

[4] Cho et al, On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, 2014

- If $r_j \approxeq 0$, then the hidden state values is reset with current input
- Update gate controls how much information from the previous hidden state will carry over to the current hidden state
- Units that learn to capture short-term dependencies will tend to have reset gates that are frequently active
- Those units that capture longer-term dependencies will have update gates active most of the time

SEQ2SEQ TRANSLATOR

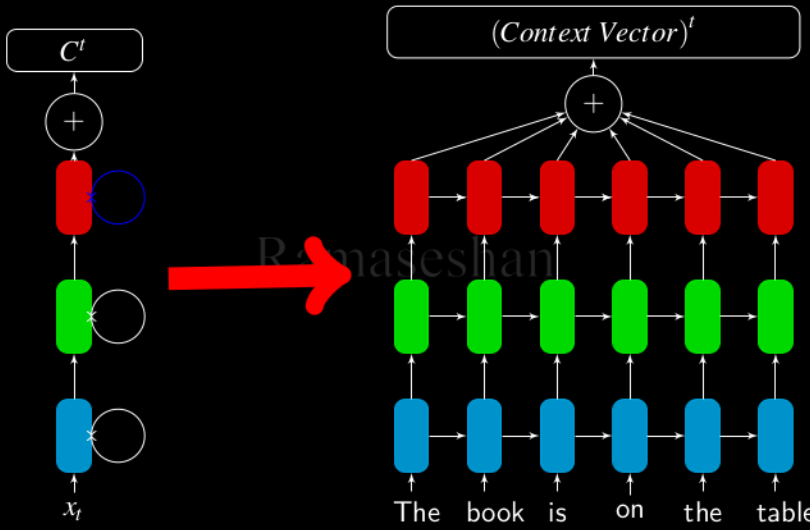


VARIATIONS IN RNN MODELS

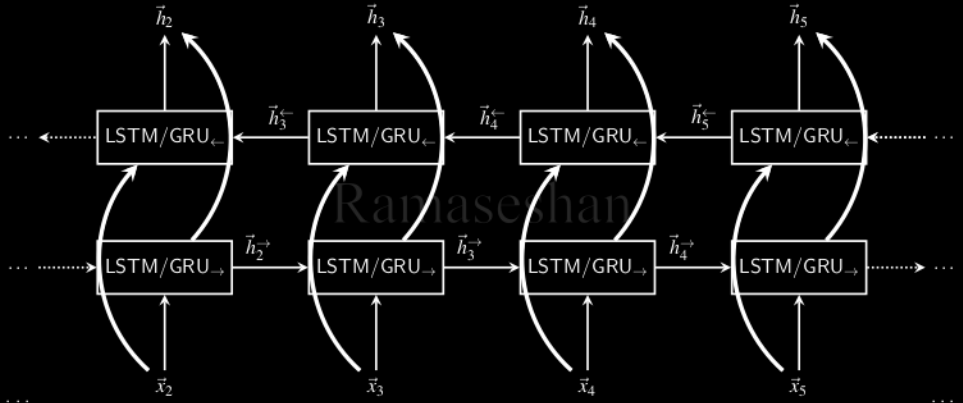
Choices vary in picking the Translation Architecture

- Directionality - Unidirectional or bidirectional
- number of hidden layers and units
- Plain vanilla RNN
- Long Short-term Memory units
- Gated Recurrent Unit
- Choice of Learning Algorithm

SEQ2SEQ ENCODER



BIDIRECTIONAL RNN



APPLICATIONS

- Translation
- Summarization
- Dialog
- Code generation!

Ramaseshan

RNN WITH ALIGNMENT

- The objective of attention is to capture the information from the passage tokens that is relevant to the contents of the translation
- Different parts of an input have different levels of significance
- Different parts of the output may even consider different parts of the input as "important"
- The purpose of the attention mechanism is to let the decoder *peek* at the relevant information encapsulating the passage and the question as it generates the answer
- Attention mechanisms provide the decoder network with the entire input sequence at every decoding step; the decoder can then decide what input words are important at any point in time

- The attention-based model learns to assign significance to different parts of the input for each step of the output.
- In the context of translation, attention can be thought of as "alignment."
- Bahdanau et al.[5] argue that the attention scores α_{ij}, at decoding step i, signify the words in the source sentence that align with word i in the target.
- We can use attention scores to build an alignment table. It is a table mapping of words in the source to corresponding words in the target sentence - based on the learned encoder and decoder from our Seq2Seq NMT system.

[5] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv, [arXiv:1409.0473].

BAHDANAU ET AL. NMT MODEL

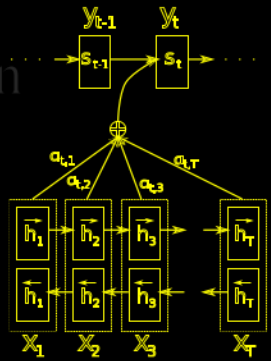
Input is a sequence of words $x_1, ..., x_n$
Target sentence is again a sequence of words $y_1, ..., y_m$

Encoder

Let $(h_1, ..., h_n)$ be the hidden vectors representing the input sentence. These vectors are the output of a bi-LSTM/bi-GRU for instance, and capture contextual representation of each word in the sentence

Decoder

The hidden states s_i of the decoder are computed using a recursive formula of the form $s_i = f(s_{i-1}, y_{i-1}, c_i)$, where s_{i-1} is the previous hidden vector, y_{i-1} is the generated word at the previous step, and c_i is a context vector that capture the context from the original sentence that is relevant to the time step i of the decoder.



NMT WITH ATTENTION

Conditional probability for each output neuron

$$P(y_i|y_1, y_2, \ldots, x) = g(y_{i-1}, s_i, c_i) \tag{15}$$

where s_i if the RNN hidden neuron at time i and

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{16}$$

The context vector c_i depends on the sequence of annotations $(h_1, h_2, \ldots . h_{Tx})$. Each h_i contains information about every word with a strong focus on context words surrounding the i^{th} word of the input sequence.

The context vector c_i is computed as the weighted sum of these annotations h_i

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{17}$$

α_{ij} of each annotation h_j is computed by

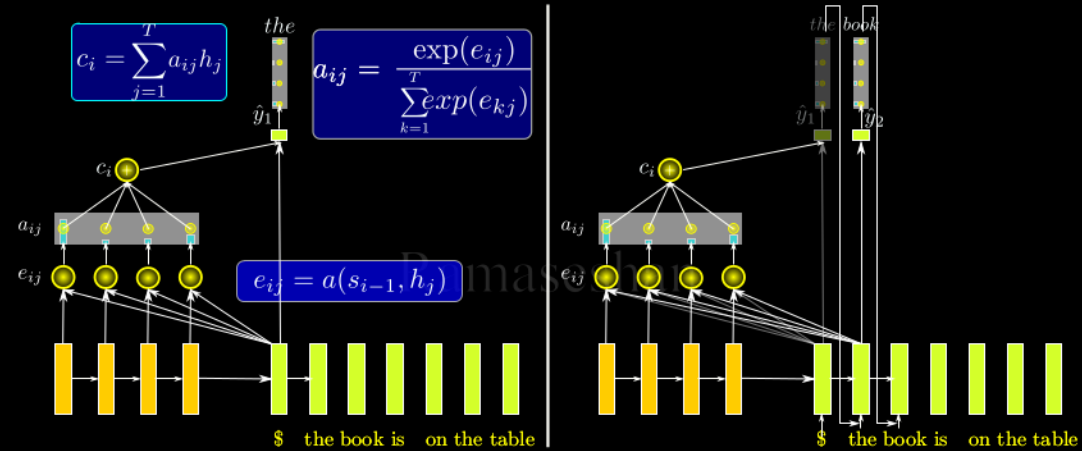
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{t_x} \exp(e_{ik})}, \tag{18}$$

where,

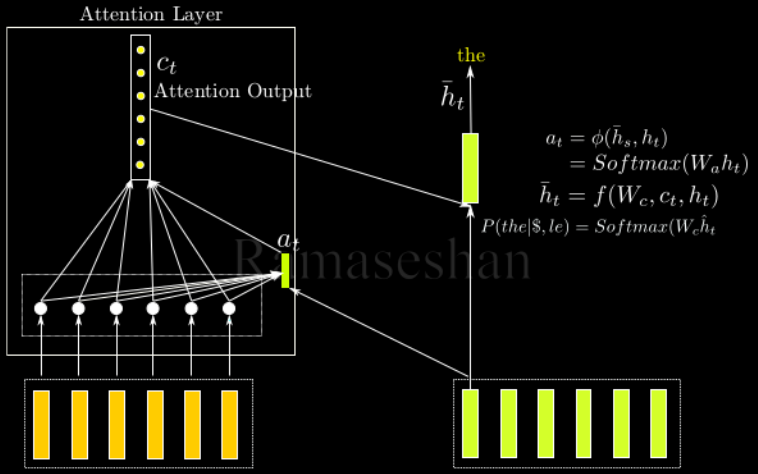
$$e_{ij} = a(s_{i-1}, h_j) \tag{19}$$

is the alignment model. This learns how well the inputs surrounding position j and the output at position i match

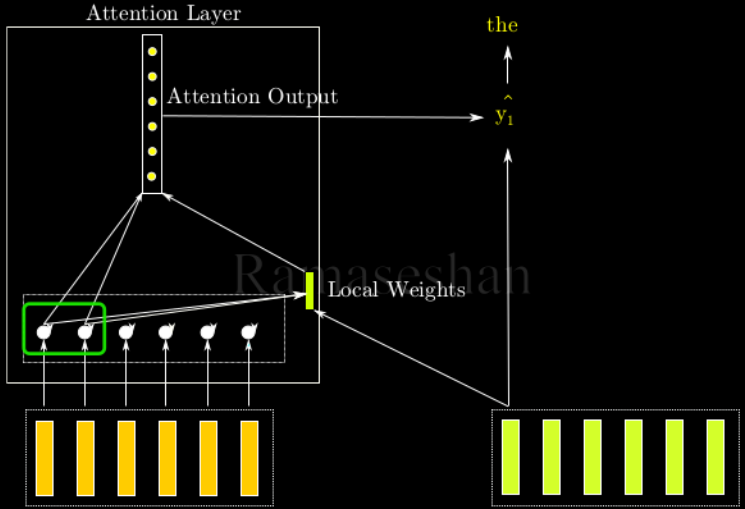
- The alignment is explicitly computed and not latent
- This alignment model is also trained along with the translation model
- α_{ij} is the probability that the target word y_i is aligned to the source x_j
- c_i is the expected annotation over all possible annotations α_{ij}
- α_{ij} or e_{ij} reflects the importance of the annotation h_j wrt to the previous hidden state s_{i-1} of the target. This enables the next state s_i to generate y_i
- The decoder decides which part of the input is important to generate a respective translation rather than depending on the encoded vector of the entire sentence
- Decoder has control over the input sequence and selectively learns to align words/phrases automatically

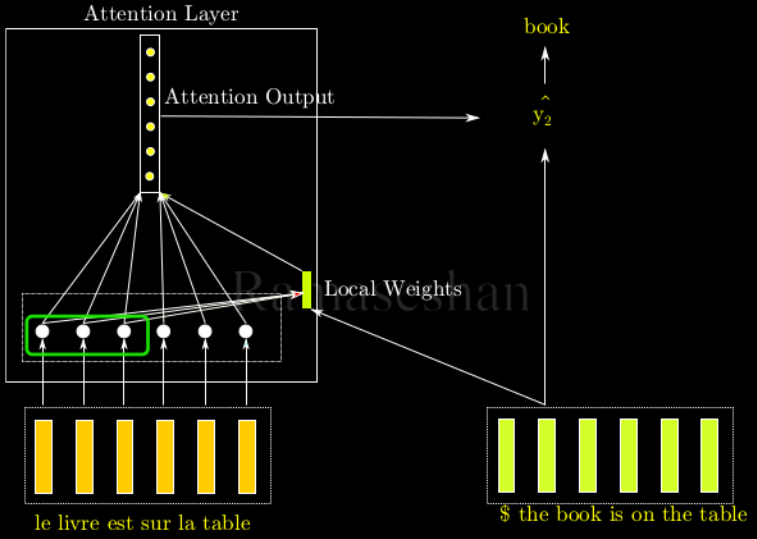

$c_i = \sum_{j=1}^{T} a_{ij} h_j$
$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{kj})}$
$e_{ij} = a(s_{i-1}, h_j)$
the
book
$\hat{y}_1$
$\hat{y}_2$
c_i
a_{ij}
e_{ij}
$ the book is on the table

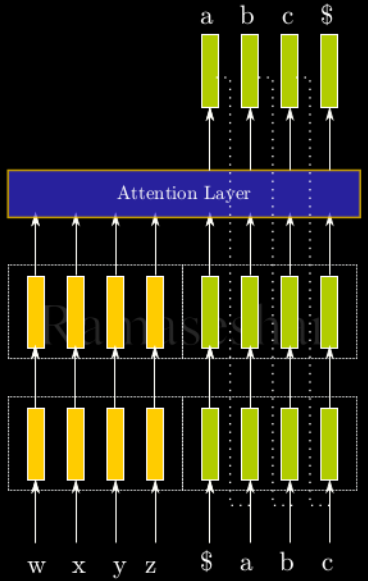
TRANSLATION WITH GLOBAL ATTENTION



TRANSLATION WITH LOCAL ATTENTION




Attention Layer
Attention Output
book
$\hat{y}_2$
Local Weights
le livre est sur la table
$ the book is on the table



Source: Minh-Thang Luong et al, Effective Approaches to AIttention-based Neural Machine Translation

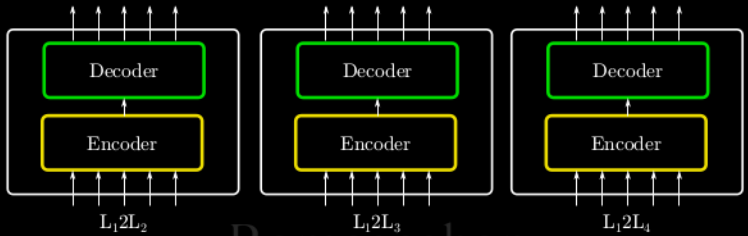
A TYPICAL SETUP

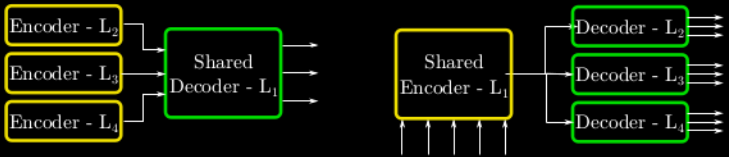
Sentence pairs	3-5M
English words	110M
French words	116M
Vocabulary	≈50K (Source and Target)
Word Embedding size	1000
Hidden layer	1000 LSTM cells
Stacked Hidden Layer	4-8
Learning Rate	Initially as high as 1 and exponential reduction
Training	
Mini batch Gradient Descend size	128
Training Time	1 GPU - about 7-10 days
Evaluation	Bleu - scores ranging from 27-32

ADVANTAGES OF ATTENTION

- Ability to focus on significant part of the sentence
- Ability to peek into source sentence
- Reduces the problem of vanishing gradient
- Alignments are found automatically - no need to train
- Improves NMT performance for alignment

DIFFERENT TYPE OF NMT



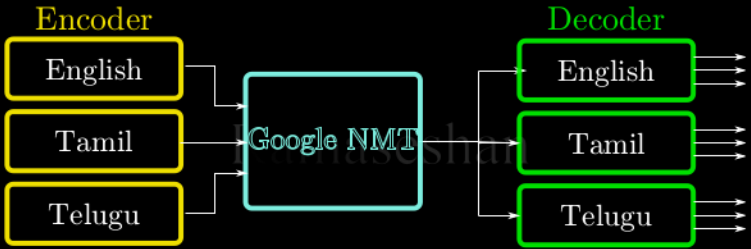


NMT FROM GOOGLE - ZERO-SHOT TRANSLATION

- Moved away from maintaining Seq2Seq model for every pair of languages
- A single system that translates between any two languages even in the absence of the training corpus for these two languages
  - Assume that only examples of Japanese-English and Korean-English translations are available, Google found hat the multilingual NMT system trained on this data could actually generate reasonable Japanese-Korean translations.
  - Is it trained create the Interlingua?
  - Is the system learning a common representation or a translational knowledge?

Ref: Johnson et el. 2016, "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation"

ZERO-SHOT TRANSLATION



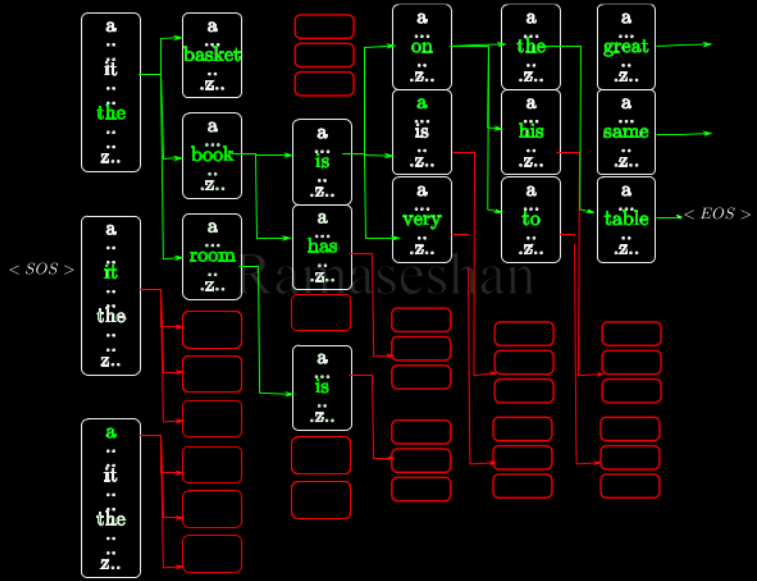
BEAM SEARCH

Beam search is a heuristic search algorithm that selects a few candidate hypothesis from $|V|$. It reduces memory requirement by using only a $M < |V|$ candidates using a score.

- Maintain M candidates/hypothesis at each time step - $C_t = (x_1^1, ...x_t^1)...(x_1^M ...x_t^M)$
- Compute C_{t+1} by expanding C_t and keeping the best M candidates
- $\tilde{C} = \bigcup_{i=1}^{M} C_{t-1}^i$

Typical Beam width of size 5-10 used in NMT. The BLEU scores computed using Beam search using B=5-10 are comparable

BEAM SEARCH - BEAM WIDTH = 3



1. Use all possible partial translations - exhaustive search
2. Beam size, $b = 1$ - greedy search - Words are predicted until the $< EOS >$ is found
3. $b > 1$ - several hypotheses
4. Each hypothesis will be produced until the $< EOS >$ is found
5. Each hypothesis will have a translation
6. The length of all hypothesis may not be the same
7. We could use different ***terminate*** conditions
   - Fixed time steps
   - Compute until $< EOS >$ is reached for each hypothesis
8. Use either log probability or product of conditional probability to find the scores for each hypothesis that maximizes
   - $P(y_1, y_2, ...y_m|\mathbf{X}) = \prod_{t=1}^{T} P(y_t| < SOS >, \ldots, y_{t-1}, \mathbf{X})$
   - $P(y_1, y_2, ...y_m|\mathbf{X}) = \sum_{t=1}^{T} \log P(y_t| < SOS >, \ldots, y_{t-1}, \mathbf{X})$

COMMON GRADIENT DESCEND TYPES

- Stochastic Gradient
- Batch
- Mini-batch

Ramaseshan