PadhAI: Encoder Decoder Models

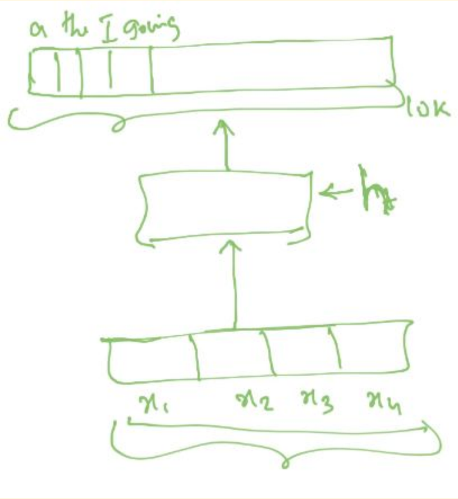
One Fourth Labs

Encoder and Decoder Models.

Let's revisit the task of language modelling.

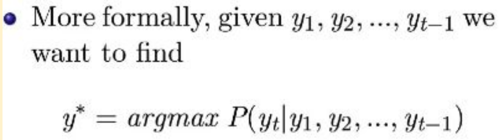
In layman's language, with every given word we try to predict the next word in the sequence.

Suppose there's a total of 10K words in English vocabulary, for each of these words we create a class for each.



In our example, We give an input in which x_1 , x_2, x_3, x_4 are the pre-typed words, they go as an input to some kind of ***neural network***, then we compute some hidden representations(h), which is the encoding **of the given inputs**, using this h, we predict the output, the word which is going to be the next word in the sequence by using the ***Argmax*** and ***Softmax functions***. ***Softmax*** is used to predict the probability for each word in the vocabulary, how probable it is, for a word to be the next word in the sequence. It creates a **probability distribution**. ***Argmax*** picks up the word with **maximum probability** from the whole probability distribution.

In this example, encoding is the process in which we pass an input through a network and produce a vector h, I.e., input is encoded into a vector h, and based on h we compute the final output.

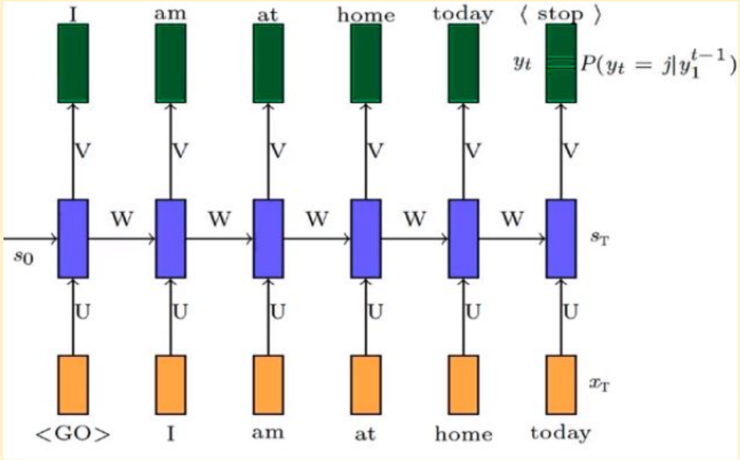


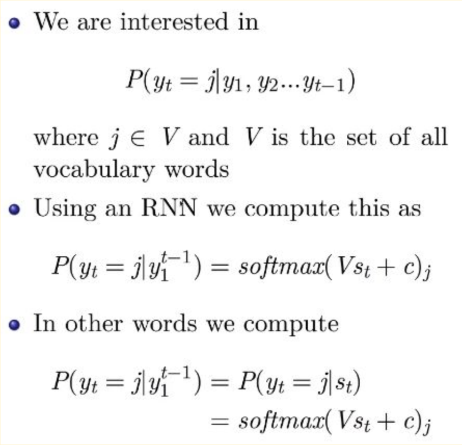
- More formally, given $y_1, y_2, ..., y_{t-1}$ we want to find

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

Where y^* is the predicted word from the whole vocabulary, for the given inputs y_1 , y_2 , ..., y_{t-1} which are the outputs of previous time steps.

Using RNNs as for language modelling.




- We are interested in

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

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

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

We calculate the value of state at each time step using the formula st = Wst-1 + Ux1 + b and using st we calculate the output (y). for now assume W, U, V, b and c are already learnt parameters using the training data. At time step t, we calculate the output yt using the previous outputs (y1, y2, y3, ..., yt-1). Previous outputs are inputs for our model. <Go> and <Stop> represents <sos> and <eos> respectively.

Introducing Encoder and Decoder model.

The main focus of the RNNs is to compute the value of state at the given time step considering the current input and all the previcompunputs, the job of the RNN is to compute the state vector. Further if we want to implement any other model such as CNN, FNN or any other function

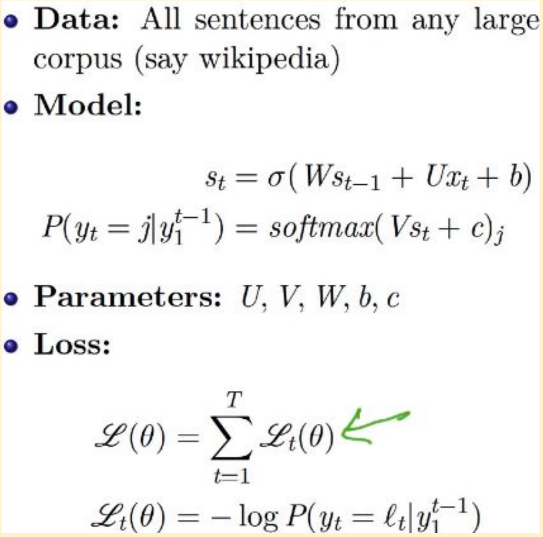
on the produced states to produce final output, it all depends upon the required task.

- **In the above task, Encoder** is a **Recurrent Neural Network** because it encodes the input into a state vector which is the fed to output layer. **Decoder** is a **model of our choice**(FNN, CNN or any other model) because it decodes the encoded input to calculate output. Choice of encoding and decoding models depends on the task.

Connecting Encoder and Decoder model to jars.

Task: Task is a sequence prediction task, in which we need to predict the output word at a given time step using a sequential input (current input and all the previous inputs).

Data: Data is all the large sentences from a large corpus, when we train a sequential model we want the maximum probability distribution over the true output and to achieve this condition we need to reduce the loss at every epoch by updating parameters (backpropagation), for that we need true long sequences , act as true label, y, which helps to calculate and reduce loss.



- **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:**

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

Loss is the sum of the cross entropies at every time step.

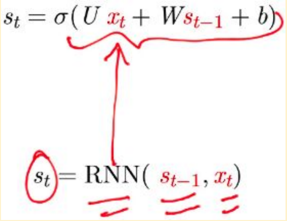
PadhAI: Encoder Decoder Models

One Fourth Labs

A compact notation for RNNs, LSTMs and GRUs.

Representing RNNs.

Compact notation:

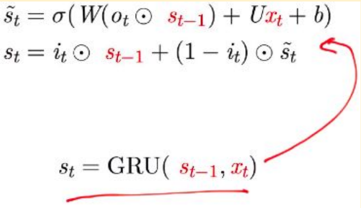


$$s_t = \sigma(U x_t + W s_{t-1} + b)$$

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

Representing GRUs.

Compact notation:



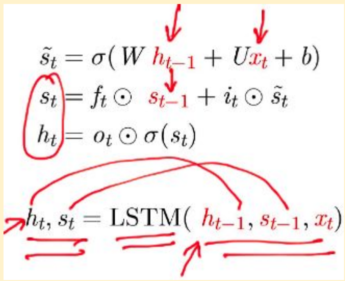
$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$

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

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

Representing LSTMs.

Compact notation:



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

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

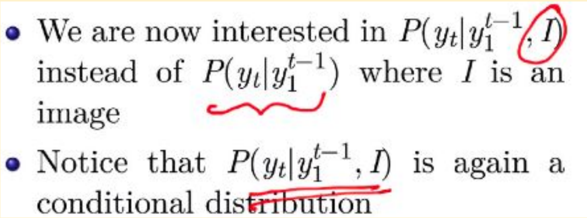
$$h_t = o_t \odot \sigma(s_t)$$

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

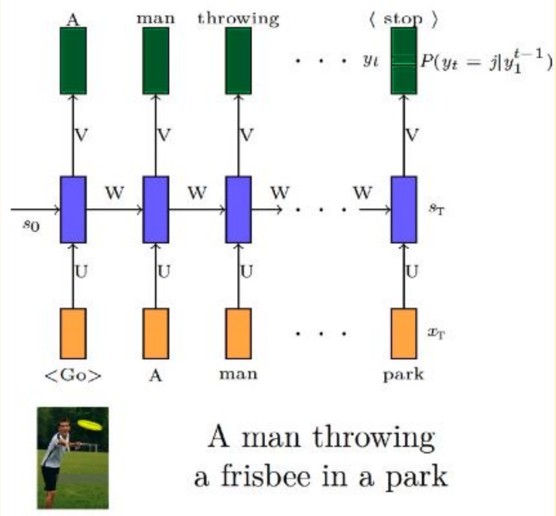
Encoder Decoder Model for image captioning.

So far we've seen how to model the conditional probability distribution, informally we have seen how to generate a sentence, given previous words.

Now we want to generate a sentence given an image.



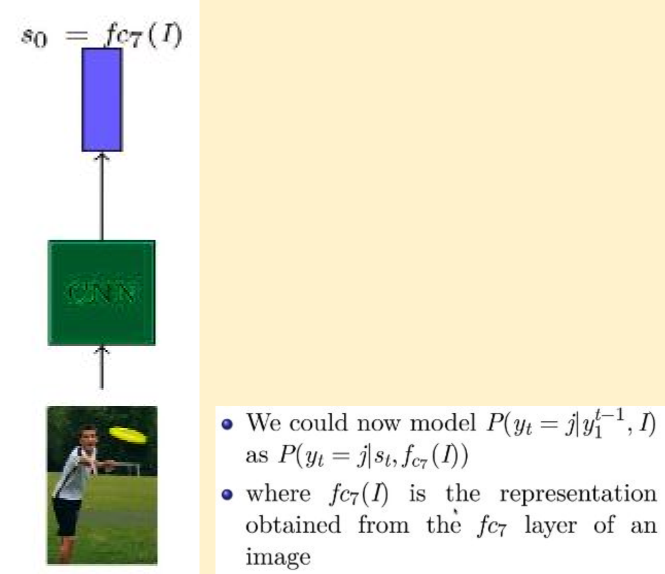
- We are now interested in $P(y_t|y_1^{t-1}, I)$ instead of $P(y_t|y_1^{t-1})$ where I is an image
- Notice that $P(y_t|y_1^{t-1}, I)$ is again a conditional distribution



Suppose we are given an image in which a man is throwing a frisbee, the expected description of the image should be **"A man throwing a frisbee in the park." ,** so given an image we need to generate a sentence which describes the image to the closest. The approach we will follow is we generate a first word, based on the image and the 1st word we generate 2nd word and so on. We now include image as well, along with the current and the previous inputs.

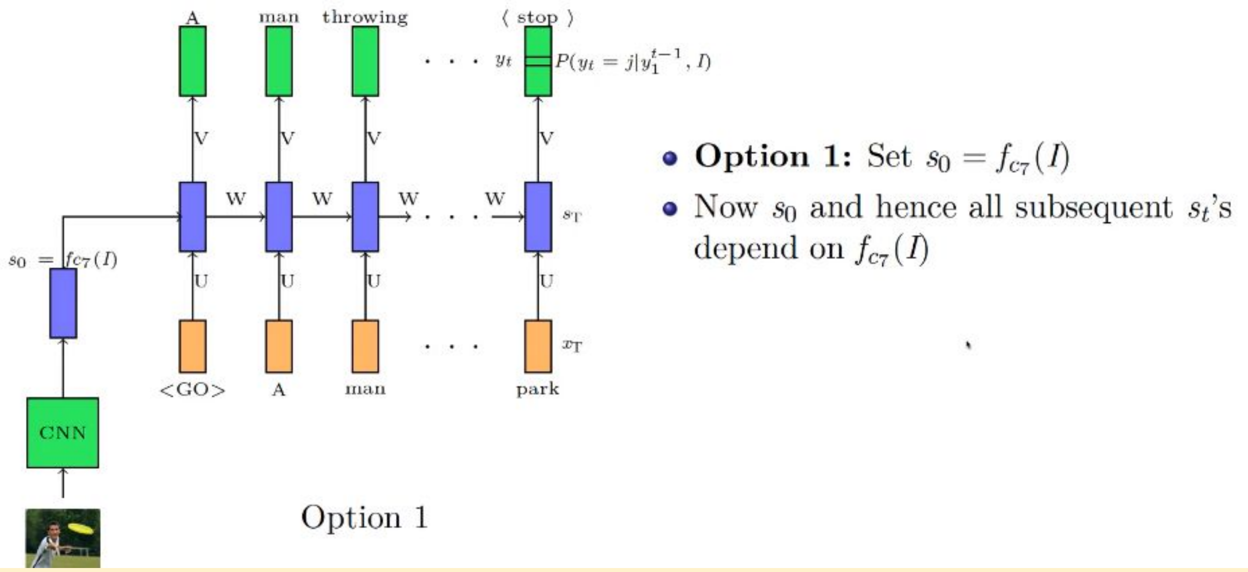
We have already learnt about sequence generation model, the only difference here is the inclusion of an image.

Previously as we encoded all the input words into a vector, now we will also encode the input image into a vector(using CNN) along with the words.



- We could now model $P(y_t = j | y_1^{t-1}, I)$ as $P(y_t = j | s_t, f_{c_7}(I))$
- where $fc_7(I)$ is the representation obtained from the fc_7 layer of an image

There are many ways of making **P(**$y_t = j$**)** conditional on f_{c7} **(I). *below mentioned are the few options.***



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

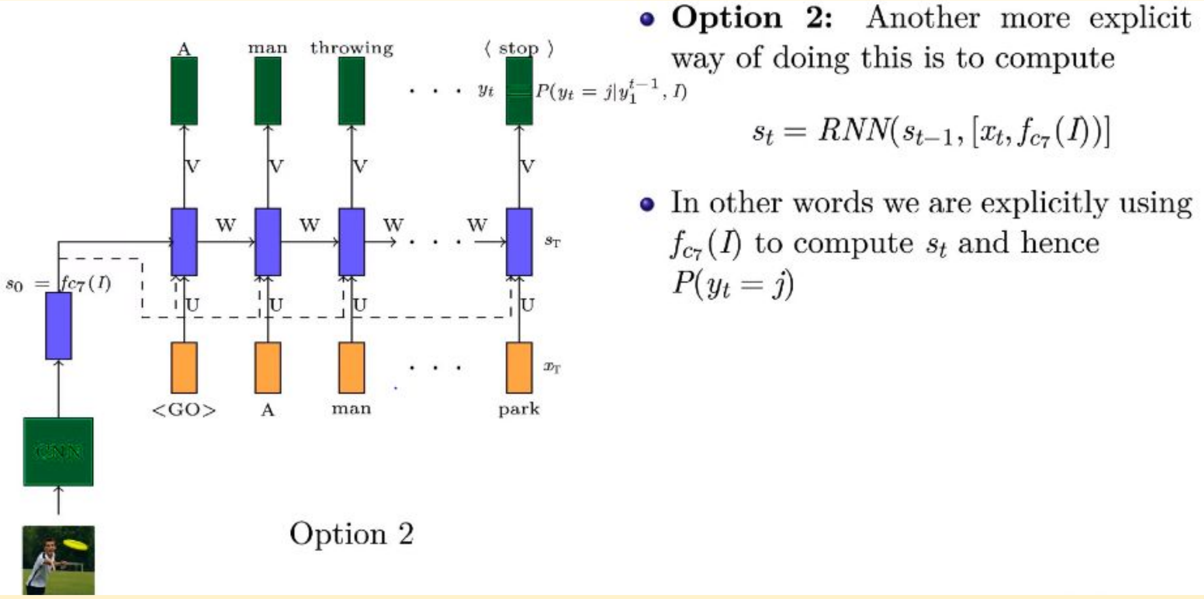
$$s_2 = \sigma(W\underset{\downarrow \atop \text{CNN}}{s_1} + U\underset{\downarrow \atop \text{A}}{x_2} + b)$$

$$y_2 = (Vs_2 + c)$$

Here s0 is the starting point, whatever vector we have calculated for the image using CNN is connected to s0 and then fed to the network further.

As when we calculated s2 as a function of s1, which in turn **implicitly** depends on s0. This makes every state in the network dependent on the image state s0 **implicitly**.

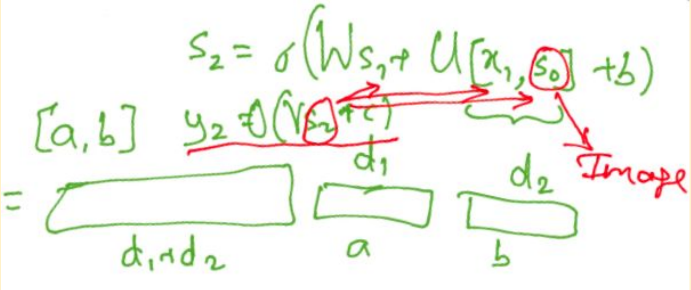
Other option does the same by making states dependent on the state s0 **explicitly**.



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

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

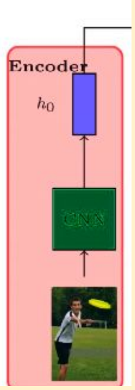
- In other words we are explicitly using $f_{c_7}(I)$ to compute s_t and hence $P(y_t = j)$



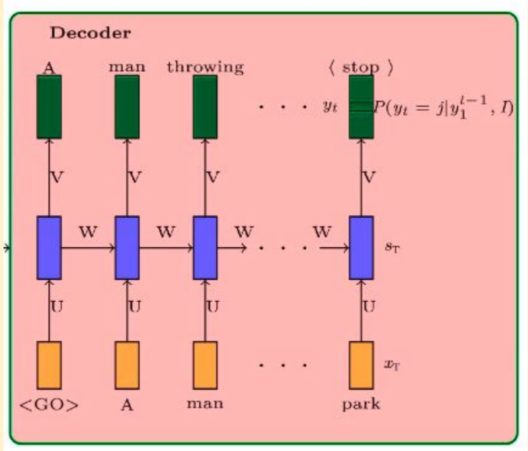
We feed the image explicitly or directly to every state/ time step. Now, each state at each time step (st) not only depends on previous time step(st-1) but also on s0.

Let's look at the full architecture.

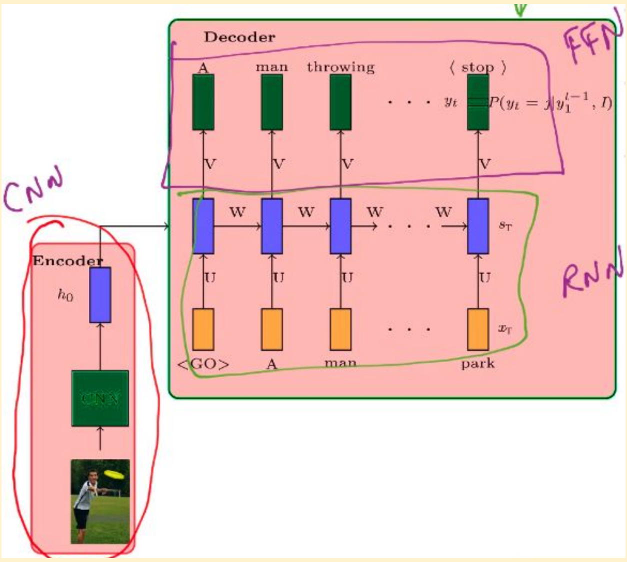
A CNN is first used to encode the image



Then a RNN is used to decode(generate) a sentence from the encoding.



Combining our modules to build the Encoder Decoder Architecture.



Six jars for image captioning

Task : Our task is **Image captioning.**

Automatically generating textual description from an artificial system is the task of image captioning.

The task is straightforward – the generated output is expected to describe in a single sentence what is shown in the image – the objects present, their properties, the actions being performed and the interaction between the objects, etc. But to replicate this behaviour in an artificial system is a huge task, as with any other image processing problem and hence the use of complex and advanced techniques such as Deep Learning to solve the task.

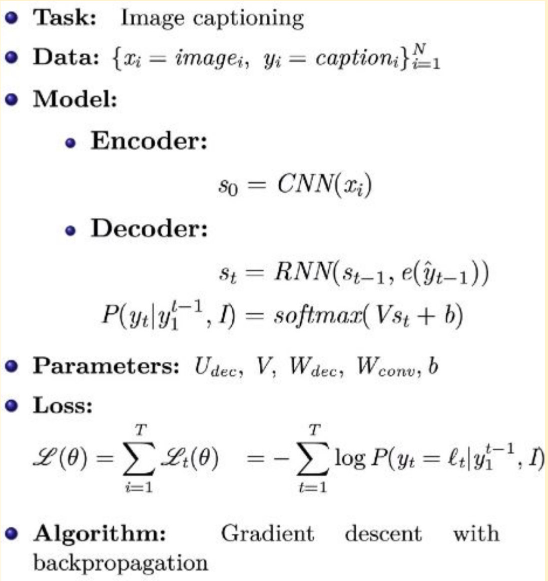
Data: A dataset of **images** and **captions** describing the image. X as a dataset of images and Y as the dataset of captions.

Model: We should be able to write an output as a function of the input.

Our model consists of two parts

1) Encoder.

2) Decoder.



- **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

Evaluation : It depends on the task we want to perform.

Note: In decoder **e** symbolises that it is a **one hot encoder.**

*Loss is the **sum of the cross entropies at every time step.***

While updating the parameters, we try to reduce the loss by setting the parameters with most appropriate values using backpropagation.

Encoder Decoder for machine translation.

Task: Machine translation.

Suppose we are given an input sentence in Hindi, say “Mai ghar jaa raha hoon" and we need to translate it into an English sentence.

The task here is a bit complicated as when we translate a sentence of one language to another, the positioning of the words in a sentence differs with a variable degree. We cannot just translate each word in a sentence of a particular language to another language and place them in the same order, the sentence that we have translated to other language might not make any sense.

So, the task is to translate a given sentence to a sentence of another language which makes not only sense but also delivers the right information.

Data: A pair of sequences which consists of original language sentences and corresponding translated sentences of the language in which they are to be translated.

Model: We just need to keep in mind that the output should be a continuous function of input.

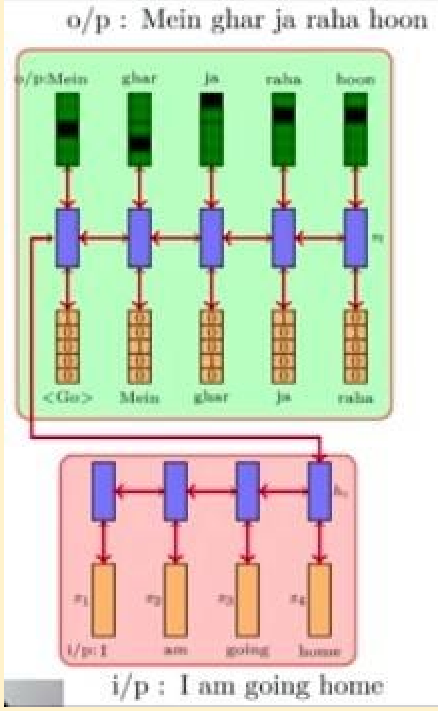
We can face problem of vanishing and exploding gradients if we use RNNs, we can use LSTMs to tackle this problem.

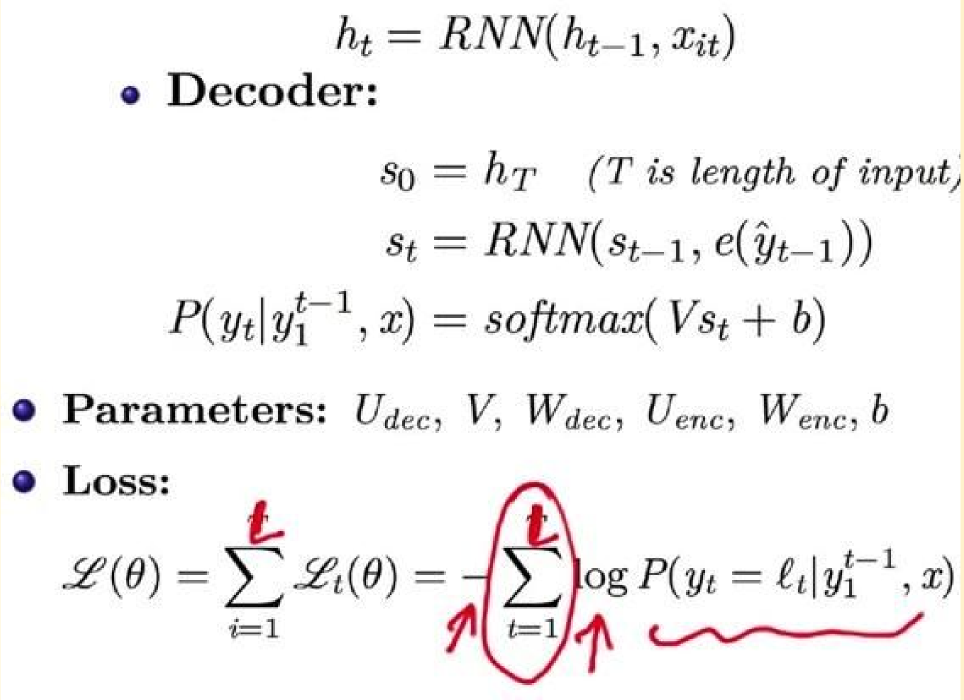
We have two options.

Option 1:

We feed input encoded representation implicitly (only at the beginning of the decoder.)

Note: for encoder we have changed the notation, st to ht.





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

- **Decoder:**

$$s_0 = h_T \quad (T\ is\ length\ of\ input$$

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

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

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

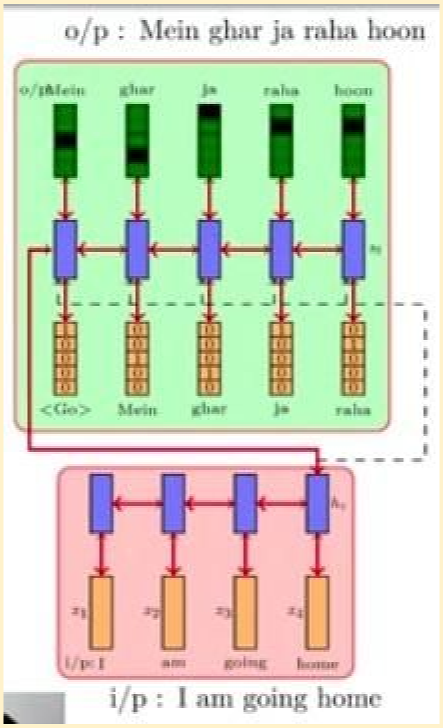
$$\mathscr{L}(\theta) = \sum_{i=1}^{t} \mathscr{L}_t(\theta) = -\boxed{\sum_{t=1}^{t}} \log \underline{P(y_t = \ell_t|y_1^{t-1}, x)}$$

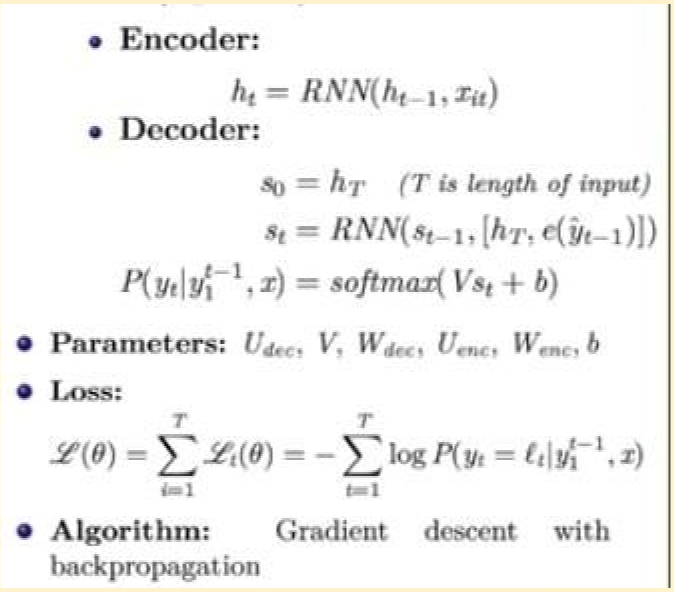
Algorithm: Gradient descent with backpropagation

Option 2:

We feed input encoded representation explicitly (at every time step of the decoder.)

Note: for encoder we have changed the notation, st to ht, e denotes one hot vector.





- **Encoder:**

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

- **Decoder:**

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

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

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

- **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 model for transliteration

Task: Transliteration

Word to word translation, i.e., we translate a word written in one language to another language, instead of a whole sentence as we did in translation. It is an easier task than translation.

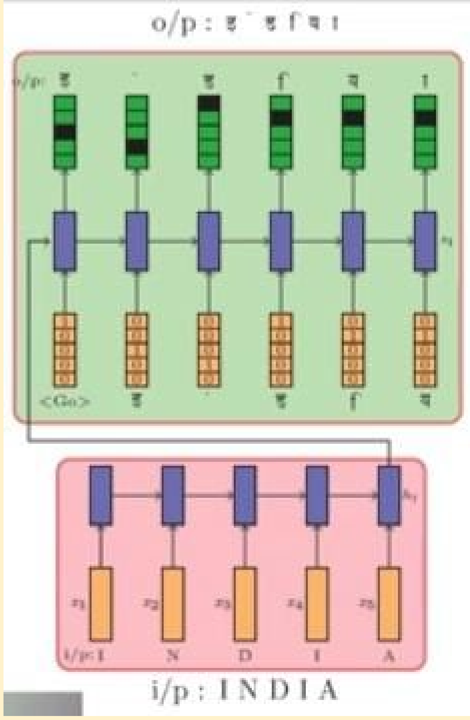
Data: A pair of words which consists of original language word and corresponding translated word of the language in which it is to be translated.

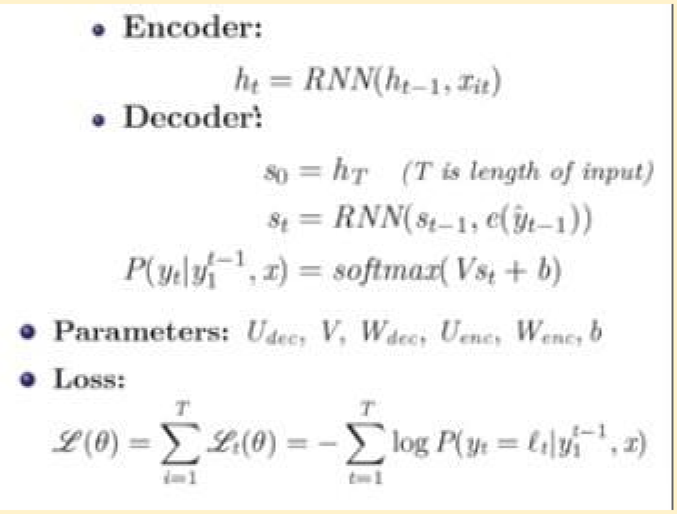
Model:

Option 1:

We feed input encoded representation implicitly (only at the beginning of the decoder.)

Note: for encoder we have changed the notation, st to ht.





- **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:**

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

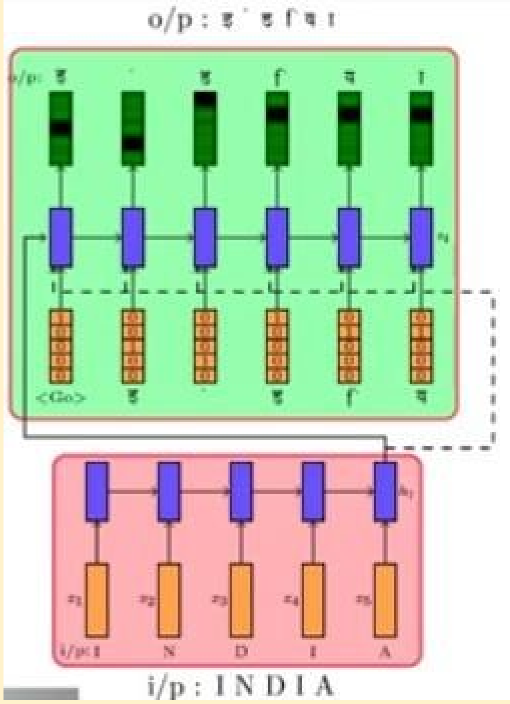
Option 2:

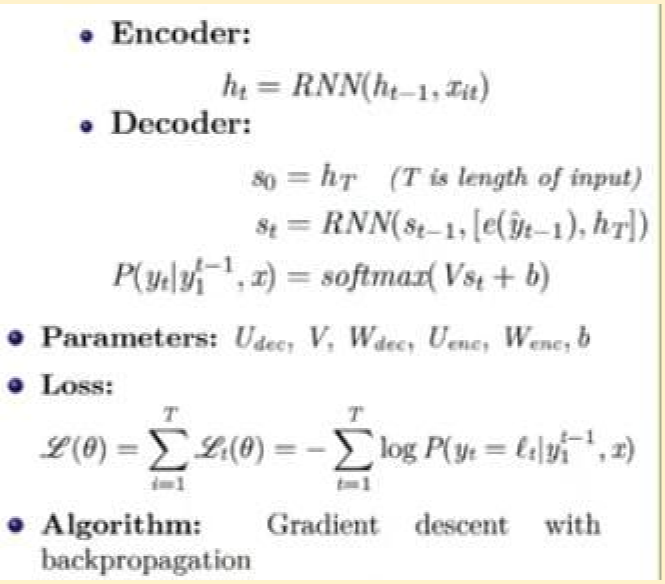
We feed input encoded representation explicitly (at every time step of the decoder.)

Note: for encoder we have changed the notation, st to ht, e denotes one hot vector.

PadhAI: Encoder Decoder Models

One Fourth Labs





- **Encoder:**

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

- **Decoder:**

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

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

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

- **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