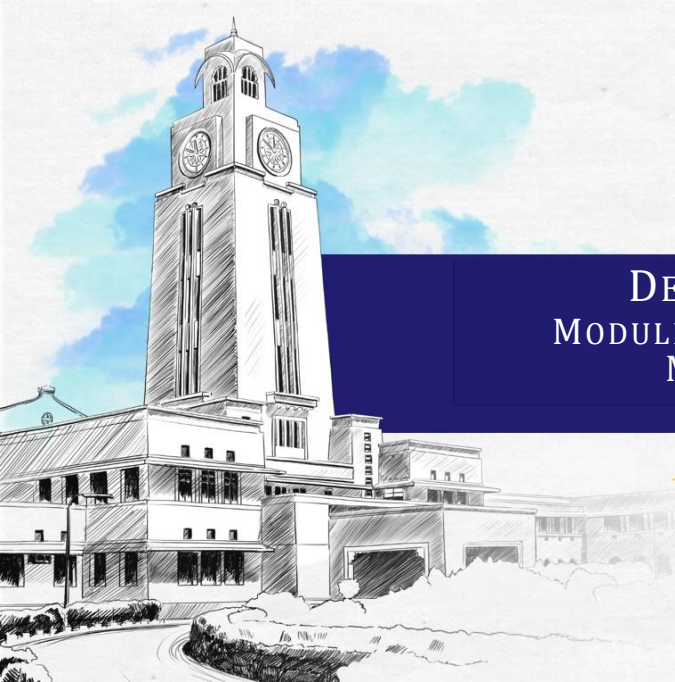
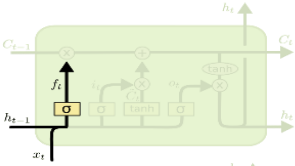


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

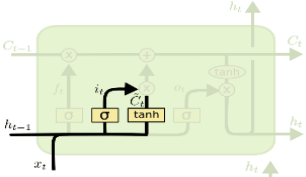
Deep Learning

Module # 7 : Attention Mechanism

Dr YVK Ravi Kumar

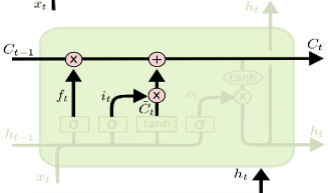


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



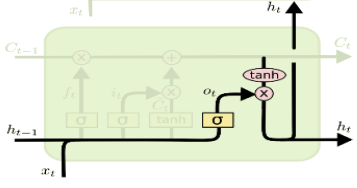
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

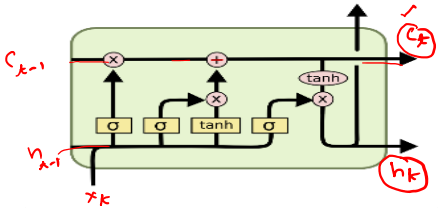
Updating the cell state



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

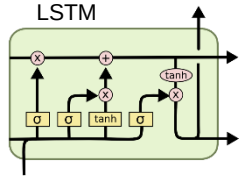
Decide what part of the cell state to output

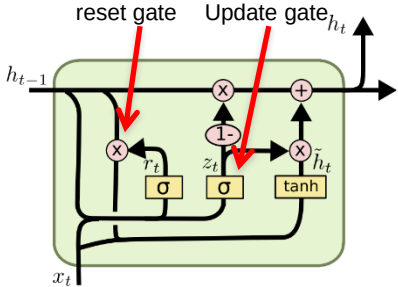


i_t decides what component is to be updated.
C'_t provides change contents

GRU – gated recurrent unit

(more compression)





$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

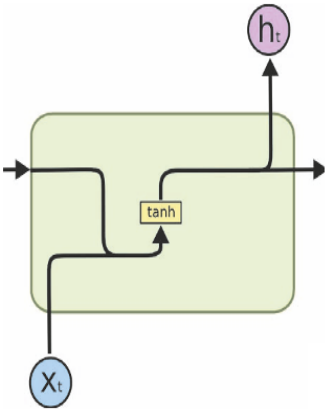
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

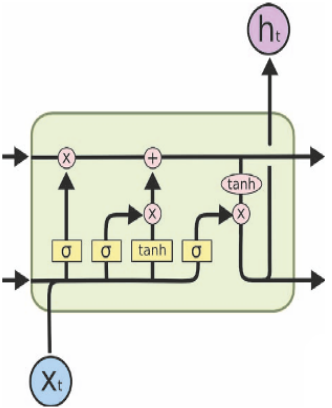
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- It combines the forget and input into a single update gate.
- It also merges the cell state and hidden state.
- This is simpler than LSTM. There are many other variants too.

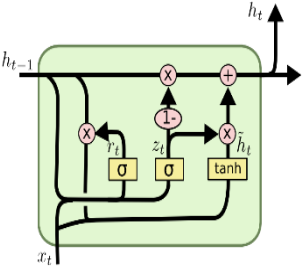
X,*: element-wise multiply



(a) RNN



(b) LSTM



Example: Name entity recognition

He said "**Teddy** bear is soft."

He said "**Teddy** Roosevelt was a President."

NER

RNN → ?

Bi RNN ✓

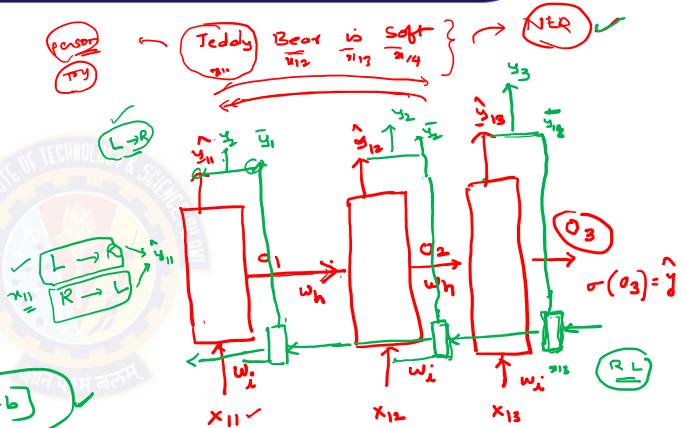
Bi LSTM ✓

RNN

$$\overrightarrow{h_t} = \tanh\left(\overrightarrow{w}\overrightarrow{h}_{t-1} + \overrightarrow{U}x_t + \overrightarrow{b}\right)$$

$$\overleftarrow{h_t} = \tanh\left(\overleftarrow{w}h_{t+1} + \overleftarrow{U}x_t + \overleftarrow{b}\right)$$

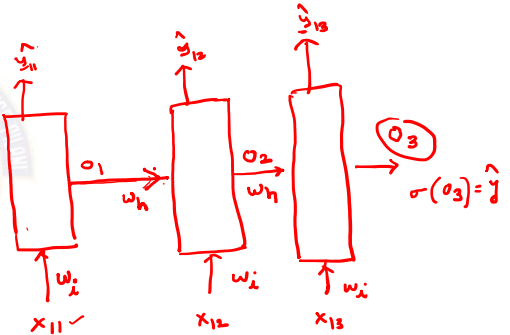
$$Y_t = \sigma\left[V\left[\overrightarrow{h_t}, \overleftarrow{h_t}\right] + b\right]$$



$$f(x_{11} w_i) = o_1$$

$$f(x_{12} w_i + o_1 w_h) = o_2$$

$$f(x_{13} w_i + o_2 w_h) = o_3 \quad \sigma(o_3) = \text{output}$$



$$f(x_{11} w_i) = o_1$$

$$f(x_{12} w_i + o_1 w_h) = o_2$$

$$f(x_{13} w_i + o_2 w_h) = o_3$$

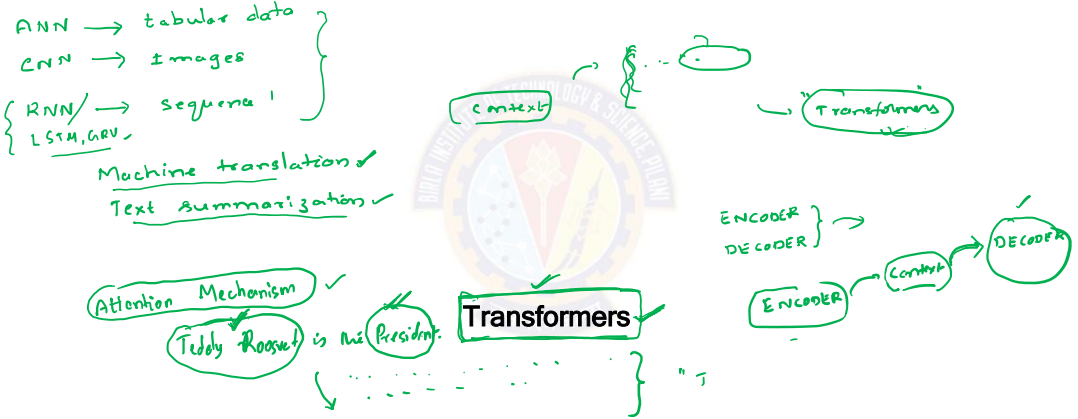
o_3 → $\sigma(o_3)$ = output

BIDIRECTIONAL RNN (BRNN)

- Forward and backward connections.
- The blocks can be RNN, GRU, LSTM.
- Mostly used in the NLP.
- Acyclic graph
- Example: Name entity recognition
  He said “Teddy bear is soft.”
  He said “Teddy Roosevelt was a President.”

SUMMARY

- Use GRU, when dependency is short. Eg: Weather forecasting
- Use LSTM, when dependency is long. Eg: NLP Translation
- Use BRNN, dependency is in both direction. Eg: Stock prediction


ANN → tabular data
CNN → Images
RNN / LSTM, GRU → sequence
Machine translation
Text summarization
context
Transformers
ENCODER
DECODER
ENCODER
context
DECODER
Attention Mechanism
Teddy Roosevelt is the President.
Transformers

Attention Is All You Need

2000 - 2014 → RNN / LSTM
2014 → "attention Mechanism"
2017 → Transformers
BERT / GPT
2017
→ Vision transfor
GenAI — chatGPT

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

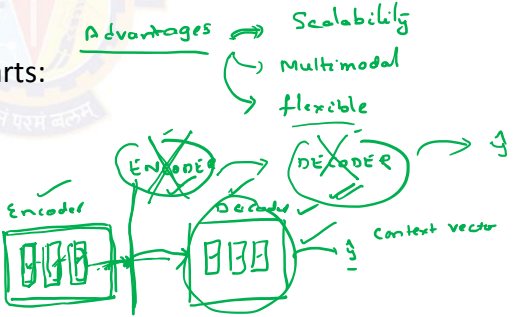
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

CNN + Nomad

Vision Transformers

- The **RNN** and **LSTM** neural models were designed to process language and perform tasks like classification, summarization, translation, and sentiment detection
    - RNN: Recurrent Neural Network
    - LSTM: Long Short Term Memory
- In both models, layers get the next input word and have access to some previous words, allowing it to use the word's left context
- They used word embeddings where each word was encoded as a vector of 100-300 real numbers representing its meaning

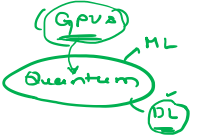
- Transformers extend this to allow the network to process a word input knowing the words in both its left and right context
- This provides a more powerful context model
- Transformers add additional features, like **attention**, which identifies the important words in this context
- And break the problem into two parts:
  - An encoder (e.g., Bert)
  - A decoder (e.g., GPT)



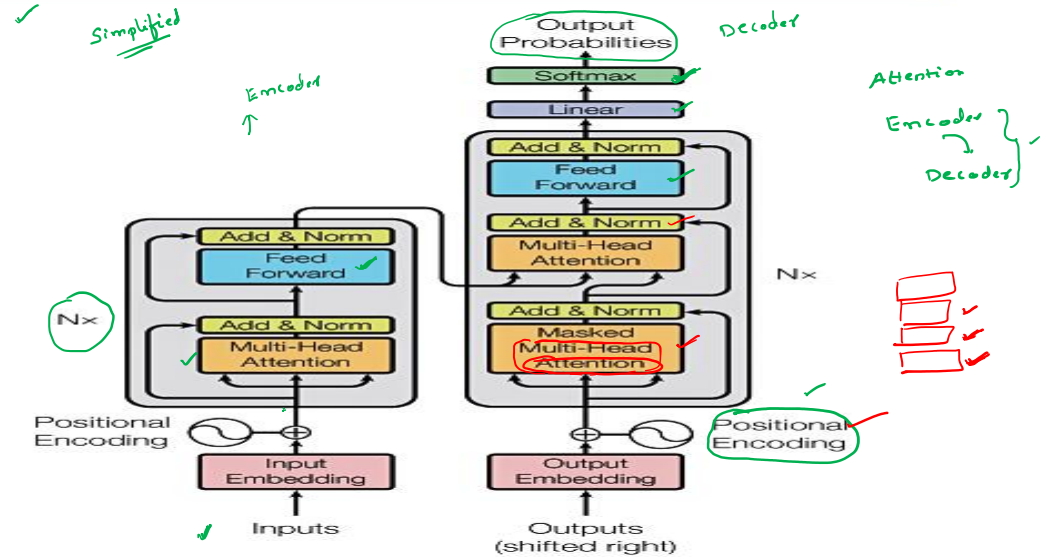
- A transformer uses an **encoder stack** to model input, and uses **decoder stack** to model output (using input information from encoder side)
- If we do not have input, we just want to model the "next word", we can get rid of the encoder side of a transformer and output "next word" one by one. This gives us **GPT**
- ✓ If we are only interested in training a language model for the input for some other tasks, then we do not need the decoder of the transformer, that gives us **BERT**

Transfer learning → "Hugging face"

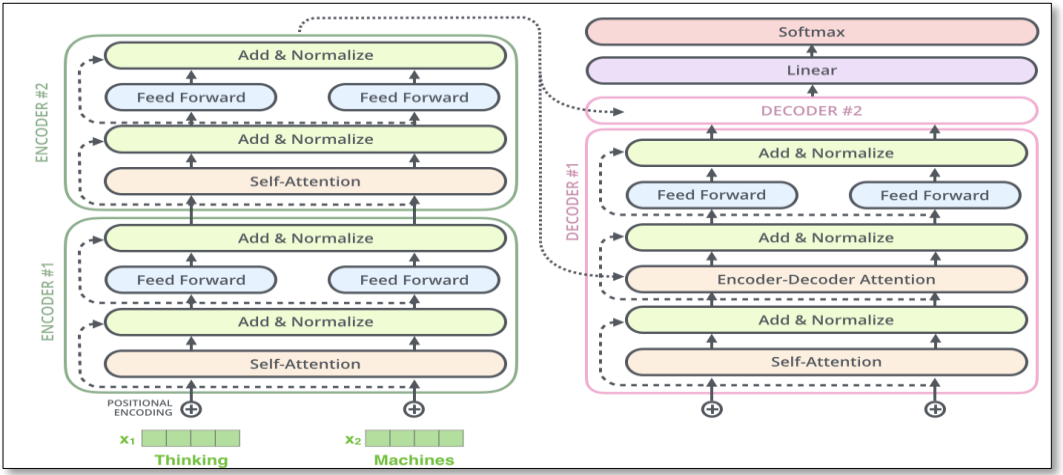
Disadvantages → ? → interpretability
→ computational expo



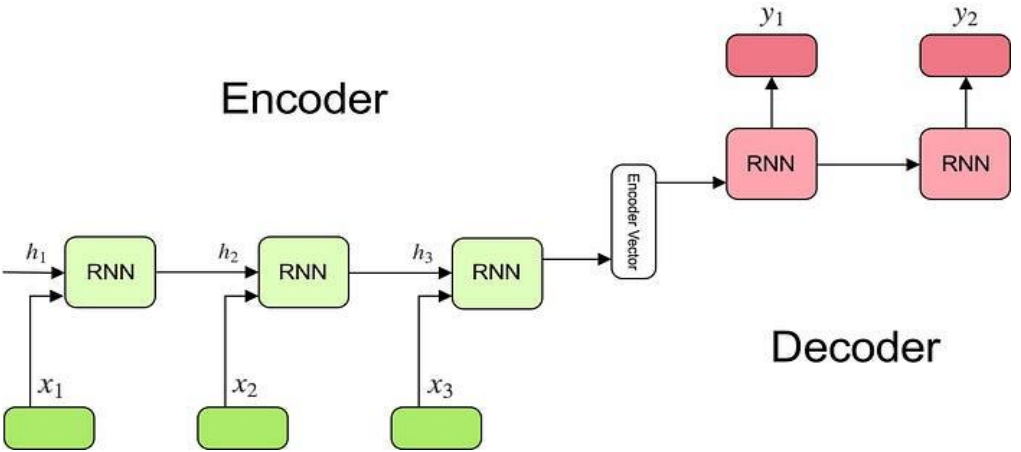
Transformer



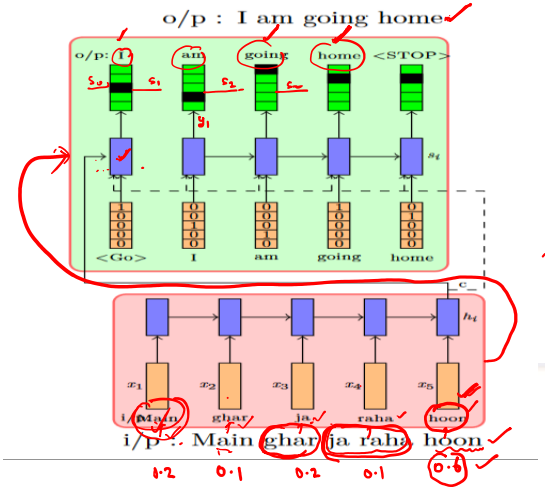
Transformer



Encoder - Decoder



Attention



o/p : I am going home

t_1 : [1 0 0 0 0] main

t_2 : [0 0 0 0 1] Hoon

t_3 : [0 0 0.5 0.5 0] ja raha

t_4 : [0 1 0 0 0] ghar

i/p : Main ghar ja raha hoon

Vanilla Encoder decoder

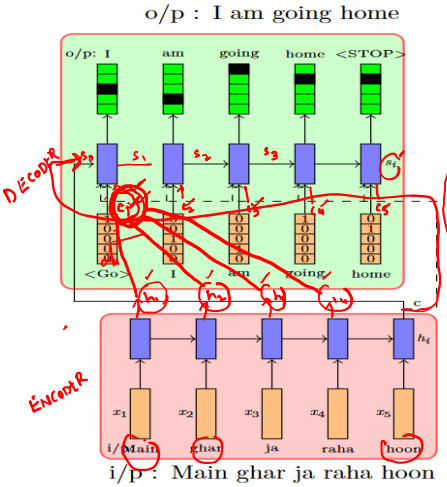
attention

attention input

attention input

Some of these slides are taken from We Prof Mitesh M Kapra's lecture notes.Thanks

Attention



$[y_{i-1}, s_{i-1}]$ — Encoder / Decoder

$[y_{i-1}, s_{i-1}, c_i]$ — attention Mech

$c_i = \alpha_{i1} h_1 + \alpha_{i2} h_2 + \alpha_{i3} h_3$

$c_{ij} = \sum \alpha_{ij} h_j$ attention

α → how to get

$\alpha_{ij} = f(h_j, s_{i-1})$

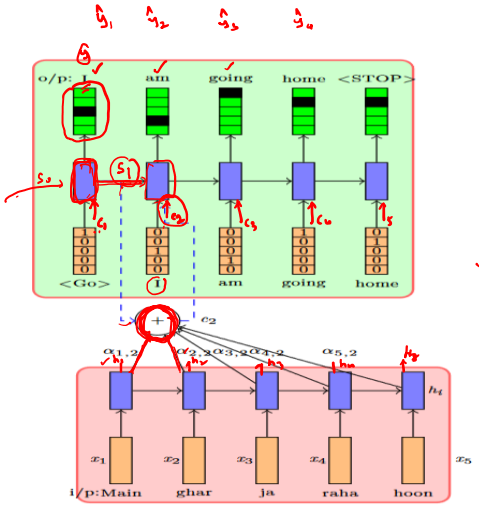
$\alpha_{21} = f(h_1, s_1)$

α values

Bahdanau attention → ANN

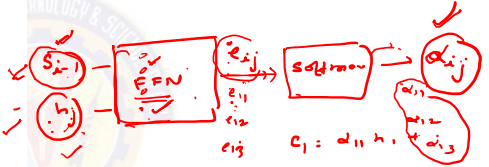
Luong attention → vector dot product

Attention



Bahdanau attention

(Multiplicative attention)

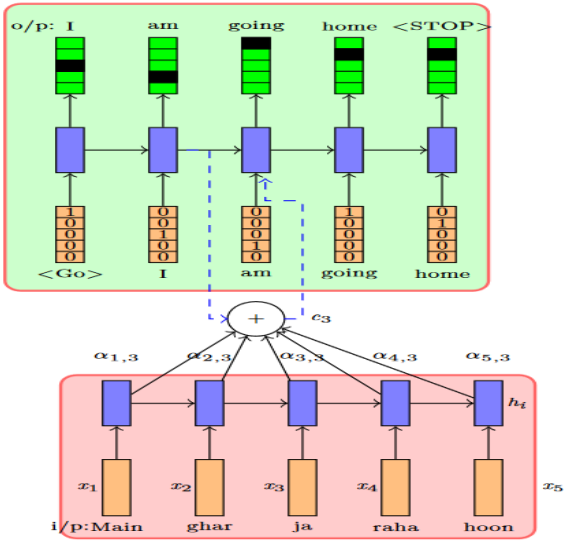


$c_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3$

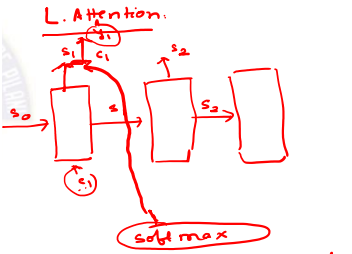
$c_2 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3$

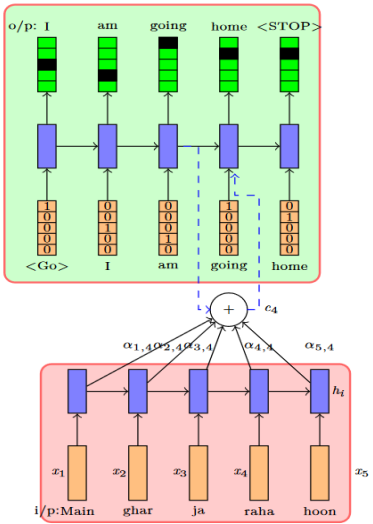
Luong attention — $f(s_i, h_j)$

'vector dot product'

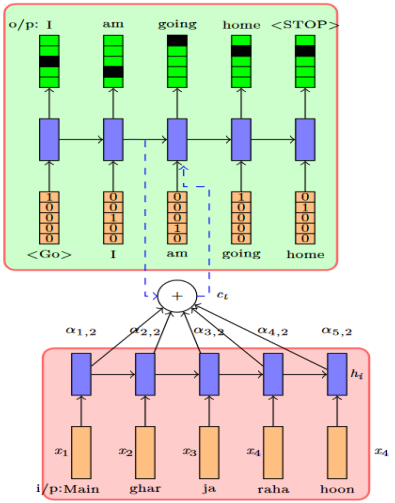


For example at timestep 3, we can just take a weighted average of the representations of 'ja' and 'raha'





- We could just take a weighted average of the corresponding word representations and feed it to the decoder
- For example at timestep 3, we can just take a weighted average of the representations of 'ja' and 'raha'
- Intuitively this should work better because we are not overloading the decoder with irrelevant information (about words that do not matter at this time step)



oracle

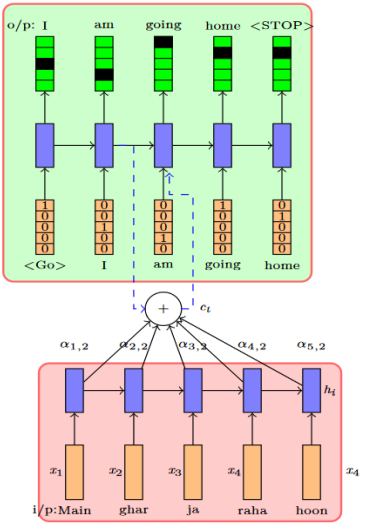
- The machine will have to learn this from the data
- To enable this we define a function

$$e_{jt} = f_{ATT}(s_{t-1}, c_j)$$

This quantity captures the importance of the jth input word for decoding the tth output word

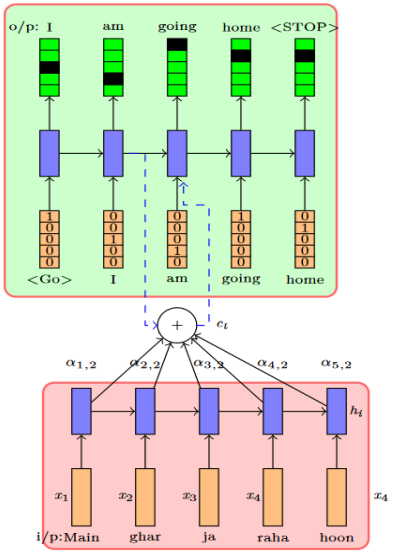
We can normalize these weights by using the softmax function

$$\alpha_{jt} = \frac{exp(e_{jt})}{\sum_{j=1}^{M} exp(e_{jt})}$$



$$\alpha_{jt} = \frac{exp(e_{jt})}{\sum_{j=1}^{M} exp(e_{jt})}$$

- α_{jt} denotes the probability of focusing on the j^{th} word to produce the t^{th} output word



This model would make a lot of sense if were given the true α's at training time

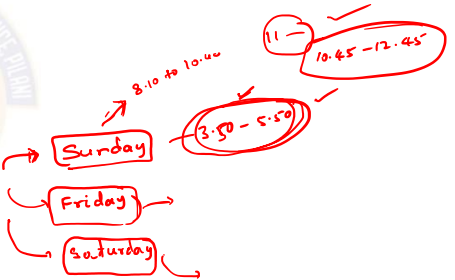
$$\alpha_{tj}^{true} = [0, 0, 0.5, 0.5, 0]$$

$$\alpha_{tj}^{pred} = [0.1, 0.1, 0.35, 0.35, 0.1]$$

We could then minimize $\mathscr{L}(\alpha^{true}, \alpha^{pred})$

Bahadanau Attention

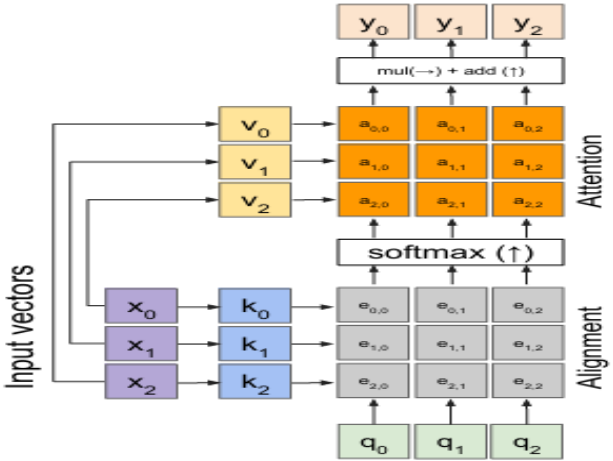
Bahadanau Attention



Luong Attention

Luong Attention

General attention layer

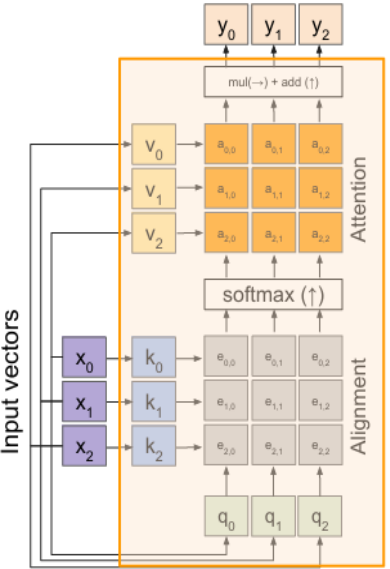


Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

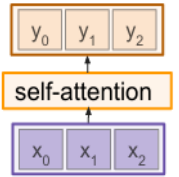
Self attention layer - attends over sets of inputs



Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)



Hierarchical Attention

Context

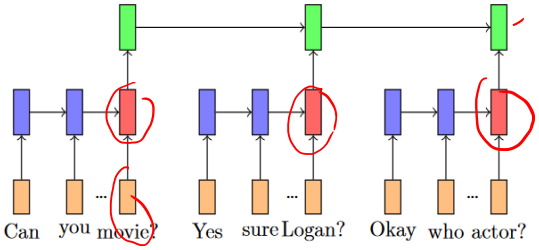
U: Can you suggest a good movie?

B: Yes, sure. How about Logan?

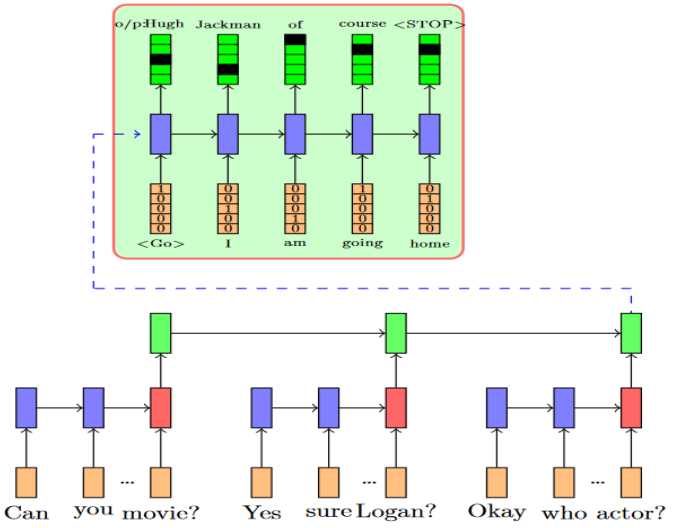
U: Okay, who is the lead actor?

Response

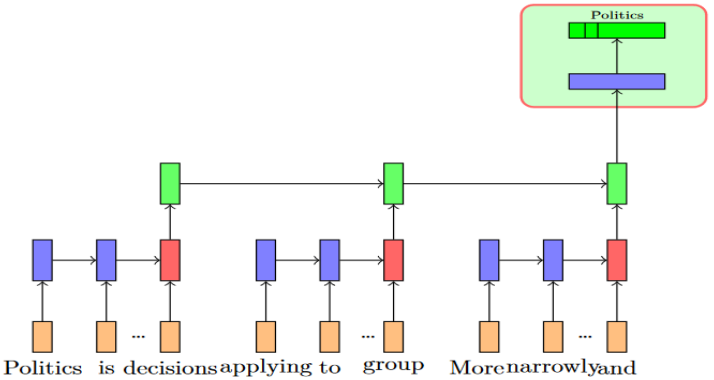
B: Hugh Jackman, of course



- We could think of a two level hierarchical RNN encoder
- The first level RNN operates on the sequence of words in each utterance and gives us a representation
- We now have a sequence of utterance representations (red vectors in the image)
- We can now have another RNN which encodes this sequence and gives a single representations for the sequences of utterances


o/pHugh Jackman of course <STOP>
<Go> I am going home
Can you ... movie?
Yes sure ... Logan?
Okay who ... actor?

Politics is the process of making decisions applying to all members of each group. More narrowly, it refers to achieving and ...



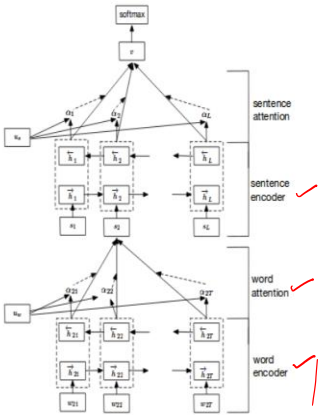


Figure: Hierarchical Attention Network [Yang et al.]

- Data: $\{Document_i, class_i\}_{i=1}^{N}$
- Word level (1) encoder:

$$h_{ij} = RNN(h_{ij-1}, w_{ij})$$
$$u_{ij} = tanh(W_w h_{ij} + b_w)$$
$$\alpha_{ij} = \frac{exp(u_{ij}^T u_w)}{\sum_t exp(u_{it}^T u_w)}$$
$$s_i = \sum_j \alpha_{ij} h_{ij}$$

- Sentence level (2) encoder:

$$h_i = RNN(h_{i-1}, s_i)$$
$$u_i = tanh(W_s h_i + b_s)$$
$$\alpha_i = \frac{exp(u_i^T u_s)}{\sum_i exp(u_i^T u_s)}$$
$$s = \sum_i \alpha_i h_i$$