Encoder - Decoder Model

In the LSTM chapter, we have gone through the overview of the encoder-decoder models. One of the biggest applications of the sequence-sequence models is in Machine Translation.

Machine Translation

Let us assume we are given a sequence of words $x^1, x^2, x^3, ..., x^t$ as the input, and the responses for this sequence are $y^1, y^2, y^3, …., y^k$. (Here $t \neq k$). It is not always mandatory for 't' to be equal to 'k'.

Here the input sequence is denoted as 'x_i' and the output sequence is denoted as 'y_i'.

$x_i = <x_i^1, x_i^2, x_i^3, ……, x_i^t>$

$y_i = <y_i^1, y_i^2, y_i^3, ……., y,^k>$

Image Captioning

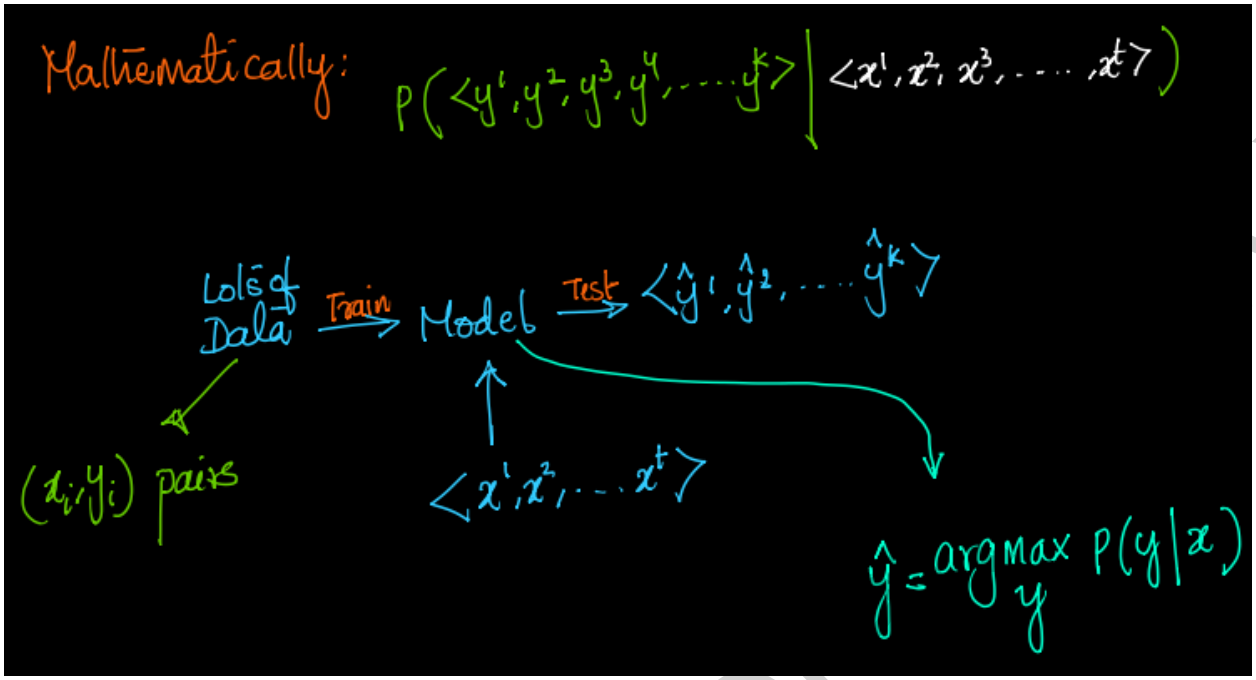
Another application of Sequence-Sequence models is Image Captioning. Let us assume we are giving an image as the input, and the output response would be a sequence of words.

So for a sequence-sequence model, we have to pass pairs of the input sequences and the corresponding actual output values, and obtain the response sequences. Mathematically, it can be written as

$P(y^1, y^2, y^3, …, y^k|x^1, x^2, x^3, …., x^t)$

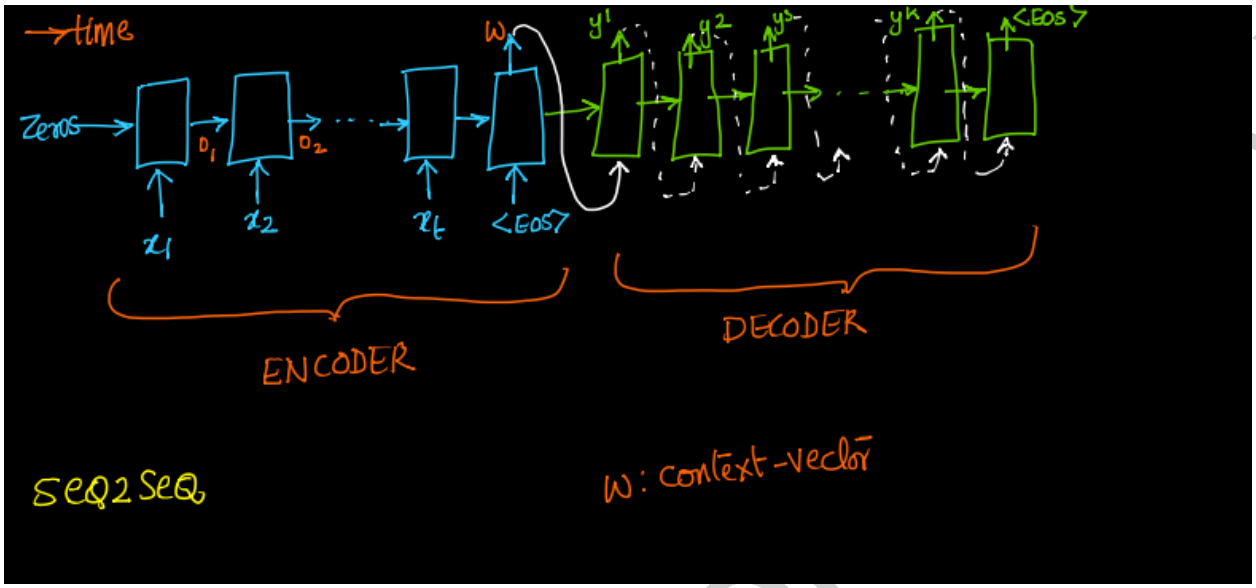
In general, for any Machine learning model 'M', if we pass (x_i, y_i) as the input, it predicts the value of 'y_i^'. So here when we have multiple words as the labels, when query point 'x_q' is passed as an input, the model computes the probabilities for each of the words in the output sequence (ie., $y^{(1)}, y^{(2)}, y^{(3)}, …, y^{(k)}$), whichever words gets the highest probability, that would be predicted.

$y^ = \text{arg-max}_y P(y|x)$



Given our input, the model is finding the probability given ‘x’. It checks which word/value in the output sequence is most likely to occur and returns it as the output. But we need a huge amount of data to train this kind of model.

Encoder-Decoder model - Intuition



In this architecture, the input we are giving is the output of the previous time step. The output at the current time step would become the input for the next time step. At the end, we add an end of the string layer, so that it indicates to the model that the input phase has ended. From here we build a vector as an output for this encoder, and let us call it a **context vector** and let us denote it with 'w'.

This context vector is passed as an input to the first cell in the decoder network. In the decoder section, the cell produces an output at the end of every time step returns it, and also generates one more output which is passed as the input to the next time step. The outputs at the end of every time step in the decoder section are denoted as $y^{(1)\wedge}$, $y^{(2)\wedge}$, $y^{(3)\wedge}$, …, $y^{(k)\wedge}$. After obtaining these outputs for every time step, we have to compute the softmax loss (ie., $L(y_i, y_i\wedge)$). Similarly for every data point, we have to compute the softmax loss value, and using the backpropagation through time algorithm, we have to train the model and update the weights in the network, such that the final softmax loss obtained across the entire training dataset is minimum.

In the first research paper, it was mention that the output of the encoder section (ie., the context word) should be passed as an input only to the first cell of the decoder section as an input, whereas the second

research paper says that the context vector should be passed as an input to every cell in the decoder section as an input. This approach seems to perform better than the former approach only in some cases. But here is a problem, as the LSTM cell doesn't accept two inputs at a time step. So in this second research paper, they've designed a new modified LSTM unit such that it accepts two inputs.

In the third research paper, we have to pass an image as the input and obtain a sequence as an output. Here we first have to pass the image through a convolutional neural network, and in the last, we are removing the softmax layer, and the obtained weight vector is called an **Image Context Vector**. This Image Context Vector is passed as an input to the first cell along with the input at the current time step. Even here, every layer takes the output of the previous time step and the input of the current time step as an input to the cell. Even the decoder section has an end of the string layer at the end which means it stops accepting the inputs.

Applications

1) Machine Translation
2) Email Auto Reply and Smart Compose
3) Image Descriptions
4) Detection of the code errors

Reference Links

a) Encoder-Decoder Models

Research Paper 1:
https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

Research Paper 2:
https://aclanthology.org/D14-1179.pdf

b) PPT Link:
https://drive.google.com/file/d/1vSjFmCbCl5raQm9BcuoWJtb9XruRbHkg/view

c) Blogs for Applications
https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

https://ai.googleblog.com/2018/05/smart-compose-using-neural-networks-to.html

https://medium.com/@martin.monperrus/sequence-to-sequence-learning-program-repair-e39dc5c0119b

https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8?gi=55b95197f75a