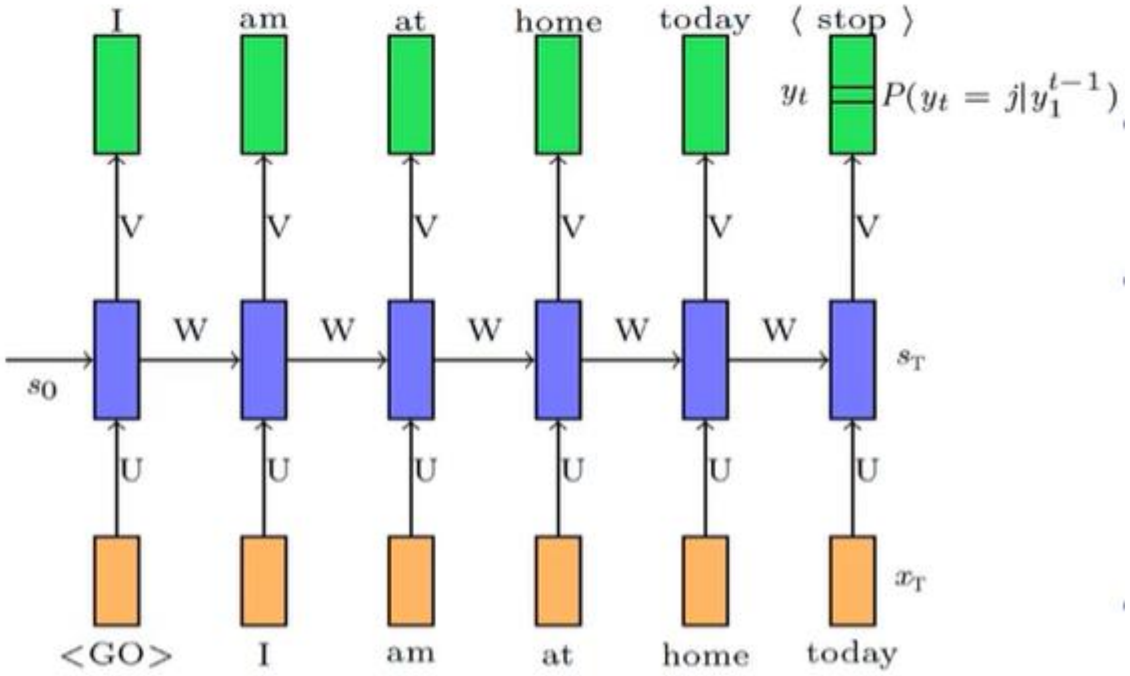
AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Introduction to Deep Learning

Encoder- Decoder Models

Recap - RNN

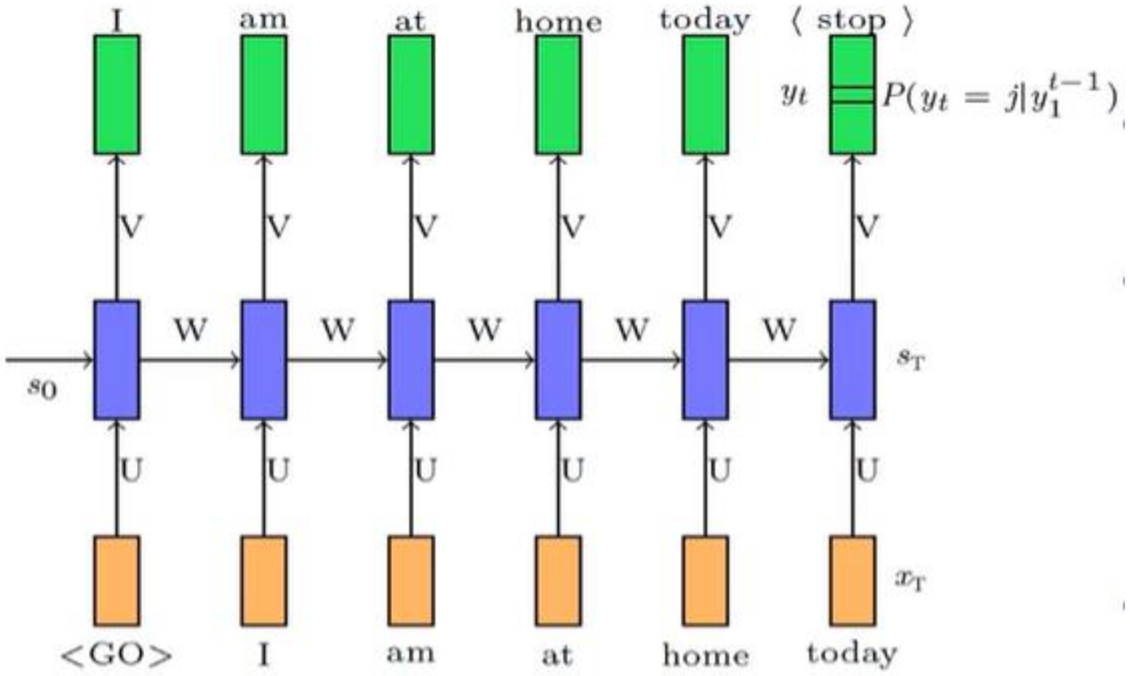


- We will start by revisiting the problem of language modeling
- Informally, given '$t - i$' words we are interested in predicting the t^{th} word
- More formally, given $y_1, y_2, ..., y_{t-1}$ we want to find

$$y^* = argmax\ P(y_t|y_1, y_2, ..., y_{t-1})$$

- Let us see how we model $P(y_t|y_1, y_2 ... y_{t-1})$ using a RNN
- We will refer to $P(y_t|y_1, y_2 ... y_{t-1})$ by shorthand notation: $P(y_t|y_1^{t-1})$

Recap - RNN



- We are interested in

$$P(y_t = j|y_1, y_2 \ldots y_{t-1})$$

where $j \in V$ and V is the set of all vocabulary words

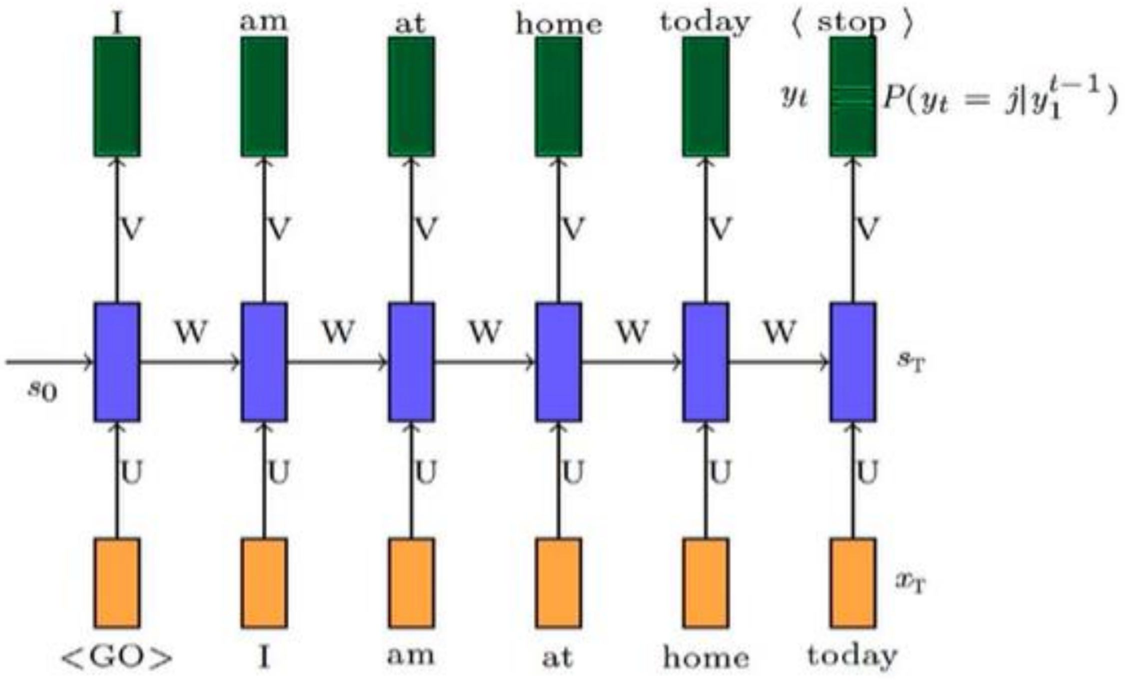
- Using an RNN we compute this as

$$P(y_t = j|y_1^{t-1}) = softmax(Vs_t + c)_j$$

- In other words we compute

$$\begin{aligned} P(y_t = j|y_1^{t-1}) &= P(y_t = j|s_t) \\ &= softmax(Vs_t + c)_j \end{aligned}$$

Recap - RNN



Data: India, officially the Republic of India, is a country in South Asia. It is the seventh-largest country by area, ...

- **Data:** All sentences from any large corpus (say wikipedia)
- **Model:**

$$s_t = \sigma(Ws_{t-1} + Ux_t + b)$$

$$P(y_t = j|y_1^{t-1}) = softmax(Vs_t + c)_j$$

- **Parameters:** U, V, W, b, c
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{t=1}^{T} \mathscr{L}_t(\theta)$$

$$\mathscr{L}_t(\theta) = -\log P(y_t = \ell_t|y_1^{t-1})$$

$$s_t = \sigma(U\, x_t + W s_{t-1} + b)$$

$$s_t = \mathrm{RNN}(\, s_{t-1}, x_t)$$

$$\tilde{s}_t = \sigma(\, W(o_t \odot \; s_{t-1}) + U x_t + b)$$

$$s_t = i_t \odot \; s_{t-1} + (1 - i_t) \odot \tilde{s}_t$$

$$s_t = \mathrm{GRU}(\, s_{t-1}, x_t)$$

$$\tilde{s}_t = \sigma(\, W\, h_{t-1} + U x_t + b)$$

$$s_t = f_t \odot \; s_{t-1} + i_t \odot \tilde{s}_t$$

$$h_t = o_t \odot \sigma(s_t)$$

$$h_t, s_t = \mathrm{LSTM}(\, h_{t-1}, s_{t-1}, x_t)$$

Encoder – Decoder for Image Captioning

Image Captioning Problem

- So far we have seen how to model the conditional probability distribution $P(y_t|y_1^{t-1})$
- More informally, we have seen how to generate a sentence given previous words
- What if we want to generate a sentence given an image?

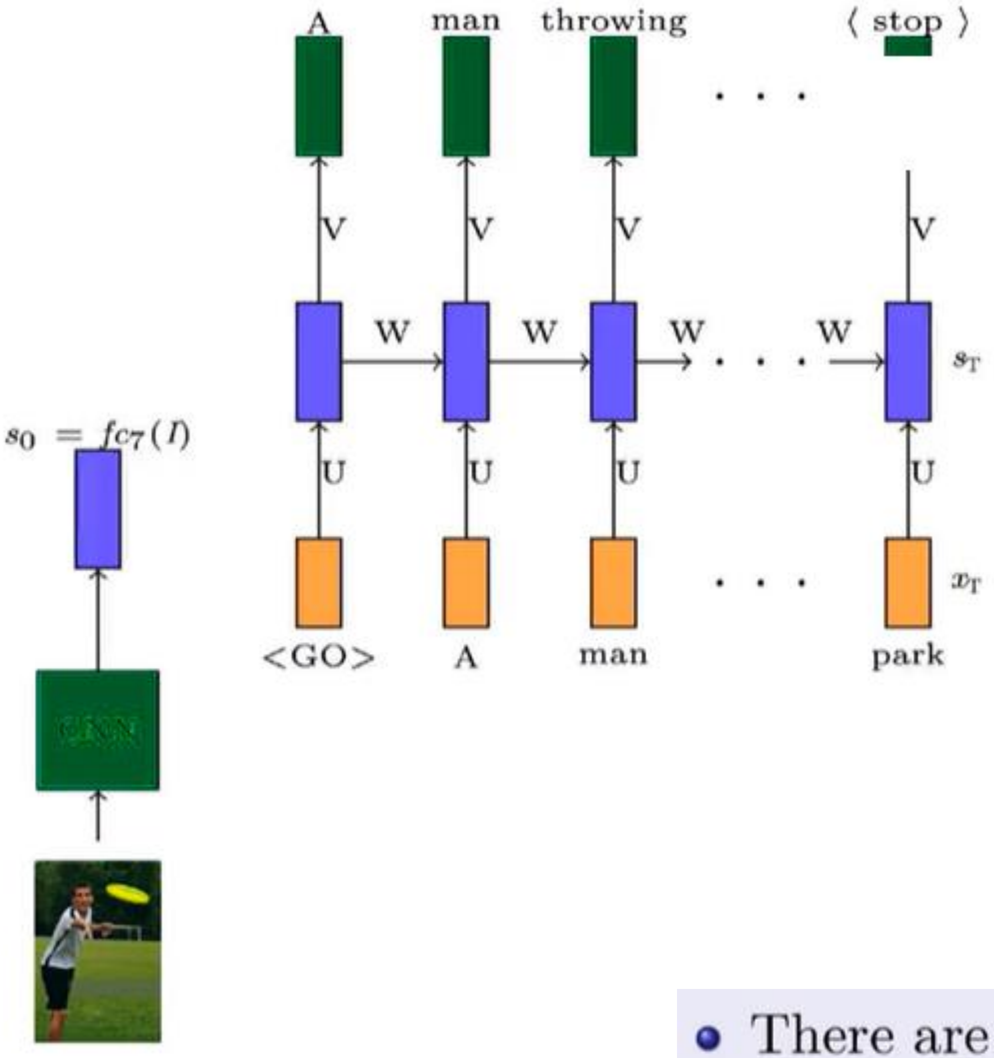
A man throwing a frisbee in a park

- Earlier we modeled $P(y_t|y_1^{t-1})$ as

$$P(y_t|y_1^{t-1}) = P(y_t = j|s_t)$$

- Where s_t was a state capturing all the previous words
- We could now model $P(y_t = j|y_1^{t-1}, I)$ as $P(y_t = j|s_t, f_{c_7}(I))$

Encoder – Decoder for Image Captioning



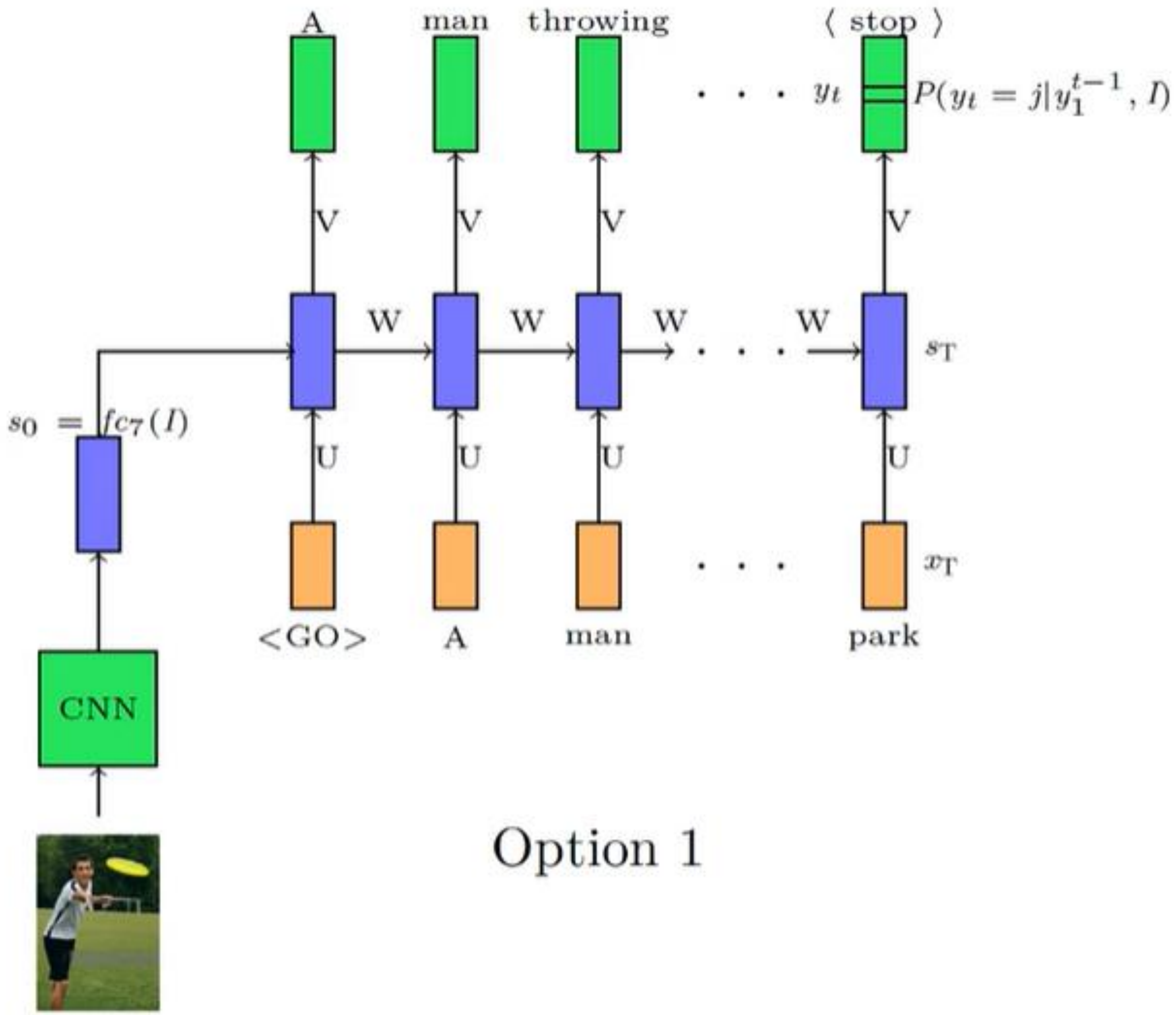
- Earlier we modeled $P(y_t|y_1^{t-1})$ as

$$P(y_t|y_1^{t-1}) = P(y_t = j|s_t)$$

- Where s_t was a state capturing all the previous words
- We could now model $P(y_t = j|y_1^{t-1}, I)$ as $P(y_t = j|s_t, f_{c_7}(I))$

- There are many ways of making $P(y_t = j)$ conditional on $f_{c_7}(I)$

Encoder – Decoder for Image Captioning



Option 1

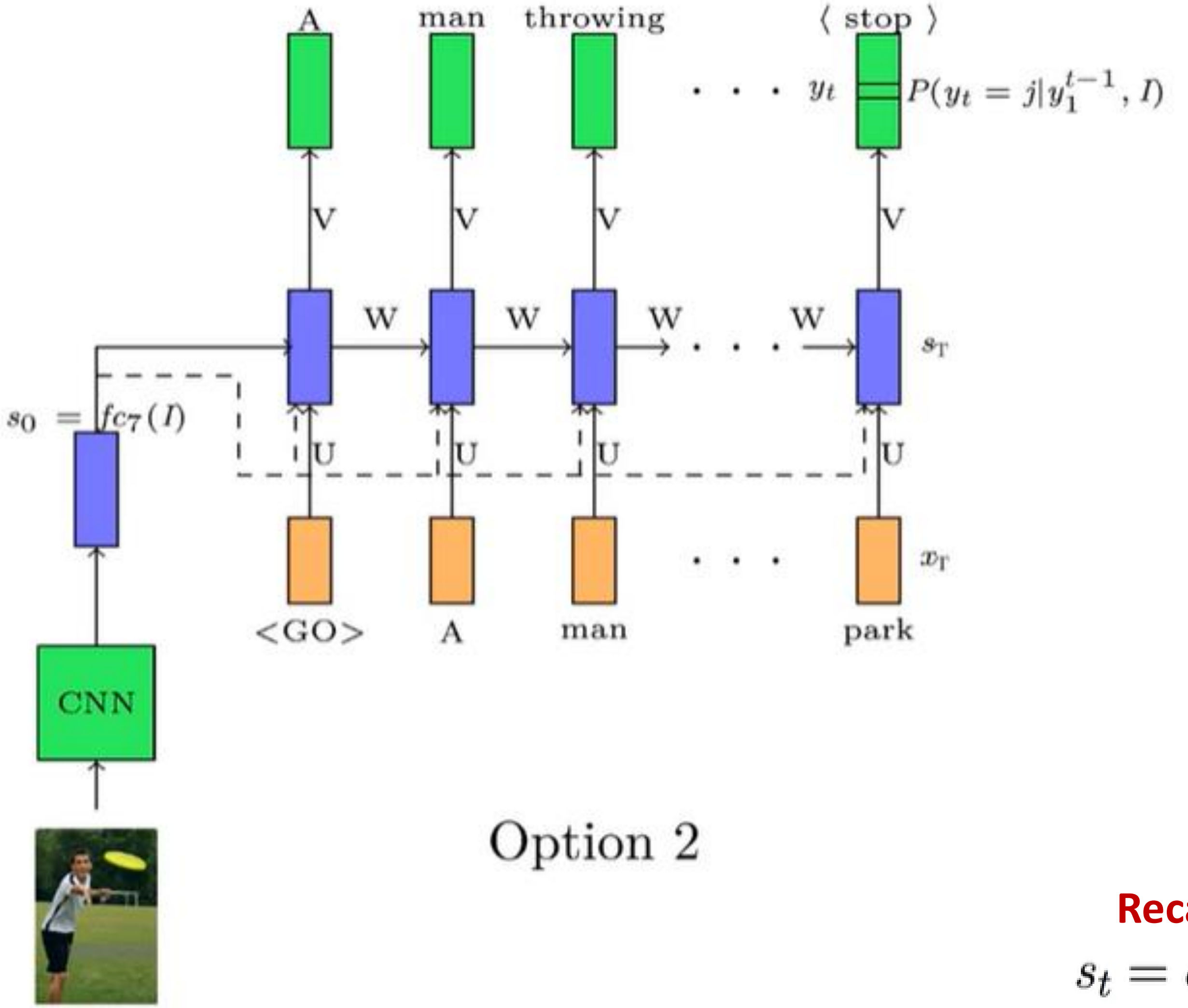
- **Option 1:** Set $s_0 = f_{c_7}(I)$
- Now s_0 and hence all subsequent s_t's depend on $f_{c_7}(I)$

$$S_1 = \sigma(W_{S0} + U_{x1} + b)$$

S0 computed from CNN, x1 <GO>

$$S_2 = \sigma(W_{S1} + U_{x2} + b)$$

Encoder – Decoder for Image Captioning



Option 2

- **Option 2:** Another more explicit way of doing this is to compute

$$s_t = RNN(s_{t-1}, [x_t, f_{c_7}(I))]$$

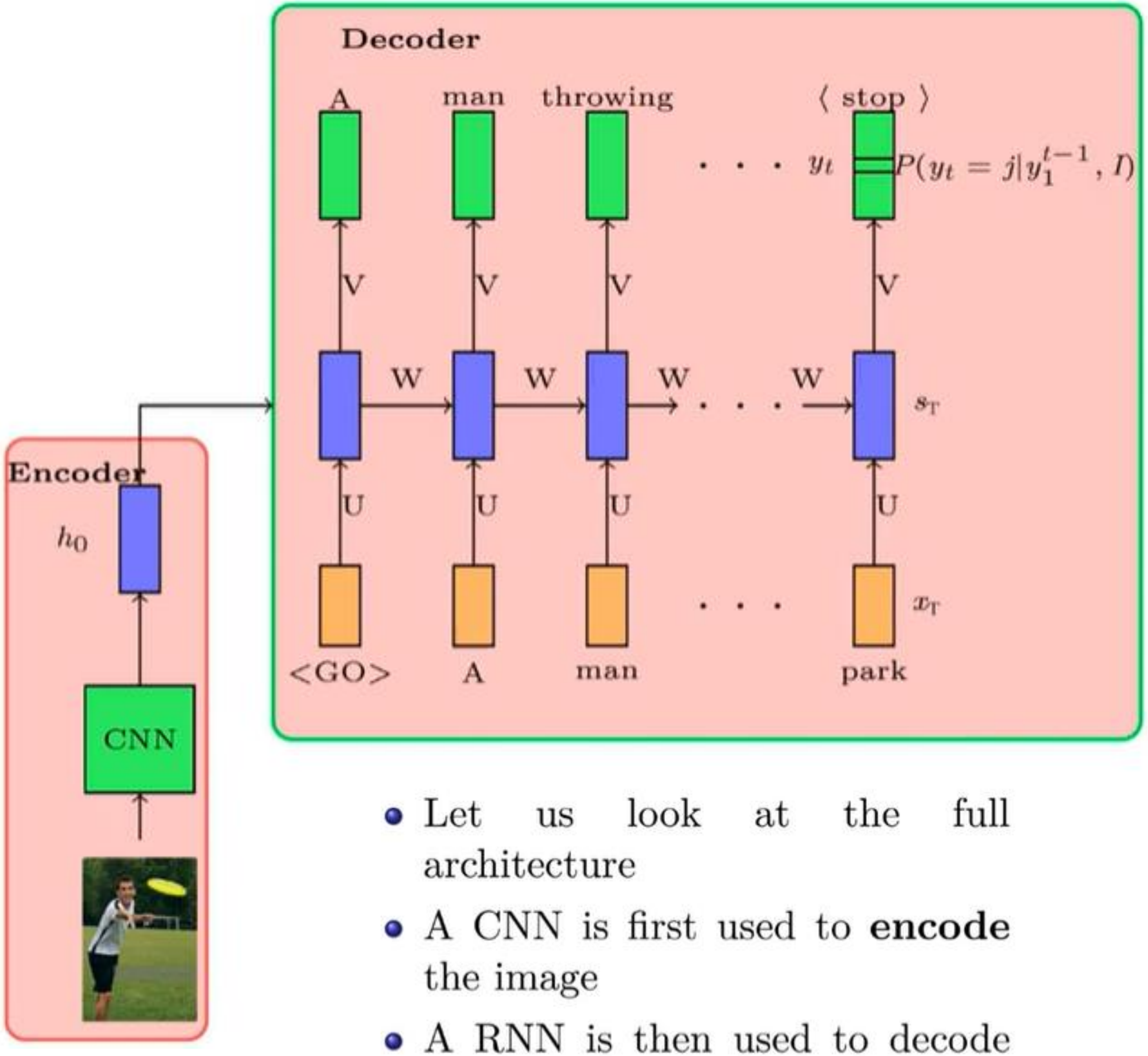
- In other words we are explicitly using $f_{c_7}(I)$ to compute s_t and hence $P(y_t = j)$

$$S_2 = \sigma\left(W_{S1} + U_{[x,1s1]} + b\right)$$

Concatenate x1 and S1

Recap

$s_t = \sigma(U\,x_t + W s_{t-1} + b)$ $\quad s_t = \text{RNN}(\, s_{t-1}, x_t)$



- Let us look at the full architecture
- A CNN is first used to **encode** the image
- A RNN is then used to decode (generate) a sentence from the encoding

- **Task:** Image captioning
- **Data:** $\{x_i = image_i, \; y_i = caption_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**

$$s_0 = CNN(x_i)$$

  - **Decoder:**

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t|y_1^{t-1}, I) = softmax(Vs_t + b)$$

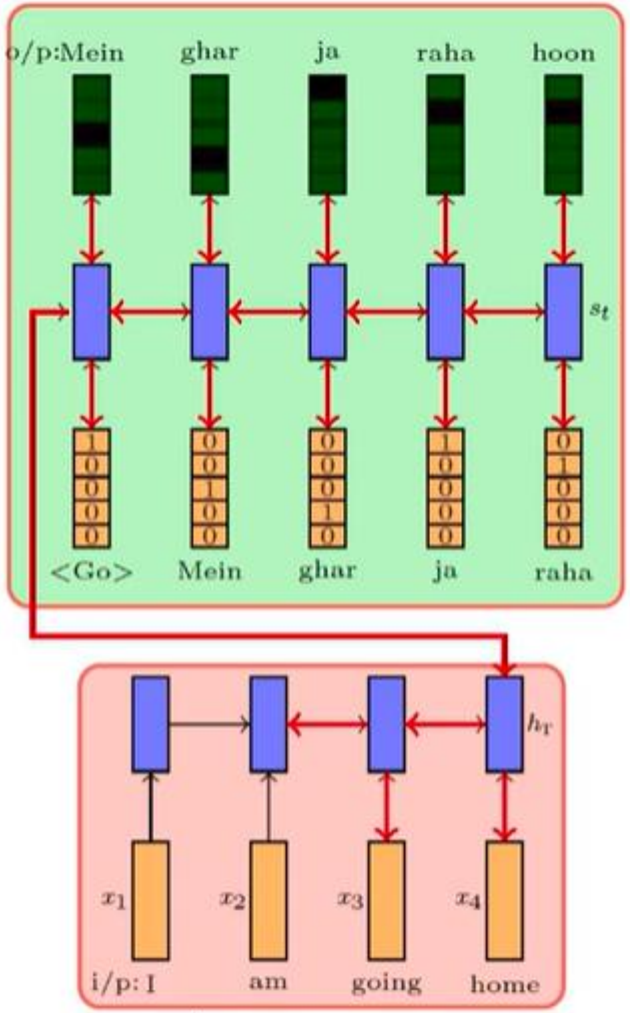
- **Parameters:** $U_{dec}, \; V, \; W_{dec}, \; W_{conv}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) \quad = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, I)$$

- **Algorithm:** Gradient descent with backpropagation

Encoder – Decoder for Machine Translation(seq2seq)

o/p : Mein ghar ja raha hoon



i/p : I am going home

- **Task:** Machine translation
- **Data:** $\{x_i = source_i, \; y_i = target_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

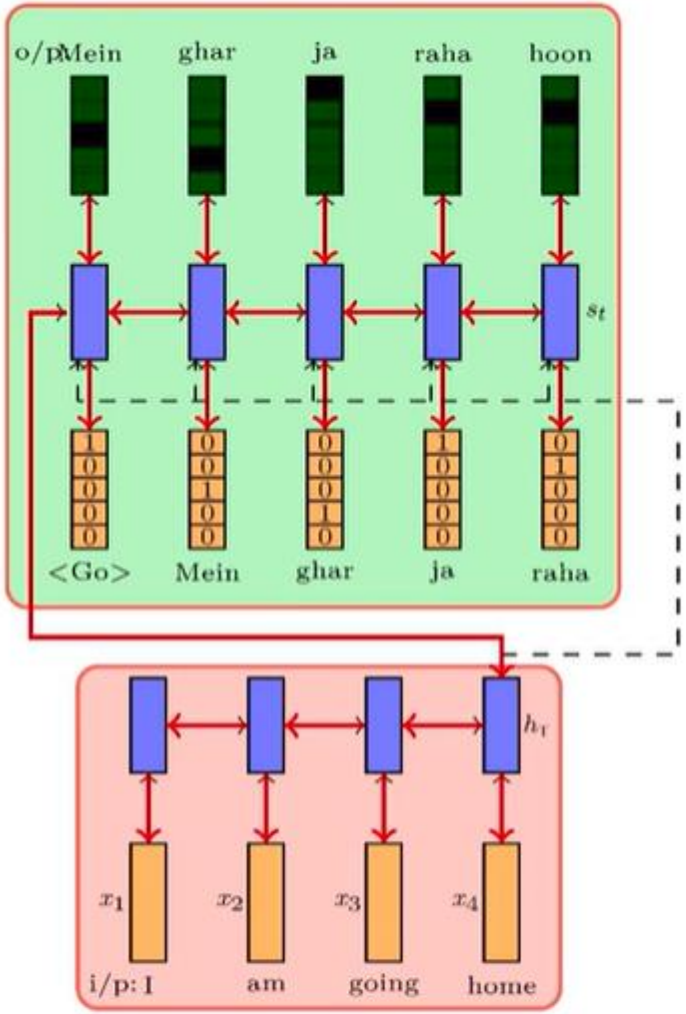
- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with backpropagation

Encoder – Decoder for Machine Translation

o/p : Mein ghar ja raha hoon



i/p : I am going home

- **Task:** Machine translation
- **Data:** $\{x_i = source_i,\ y_i = target_i\}_{i=1}^{N}$
- **Model (Option 2):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$

$$s_t = RNN(s_{t-1}, [h_T, e(\hat{y}_{t-1})])$$

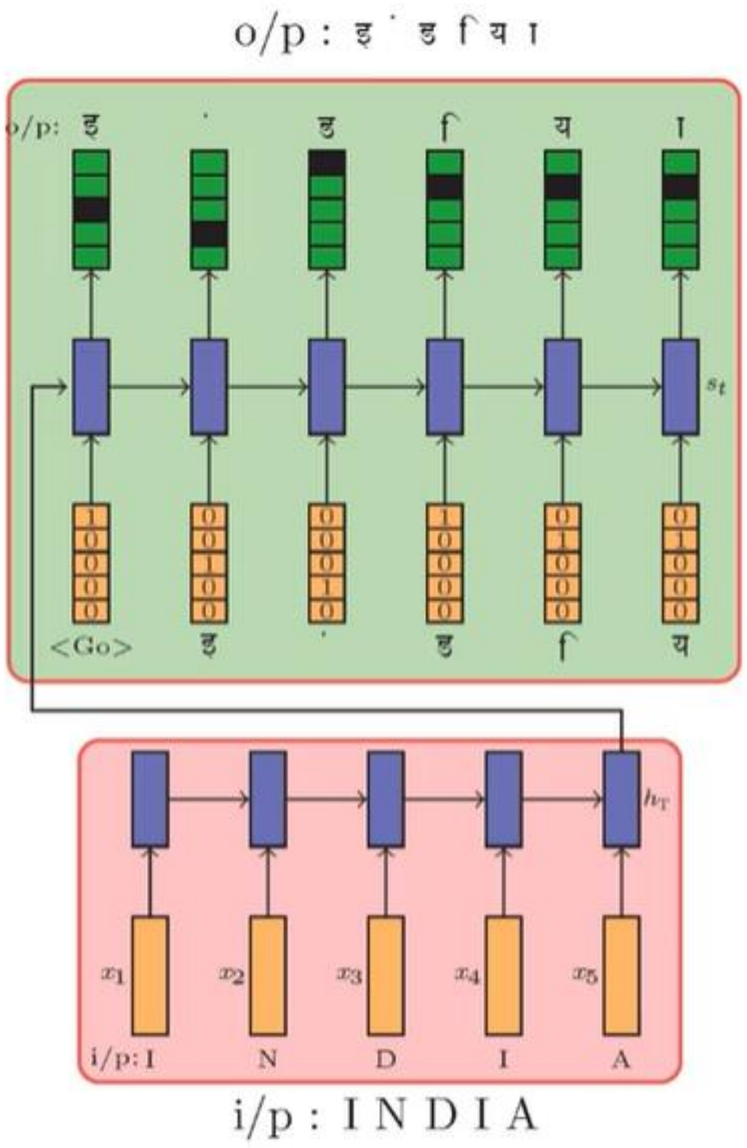
$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with backpropagation

Encoder – Decoder for Machine Transliteration



- **Task:** Transliteration
- **Data:** $\{x_i = srcword_i,\ y_i = tgtword_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \textit{ is length of input})$$

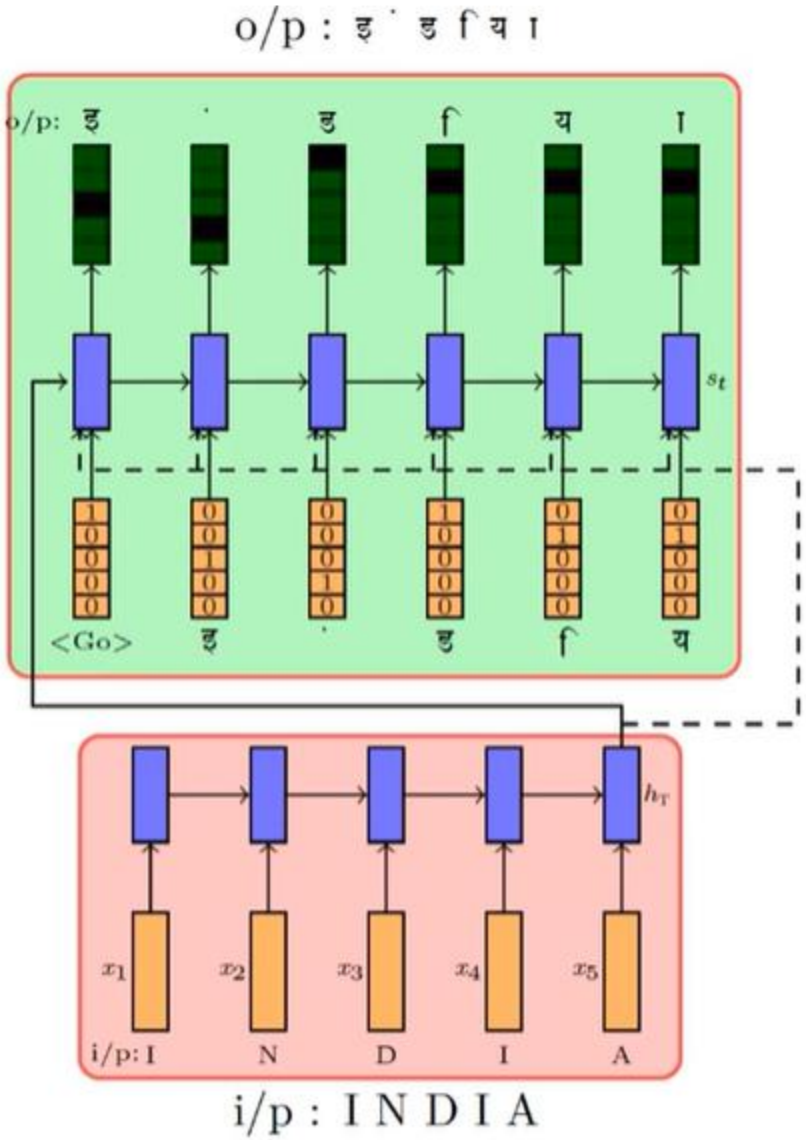
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc},\ b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$

Encoder – Decoder for Machine Transliteration



- **Task:** Transliteration
- **Data:** $\{x_i = srcword_i,\ y_i = tgtword_i\}_{i=1}^{N}$
- **Model (Option 2):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \textit{ is length of input})$$

$$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), h_T])$$

$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with backpropagation

Attention Mechanism

o/p : I am going home

t_1 : [1 0 0 0 0]

t_2 : [0 0 0 0 1]

t_3 : [0 0 0.5 0.5 0]

i/p : Main ghar ja raha hoon

o/p : I am going home

t_1 : [1 0 0 0 0]

t_2 : [0 0 0 0 1]

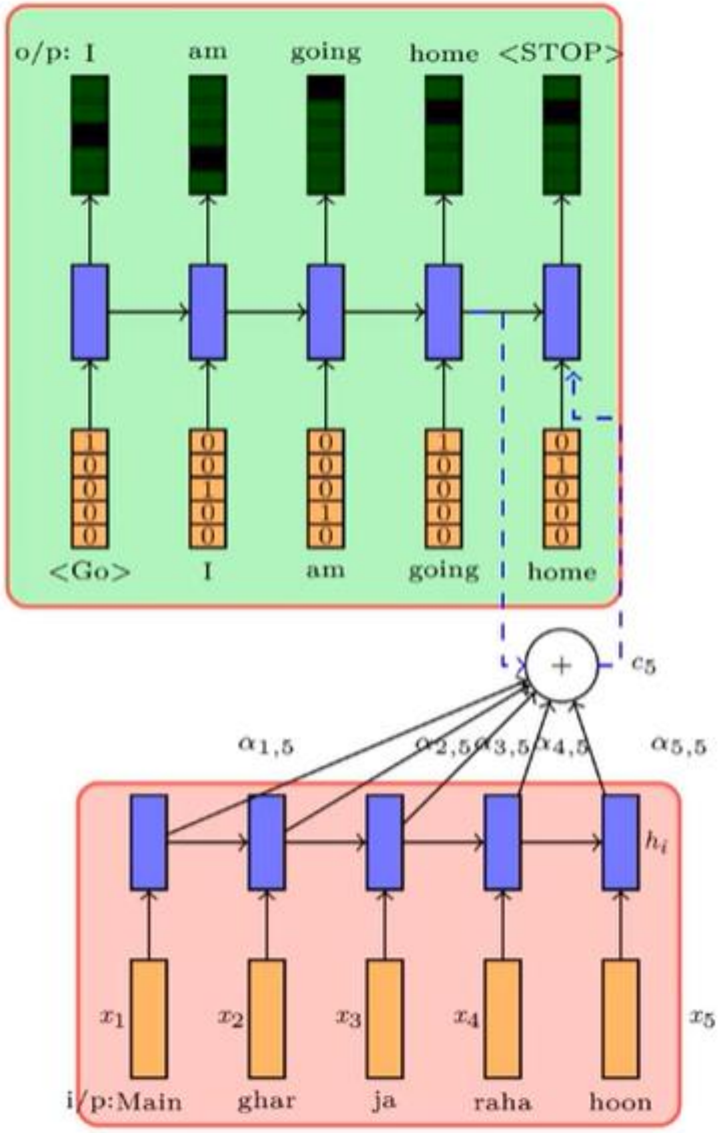
t_3 : [0 0 0.5 0.5 0]

t_4 : [0 1 0 0 0]

i/p : Main ghar ja raha hoon

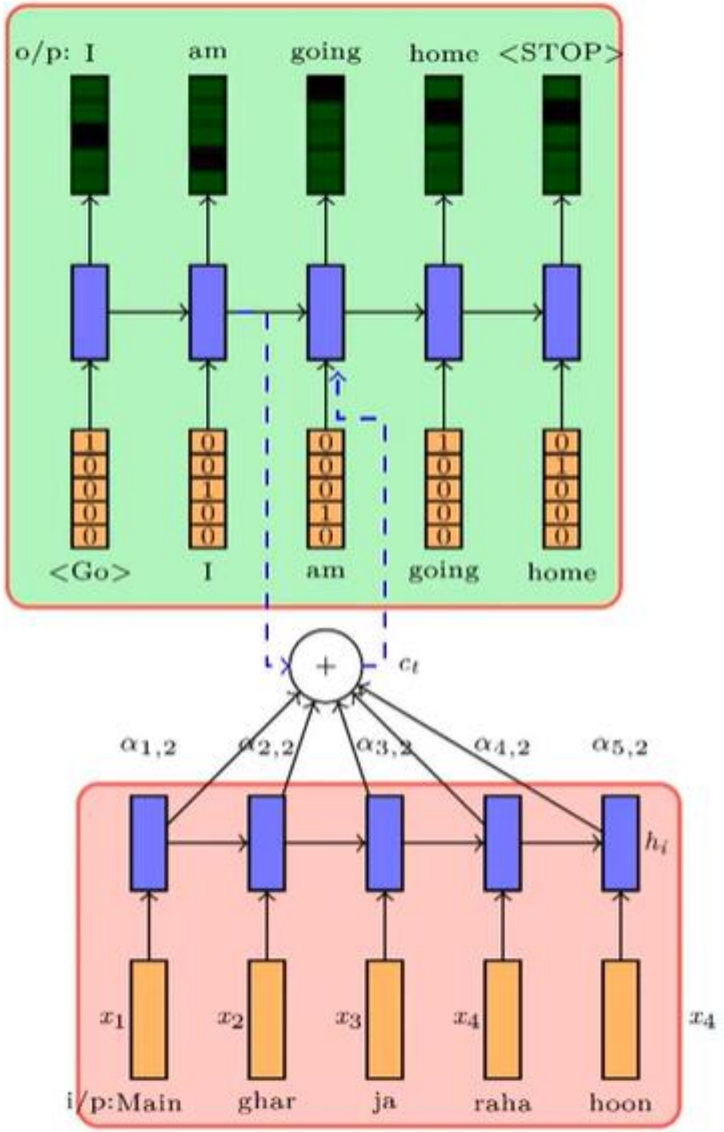
- Humans try to produce each word in the output by focusing only on certain words in the input
- Essentially at each time step we come up with a distribution on the input words
- This distribution tells us how much attention to pay to each input words at each time step
- Ideally, at each time-step we should feed only this relevant information (i.e. encodings of relevant words) to the decoder

Machine Translation with Attention Mechanism



- 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)
- How do we convert this intuition into a model ?

Machine Translation with Attention Mechanism



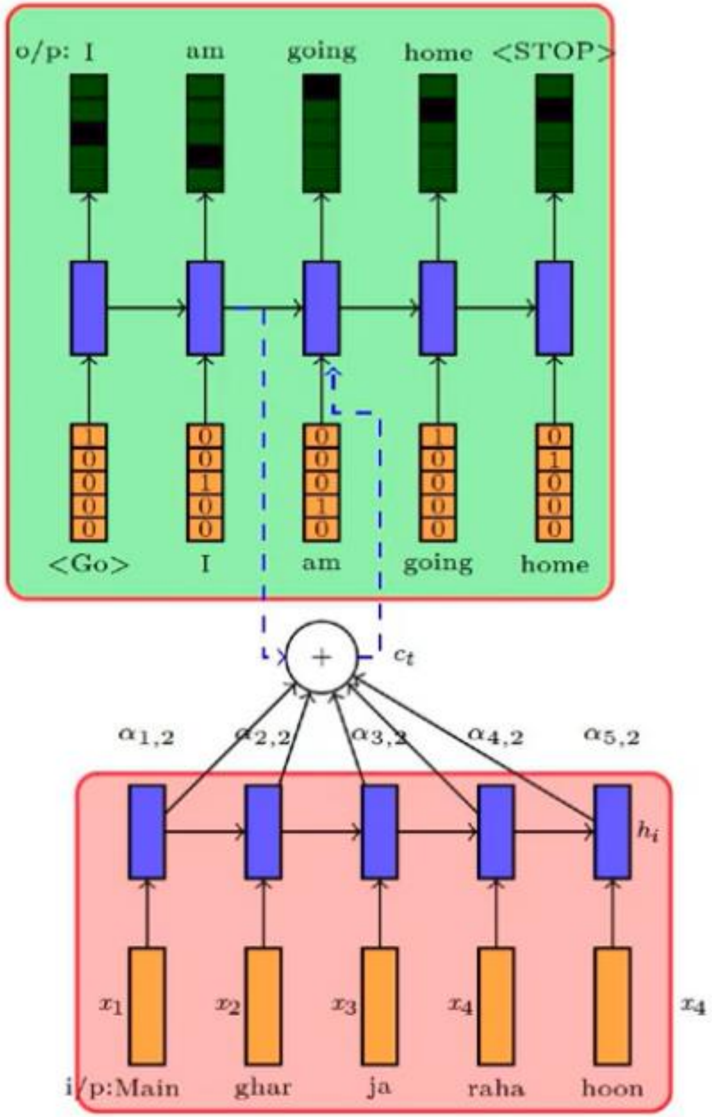
- Of course in practice we will not have this oracle
- The machine will have to learn this from the data
- To enable this we define a function

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

- This quantity captures the importance of the j^{th} input word for decoding the t^{th} output word (we will see the exact form of f_{ATT} later)
- We can normalize these weights by using the softmax function

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

Machine Translation with Attention Mechanism

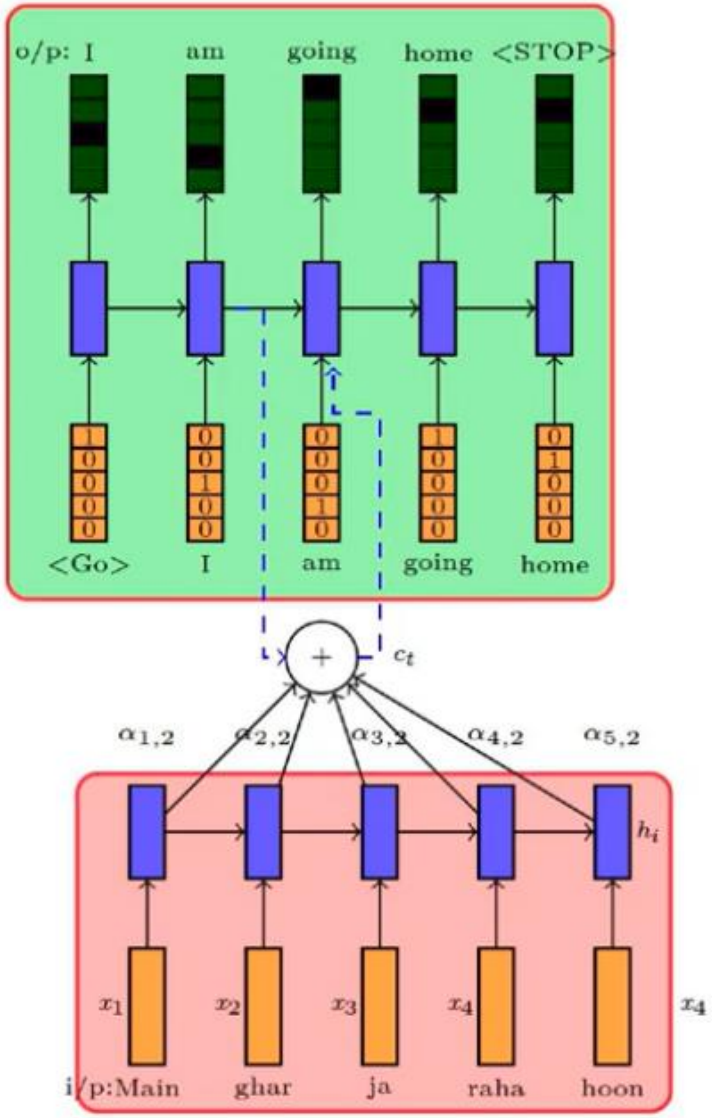


$$\alpha_{jt} = \frac{exp(e_{jt})}{\sum_{j=1}^{M} exp(e_{jt})}$$

$$c_t = \sum_{j=1}^{T} \alpha_{jt} h_j$$

- α_{jt} denotes the probability of focusing on the j^{th} word to produce the t^{th} output word
- We are now trying to learn the α's instead of an oracle informing us about the α's

Machine Translation with Attention Mechanism

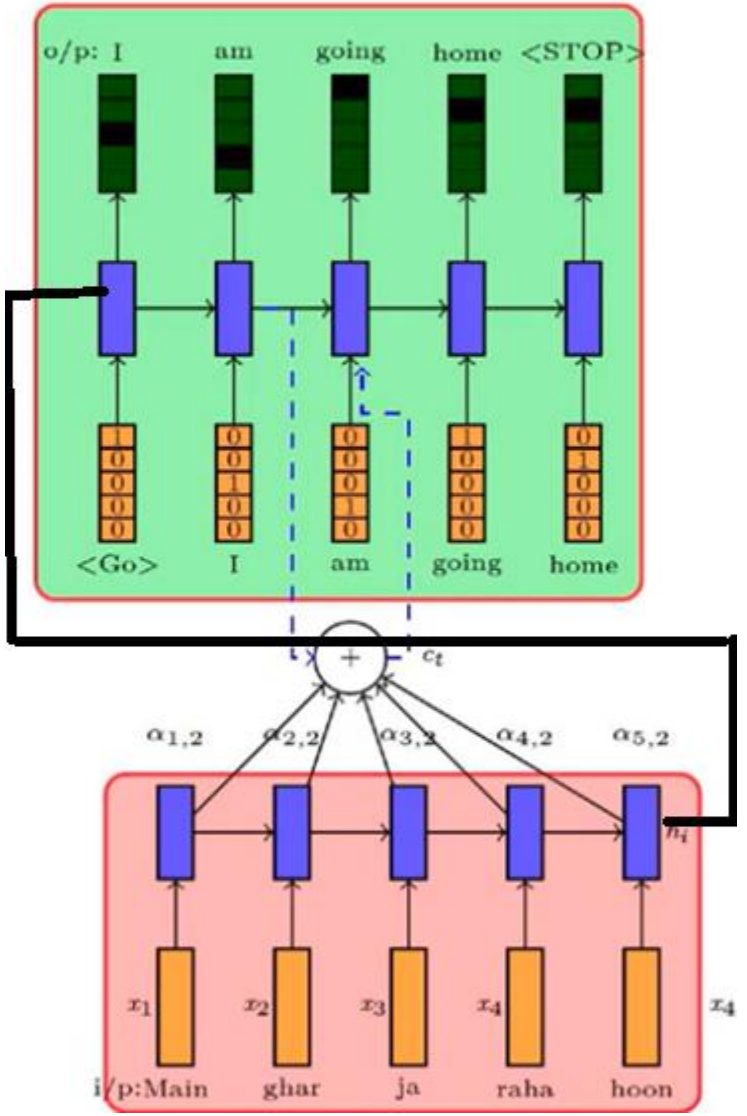


- From now on we will refer to the decoder RNN's state at the t-th timestep as s_t and the encoder RNN's state at the j-th time step as h_j
- Given these new notations, one (among many) possible choice for f_{ATT} is

$$e_{jt} = V_{att}^T \tanh(U_{att} s_{t-1} + W_{att} h_j)$$

- $V_{att} \in \mathbb{R}^d$, $U_{att} \in \mathbb{R}^{d \times d}$, $W_{att} \in \mathbb{R}^{d \times d}$ are additional parameters of the model
- These parameters will be learned along with the other parameters of the encoder and decoder

Machine Translation with Attention Mechanism



- **Task:** Machine Translation
- **Data:** $\{x_i = source_i,\ y_i = target_i\}_{i=1}^{N}$
- **Encoder:**

$$h_t = RNN(h_{t-1}, x_t)$$
$$s_0 = h_T$$

- **Decoder:**

$$e_{jt} = V_{attn}^{T} tanh(U_{attn} h_j + W_{attn} s_{t-1})$$
$$\alpha_{jt} = softmax(e_{jt})$$
$$c_t = \sum_{j=1}^{T} \alpha_{jt} h_j$$
$$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), c_t])$$
$$\ell_t = softmax(V s_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc},\ b,\ U_{attn},\ V_{attn}$
- **Loss** and **Algorithm** remains same

Namah Shivaya