Encoder-Decoder Seq2Seq Models, Clearly Explained!!

A step-by-step guide to understanding Encoder-Decoder Sequence-to-Sequence models in detail!

Kriz Moses · Follow

Published in Analytics Vidhya · 17 min read · Mar 12, 2021

Image by CoxinhaFotos from Pixabay

Table of Contents

Introduction

> *The traditional Deep Neural Networks (DNNs) are powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition and visual object recognition. But they can only be used for large labeled data where inputs and targets can be sensibly encoded with vectors of fixed dimensionality. They cannot be used to map sequences-to-sequences (for eg. machine translation)*
>
> *Sequence to Sequence Learning with Neural Networks [1]*

This was the motivation behind coming up with an architecture that can solve general sequence-to-sequence problems and so encoder-decoder models were born.

In this article, I aim to explain the encoder-decoder sequence-to-sequence models in detail and help build your intuition behind its working. For this, I have taken a step-by-step approach and tried to explain the concepts **visually**. I will mainly refer to the seminal research paper Sequence to Sequence Learning with Neural Networks by Ilya Sutskever, et al which implemented encoder-decoder models for neural machine translation. In the end, I will also give the intuition behind a modified version of encoder-decoder used for image captioning in the paper Deep Visual-Semantic Alignments for Generating Image Descriptions by Andrej Karpathy, Li Fei-Fei, et al

Motivation

The Encoder-Decoder architecture is relatively new and had been adopted as the core technology inside Google's translate service in late 2016. It forms the basis for advanced sequence-to-sequence models like Attention models, GTP Models, Transformers, and BERT. Hence, understanding it can be very crucial before moving onto advanced mechanisms.

Prerequisites

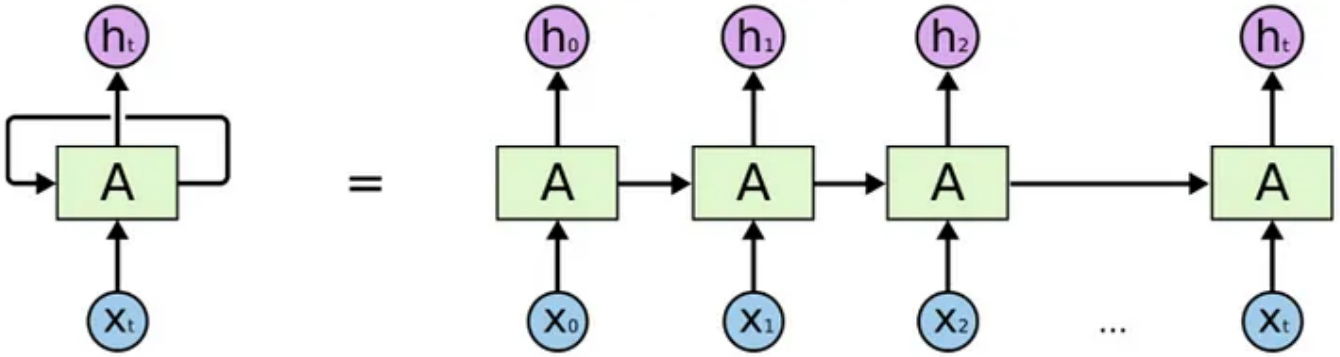
This post assumes that you are familiar with neural networks and, in particular, the working of RNNs/LSTMs. If you are unsure, then I strongly encourage you to check the blogs Illustrated Guide to Recurrent Neural Networks by Michael Phi *and* Understanding LSTM Networks by Christopher Olah.

Sequence Modelling Problems

Sequence Modelling problems refer to the problems where either the input and/or the output is a sequence of data (words, letters…etc.)

Consider a very simple problem of predicting whether a movie review is positive or negative. Here our input is a sequence of words and output is a single number between 0 and 1. If we used traditional DNNs, then we would typically have to encode our input text into a vector of fixed length using techniques like BOW, Word2Vec, etc. But note that here the sequence of words is not preserved and hence when we feed our input vector into the model, it has no idea about the order of words and thus it is missing a very important piece of information about the input.

Thus to solve this issue, RNNs came into the picture. In essence, for any input $X = (x_0, x_1, x_2, \ldots x_t)$ with a variable number of features, at each time-step, an RNN cell takes an item/token $\mathbf{x_t}$ as input and produces an output $\mathbf{h_t}$ while passing some information onto the next time-step. These outputs can be used according to the problem at hand.

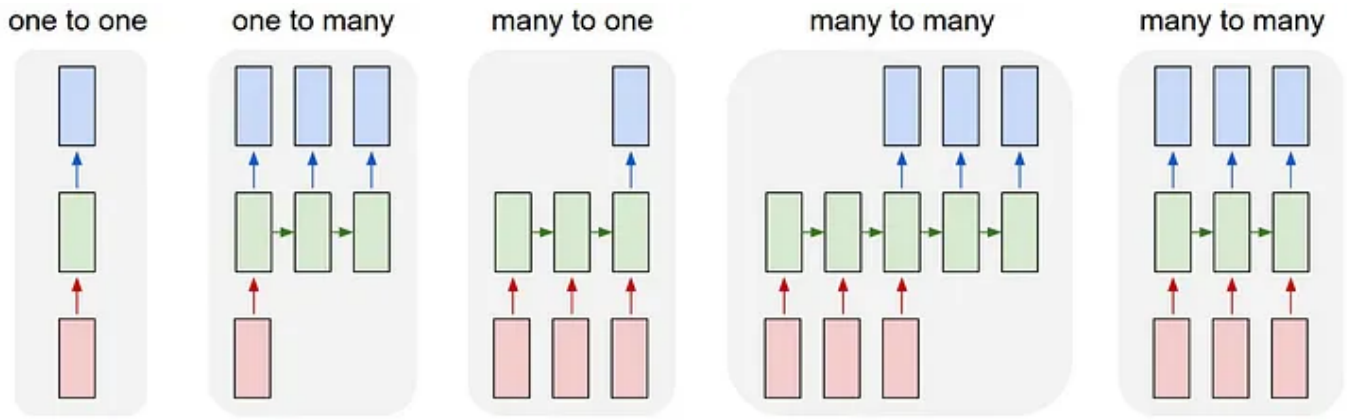


An unrolled recurrent neural network.

Understanding LSTM Networks by Christopher Olah.

The movie review prediction problem is an example of a very basic sequence problem called many to one prediction. There are different types of sequence problems for which modified versions of this RNN architecture are used.

Sequence problems can be broadly classified into the following categories:



The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy

> *Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right:* ***(1)*** *Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).* ***(2)*** *Sequence output (e.g. image captioning takes an image and outputs a sentence of words).* ***(3)*** *Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).* ***(4)*** *Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).* ***(5)*** *Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.*

> *— Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks [5]*

Sequence-to-Sequence Problems

Sequence-to-Sequence (Seq2Seq) problems is a special class of Sequence Modelling Problems in which both, the input and the output is a sequence. Encoder-Decoder models were originally built to solve such Seq2Seq problems. In this post, I will be using a many-to-many type problem of Neural Machine Translation (NMT) as a running example. Later, we also see how they can also be used for image captioning (one-to-many example).

The Architecture of Encoder-Decoder models

My main goal is to help you understand the architecture used in the paper Sequence to Sequence Learning with Neural Networks by Ilya Sutskever, et al. This was one of the first papers to introduce the Encoder-Decoder model for machine translation and more generally sequence-to-sequence models. The model was applied to English to French translation.

NOTE: To get to the final model, I will attempt to simplify things and introduce the concepts one by one to hopefully make it easier for people to understand without too much in-depth knowledge of the subject matter, filling in the gaps as and when needed.

The Neural Machine Translation Problem

To help you understand better, I will be taking Neural Machine Translation as a running example throughout this post. In neural machine translation, the input is a series of words, processed one after another. The output is, likewise, a series of words.

Task: Predict the French translation for every English sentence used as input.

For simplicity let's consider that there is only a single sentence in our data corpus. So let our data be:

- **Input: English sentence: "nice to meet you"**
- **Output: French translation: "ravi de vous rencontrer"**

Terms Used

To avoid any confusion:

- I will refer to the Input sentence **"nice to meet you"** as **X/input-sequence**
- I will refer to the output sentence **"ravi de vous rencontrer"** as **Y_true/target-sequence** → This is what we want our model to predict (the ground truth).
- I will refer to the predicted output sentence of the model as **Y_pred/predicted-sequence**
- The individual words of the English and French sentence are referred to as **tokens**

Hence, given the input-sequence "nice to meet you", we want our model to predict the target-sequence/Y_true i.e "ravi de vous rencontrer"

High-Level Overview

At a very high level, an encoder-decoder model can be thought of as two blocks, the encoder and the decoder connected by a vector which we will refer to as the 'context vector'.

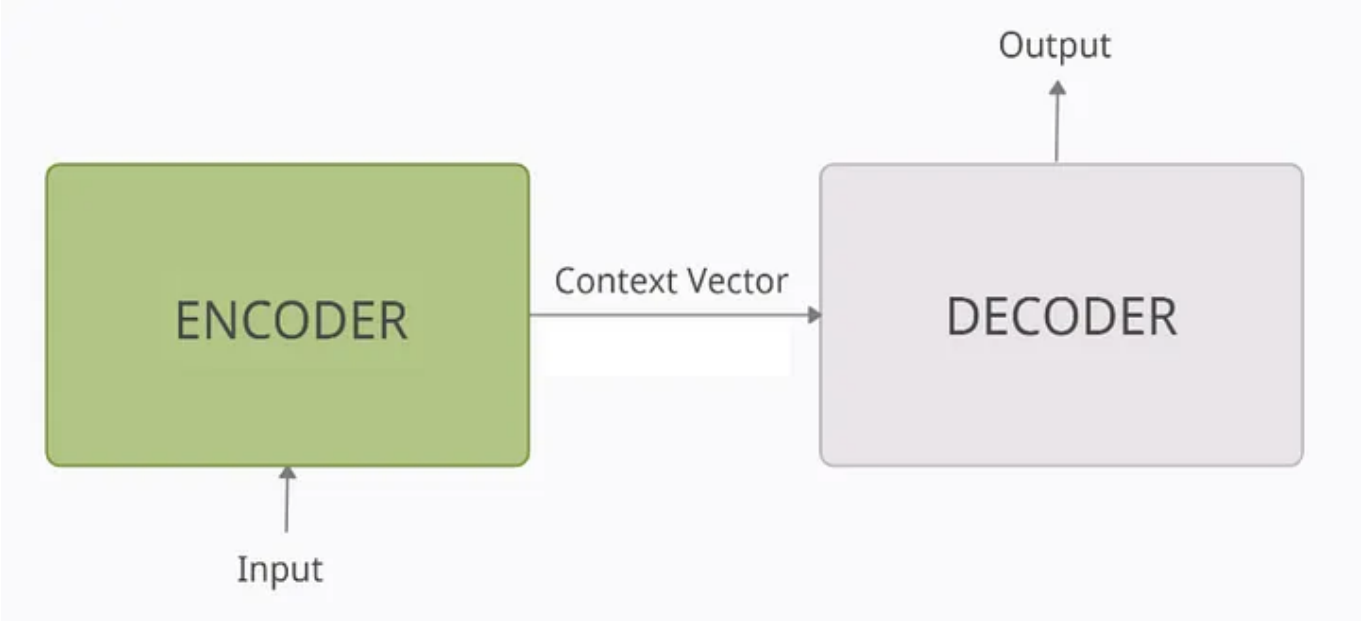


Image by author

- **Encoder:** The encoder processes each token in the input-sequence. It tries to cram all the information about the input-sequence into a vector of fixed length i.e. the 'context vector'. After going through all the tokens, the encoder passes this vector onto the decoder.
- **Context vector:** The vector is built in such a way that it's expected to encapsulate the whole meaning of the input-sequence and help the decoder make accurate predictions. We will see later that this is the final internal states of our encoder block.
- **Decoder:** The decoder reads the context vector and tries to predict the target-sequence token by token.

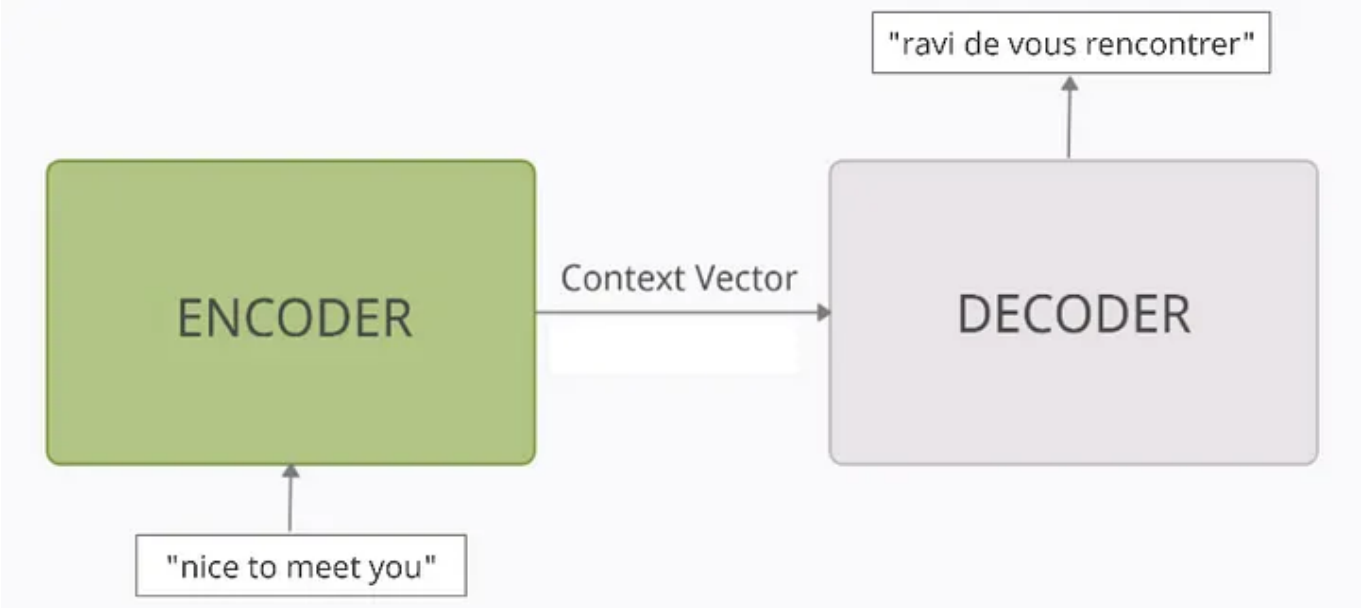


Image by author

What's under the hood?

The internal structure of both the blocks would look something like this:

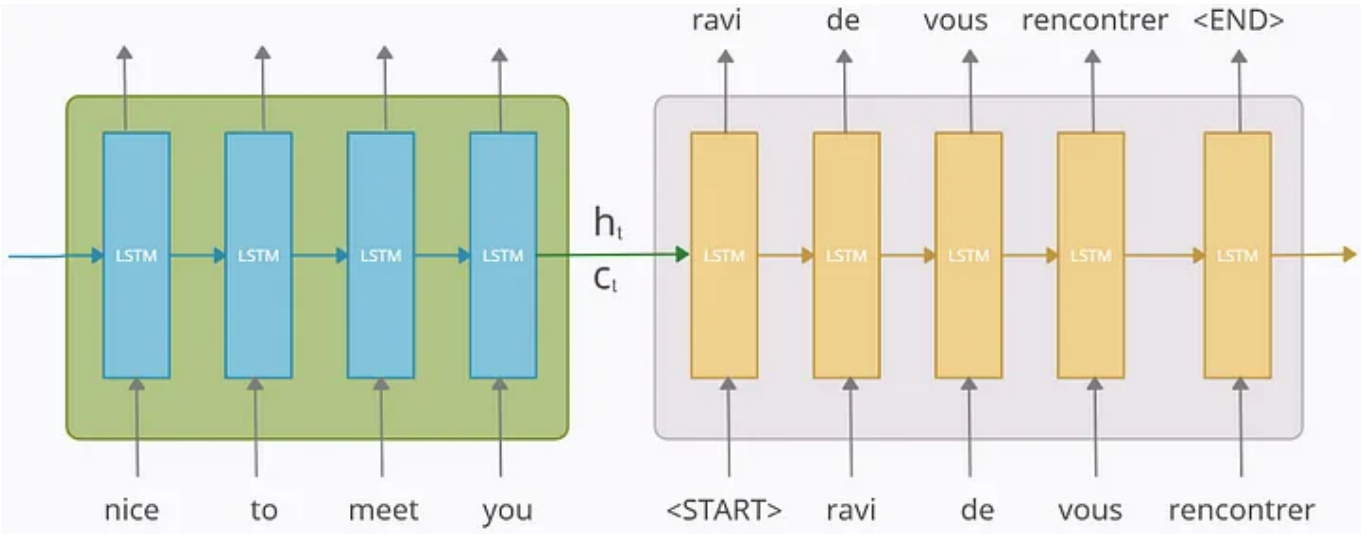


Image by author

As far as architecture is concerned, it's quite straightforward. The model can be thought of as two LSTM cells with some connection between them. The main thing here is how we deal with the inputs and the outputs. I will be explaining each part one by one.

The Encoder Block

The encoder part is an LSTM cell. It is fed in the input-sequence over time and it tries to encapsulate all its information and store it in its final internal states $\mathbf{h_t}$ (hidden state) and $\mathbf{c_t}$ (cell state). The internal states are then passed onto the decoder part, which it will use to try to produce the target-sequence. This is the 'context vector' which we were earlier referring to.

The outputs at each time-step of the encoder part are all discarded

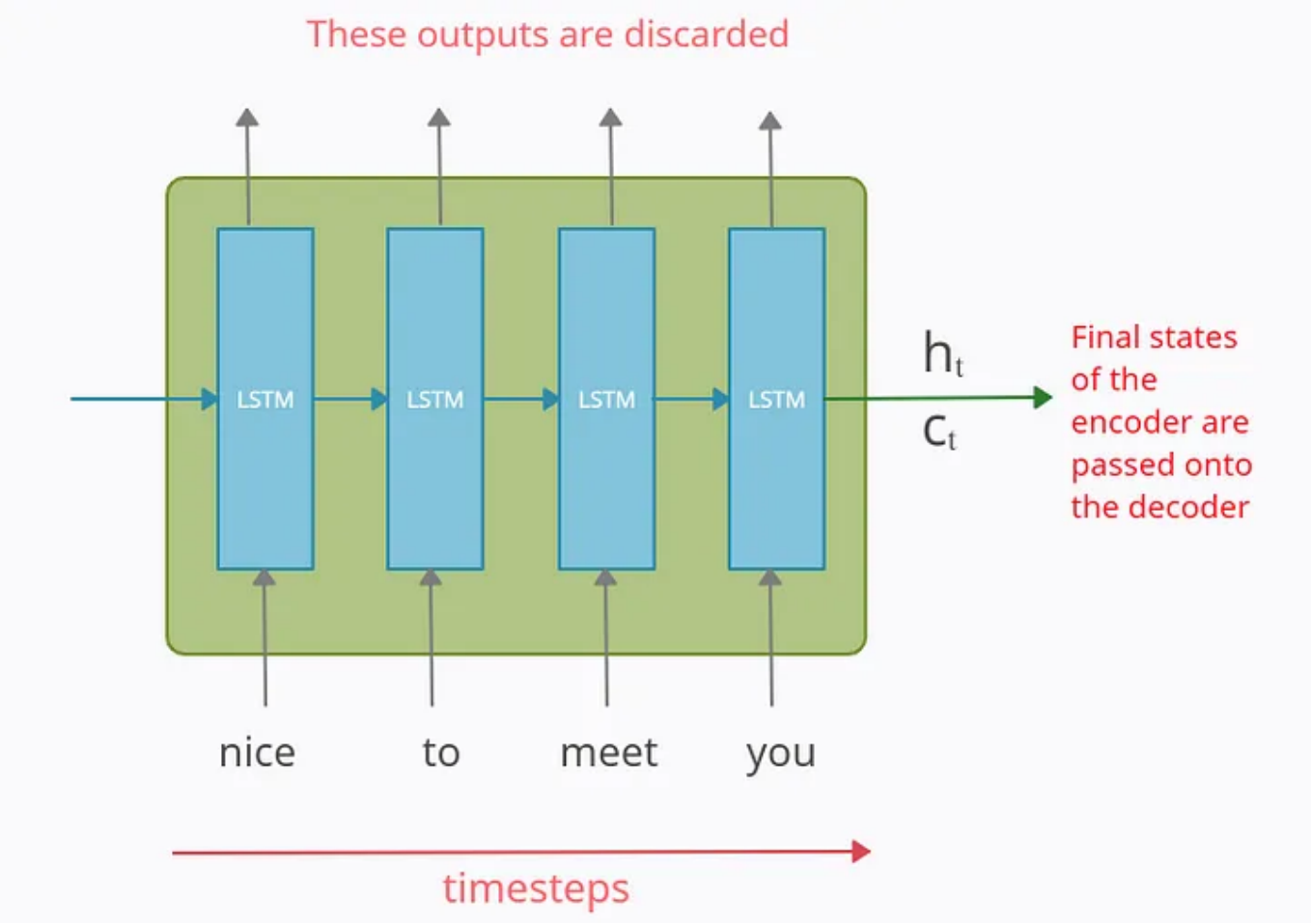


Image by author

Note: Above diagram is what an LSTM/GRU cell looks like when we unfold it on the time axis. i.e. it is the single LSTM/GRU cell that takes a single word/token at each timestamp.

In the paper, they have used LSTMs instead of classical RNNs because they work better with long-term dependencies. I have seen people use GRUs as well.

The Decoder Block

So after reading the whole input-sequence, the encoder passes the internal states to the decoder and this is where the prediction of output-sequence begins.

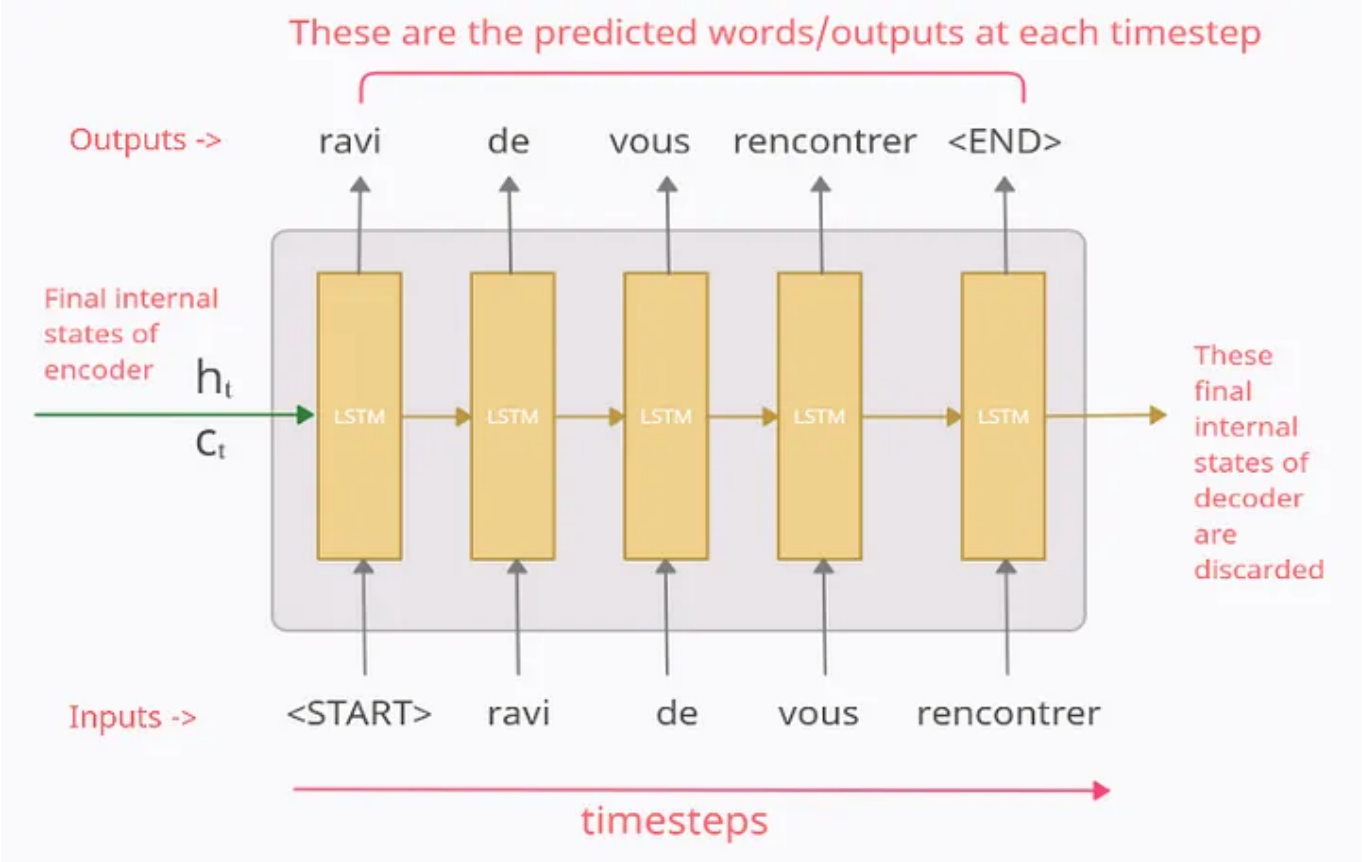


Image by author

The decoder block is also an LSTM cell. The main thing to note here is that the initial states $(\mathbf{h_0}, \mathbf{c_0})$ of the decoder are set to the final states $(\mathbf{h_t}, \mathbf{c_t})$ of the encoder. These act as the 'context' vector and help the decoder produce the desired target-sequence.

Now the way decoder works, is, that its output at any time-step **t** is supposed to be the **t^{th}** word in the target-sequence/Y_true ("ravi de vous rencontrer"). To explain this, let's see what happens at each time-step.

At time-step 1

The input fed to the decoder at the first time-step is a special symbol **"<START>"**. This is used to signify the start of the output-sequence. Now the decoder uses this input and the internal states **(h_t, c_t)** to produce the output in the 1st time-step which is supposed to be the 1st word/token in the target-sequence i.e. **'ravi'**.

At time-step 2

At time-step 2, the output from the 1st time-step **"ravi"** is fed as input to the 2nd time-step. The output in the 2nd time-step is supposed to be the 2nd word in the target-sequence i.e. **'de'**

And similarly, the output at each time-step is fed as input to the next time-step. This continues till we get the "<END>" symbol which is again a special symbol used to mark the end of the output-sequence. The final internal states of the decoder are discarded.

Note that these special symbols need not necessarily be "<START>" and "<END>" *only. These can be any strings given that these are not present in our data corpus so the model doesn't confuse them with any other word. In the paper, they have used the symbol* "<EOS>" *and in a slightly different manner. I will be talking a little bit more about this later.*

NOTE: The process mentioned above is how an **ideal** decoder will work in the testing phase. But in the training phase, a slightly different implementation is required, to make it train faster. I have explained this in the next section.

Training and Testing Phase: Getting into the tensors

Vectorizing our data

Before getting into details of it, we first need to vectorize our data.

The raw data that we have is

- **X = "nice to meet you" → Y_true = "ravi de vous rencontrer"**

Now we put the special symbols "<START>" and "<END>" at the start and the end of our target-sequence

- **X = "nice to meet you" → Y_true = "<START> ravi de vous rencontrer <END>"**

Next the input and output data is vectorized using one-hot-encoding (ohe). Let the input and output be represented as

- **X = (x1, x2, x3, x4) → Y_true = (y0_true, y1_true, y2_true, y3_true, y4_true, y5_true)**

where xi's and yi's represent the ohe vectors for input-sequence and output-sequence respectively. They can be shown as:

For input X

‘nice’ → x1 : [1 0 0 0]

‘to’ → x2 : [0 1 0 0]

‘meet’ →x3 : [0 0 1 0]

‘you’ → x4 : [0 0 0 1]

For Output Y_true

‘<START>’ → y0_true : [1 0 0 0 0 0]

‘ravi’ → y1_true : [0 1 0 0 0 0]

‘de’ → y2_true : [0 0 1 0 0 0]

‘vous’ → y3_true : [0 0 0 1 0 0]

‘rencontrer’ → y4_true : [0 0 0 0 1 0]

‘<END>’ → y5_true : [0 0 0 0 0 1]

Note: I have used this representation for easier explanation. Both the terms ‘true sequence’ and ‘target-sequence’ have been used to refer to the same sentence “ravi de vous rencontrer” which we want our model to learn.

Training and Testing of Encoder

The working of the encoder is the same in both the training and testing phase. It accepts each token/word of the input-sequence one by one and sends the final states to the decoder. Its parameters are updated using backpropagation over time.

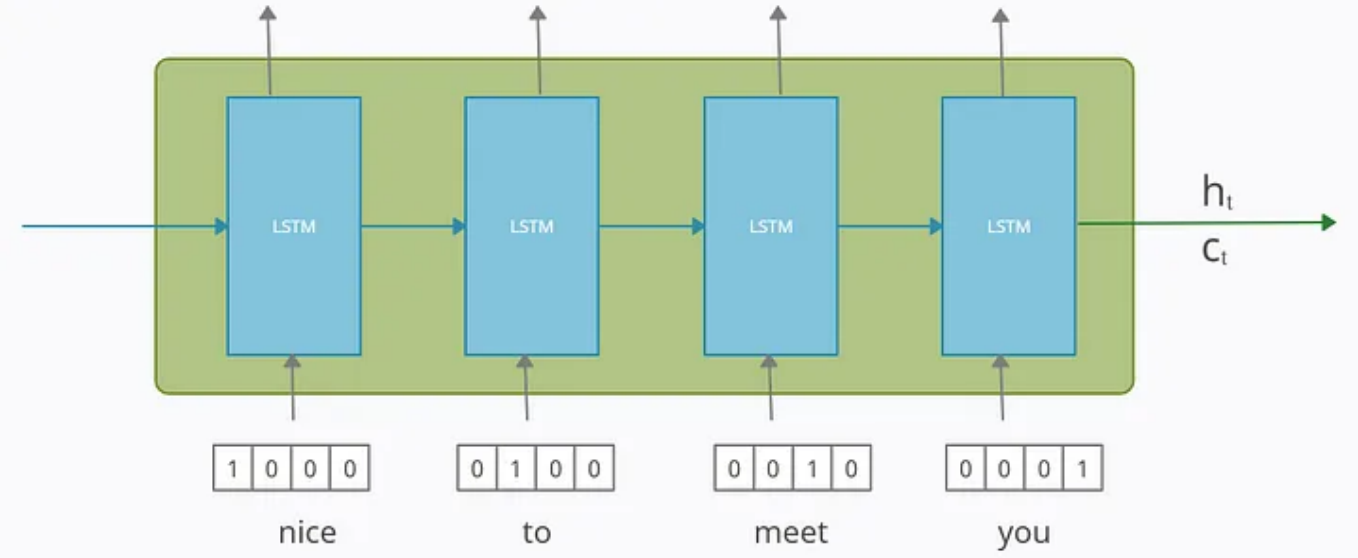


Image by author

The Decoder in Training Phase: Teacher Forcing

The working of the decoder is different during the training and testing phase, unlike the encoder part. Hence we will see both separately.

To train our decoder model, we use a technique called "**Teacher Forcing**" in which we feed the **true** output/token (and **not the predicted** output/token) from the previous time-step as input to the current time-step.

To explain, let's look at the 1st iteration of training. Here, we have fed our input-sequence to the encoder, which processes it and passes its final internal states to the decoder. Now for the decoder part, refer to the diagram below.

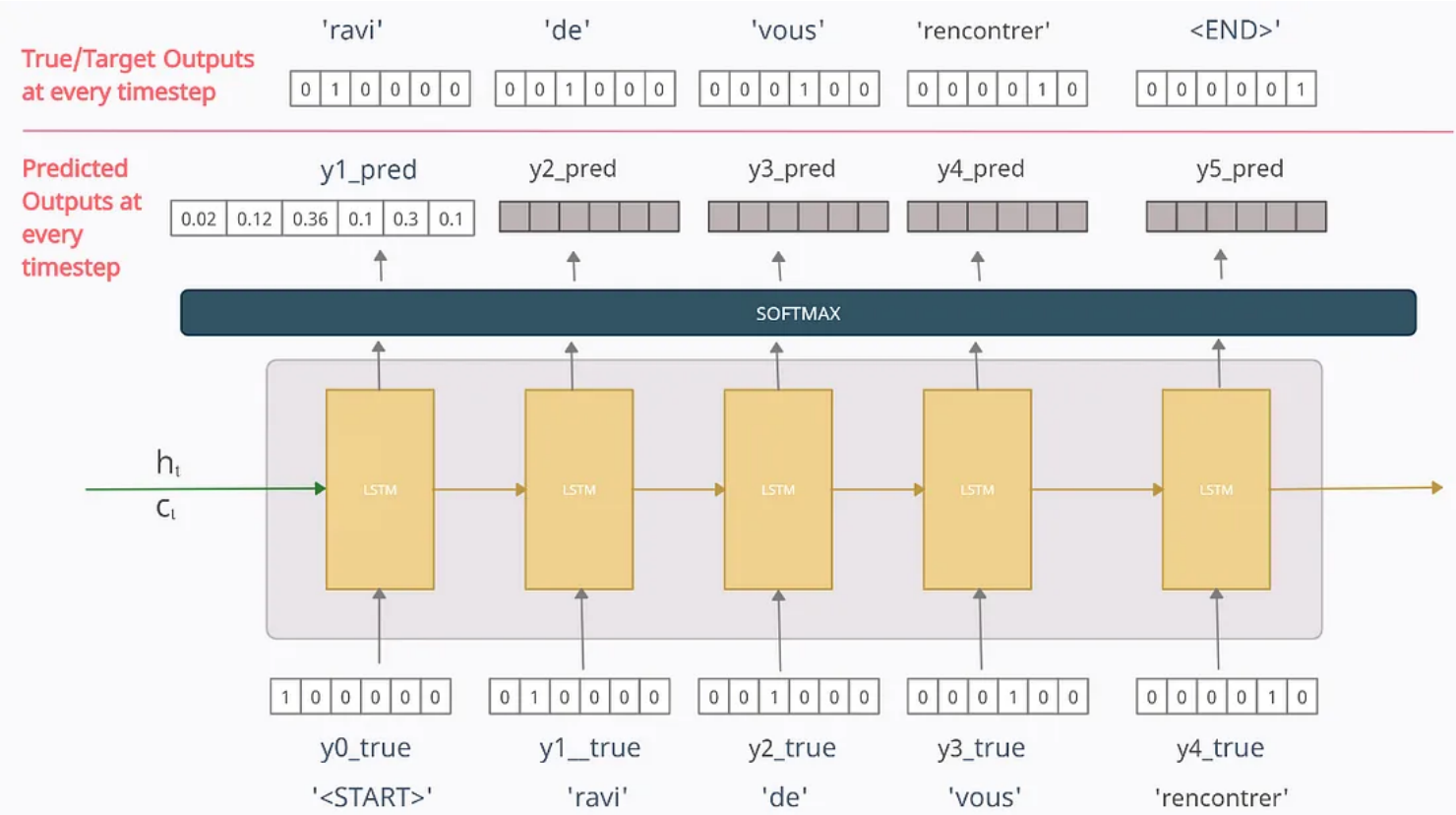


Image by author

Before moving on, note that in the decoder, at any time-step **t**, the output **yt_pred** is the probability distribution over the entire vocabulary in the output dataset which is generated by using the Softmax activation function. The token with the maximum probability is chosen to be the predicted word.

For eg. referring to the above diagram, y1_pred = [0.02 0.12 0.36 0.1 0.3 0.1] tells us that our model thinks that the probability of 1st token in the output-sequence being '<START>' is 0.02, 'ravi' is 0.12, 'de' is 0.36 and so on. We take the predicted word to be the one with the highest probability. Hence here the predicted word/token is **'de'** with a probability of 0.36

Moving on...

At time-step 1

The vector [1 0 0 0 0 0] for the word '<START>' is fed as the input vector. Now here I want my model to predict the output as y1_true=[0 1 0 0 0 0] but since my model has just started training, it will output something random. Let the predicted value at time-step 1 be y1_pred=[0.02 0.12 0.36 0.1 0.3 0.1] meaning it predicts the 1st token to be **'de'**. Now, should we use this y1_pred as the input at time-step 2?. We can do that, but in practice, it was seen that this leads to problems like slow convergence, model instability, and poor skill which is quite logical if you think.

Thus, **teacher forcing** was introduced to rectify this. in which we feed the true output/token (and not the predicted output) from the previous time-step as input to the current time-step. That means the input to the time-step 2 will be y1_true=[0 1 0 0 0 0] and not y1_pred.

Now the output at time-step 2 will be some random vector y2_pred. But at time-step 3 we will be using input as y2_true=[0 0 1 0 0 0] and not y2_pred. Similarly at each time-step, we will use the true output from the previous time-step.

Finally, the loss is calculated on the predicted outputs from each time-step and the errors are backpropagated through time to update the parameters of the model. The loss function used is the categorical cross-entropy loss function between the target-sequence/**Y_true** and the predicted-sequence/**Y_pred** such that

- Y_true = [y0_true, y1_true, y2_true, y3_true, y4_true, y5_true]
- Y_pred = ['<START>', y1_pred, y2_pred, y3_pred, y4_pred, y5_pred]

The final states of the decoder are discarded

The Decoder in Test Phase

In a real-world application, we won't have Y_true but only X. Thus we can't use what we did in the training phase as we don't have the target-sequence/Y_true. Thus when we are testing our model, the predicted output (and not the true output unlike the training phase) from the previous time-step is fed as input to the current time-step. Rest is all same as the training phase.

So let's say we have trained our model and now we test it on the single sentence we trained it on. **Now If we trained the model well and that too only on a single sentence then it should perform almost perfectly but for the sake of explanation say our model is not trained well or partially trained and now we test it. Let the scenario be depicted by the diagram below**

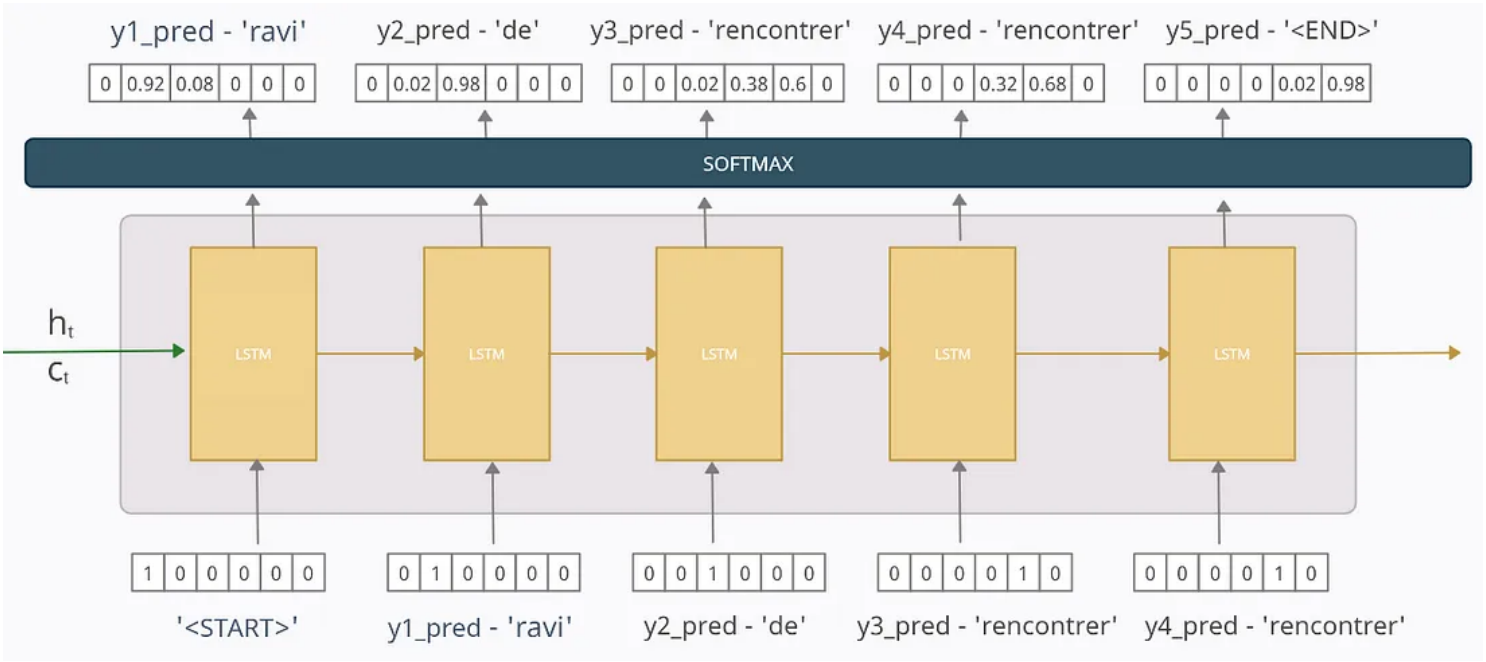


Image by author

At time-step 1

y1_pred = [0 0.92 0.08 0 0 0] tells that the model is predicting the 1st token/word in the output-sequence to be 'ravi' with a probability of 0.92 and so now at the next time-step this predicted word/token will only be used as input.

At time-step 2

The predicted word/token **"ravi"** from 1st time-step is used as input here. Here the model predicts the next word/token in the output-sequence to be **'de'** with a probability of 0.98 which is then used as input at time-step 3

And the similar process is repeated at every time-step till the '<END>' token is reached

Better visualization for the same would be:

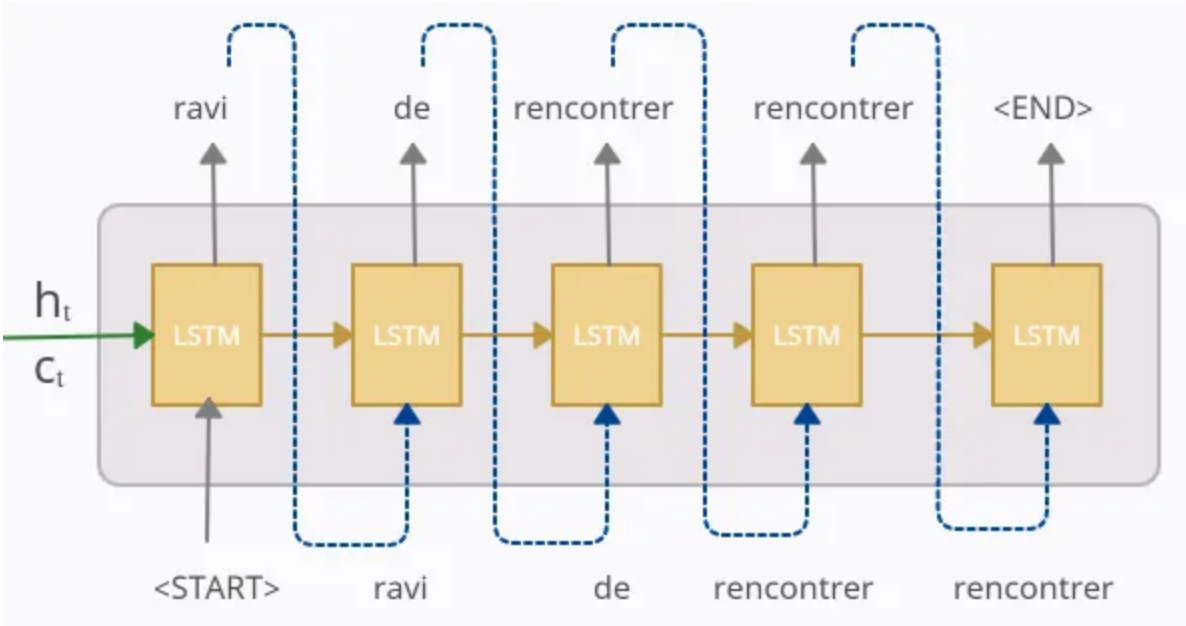


Image by author

So according to our trained model, the predicted-sequence at test time is "ravi de rencontrer rencontrer". Hence though the model was incorrect on

the 3rd prediction, we still fed it as input to the next time-step. The correctness of the model depends on the amount of data available and how well it has been trained. The model may predict a wrong output but nevertheless, the same output is only fed to the next time-step in the test phase.

The Embedding Layer

One little detail that I didn't tell you before was that the input-sequence in both the decoder and the encoder is passed through an embedding layer to reduce the dimensions of the input word vectors because in practice, the one-hot-encoded vectors can be very large and the embedded vectors are a much better representation of words. For the encoder part, this can be shown below where the embedding layer reduces the dimensions of the word vectors from four to three.

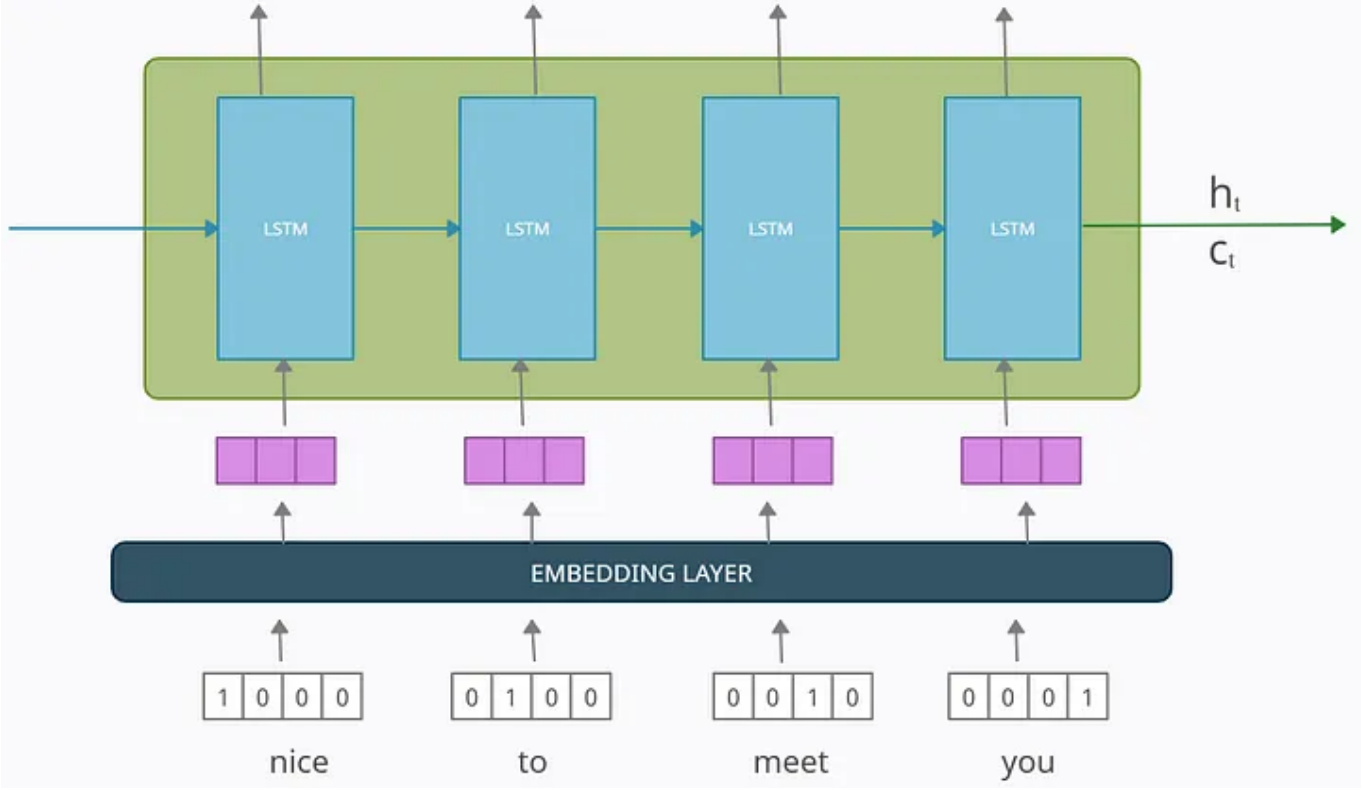


Image by author

This embedding layer can be pre-trained like Word2Vec embeddings or can be trained with the model itself.

The Final Visualization at test time

Somehow tried to capture everything in one diagram

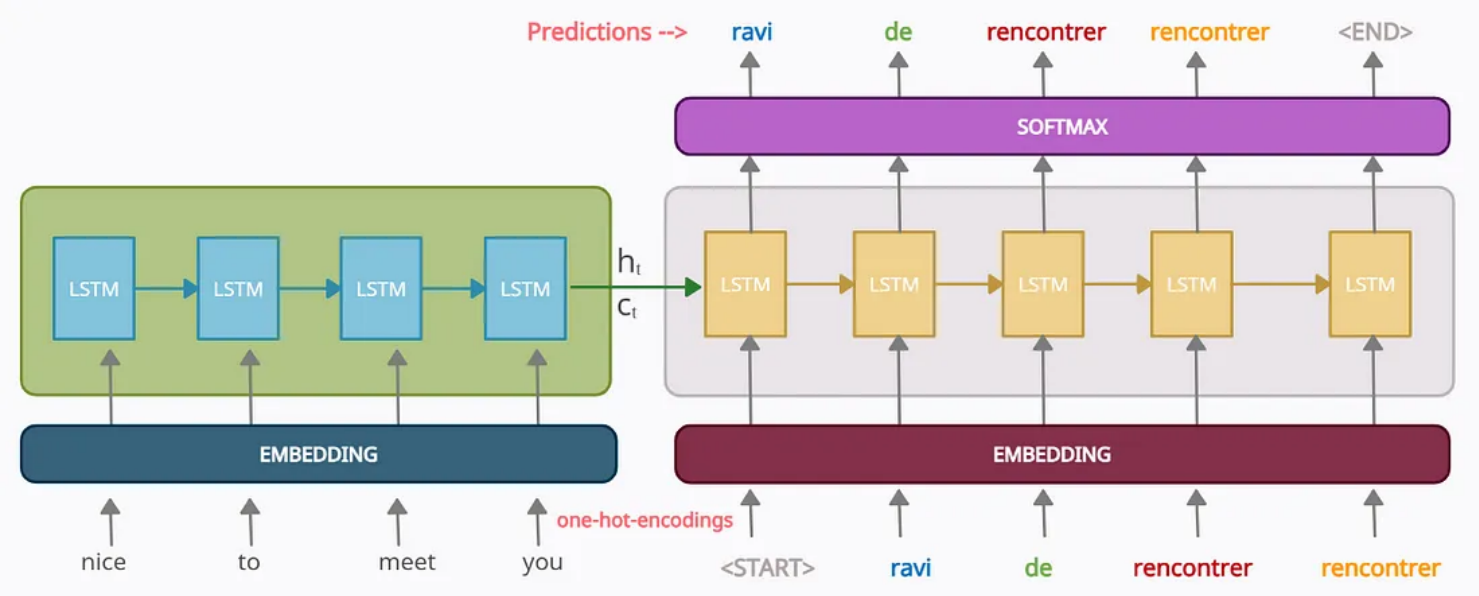


Image by author

You might find some slight differences in implementations here and there but the main idea remains the same.

The "Sutskever Model"

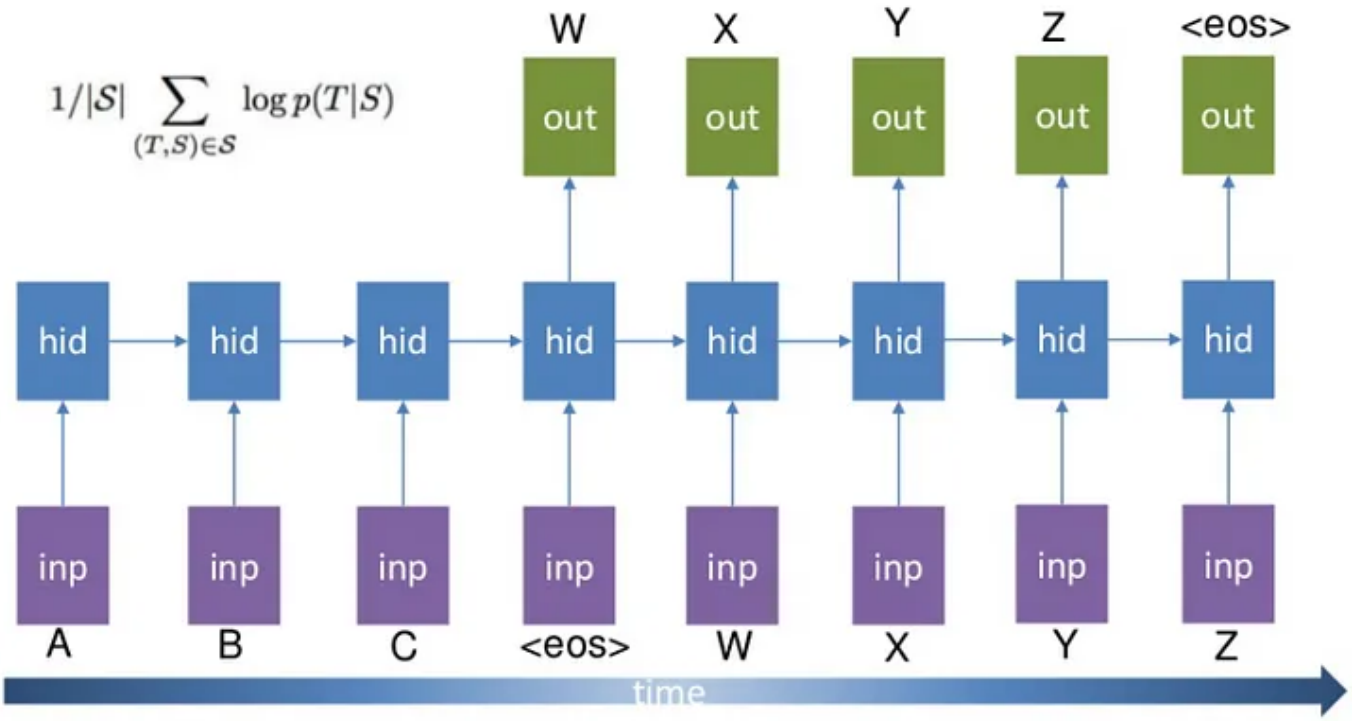
The paper Sequence to Sequence Learning with Neural Networks by Ilya Sutskever, et al was one of the pioneer papers introducing the encoder-decoder model for machine translation and in general Seq2Seq modelling.

Here, I will mention the important points from the paper and how their implementation was different from the toy example that we learned from.

- The model was applied to English to French translation
- Each sentence was made to end with a special end-of-sentence symbol "<EOS>"

> *"Note that we require that each sentence ends with a special end-of-sentence symbol "<EOS>", which enables the model to define a distribution over sequences of all possible lengths." [1]*

In the paper, to explain, they took input-sequence as "A B C" and target-sequence as "W X Y Z"



https://www.slideshare.net/quangntta/sequence-to-sequence-learning-with-neural-networks

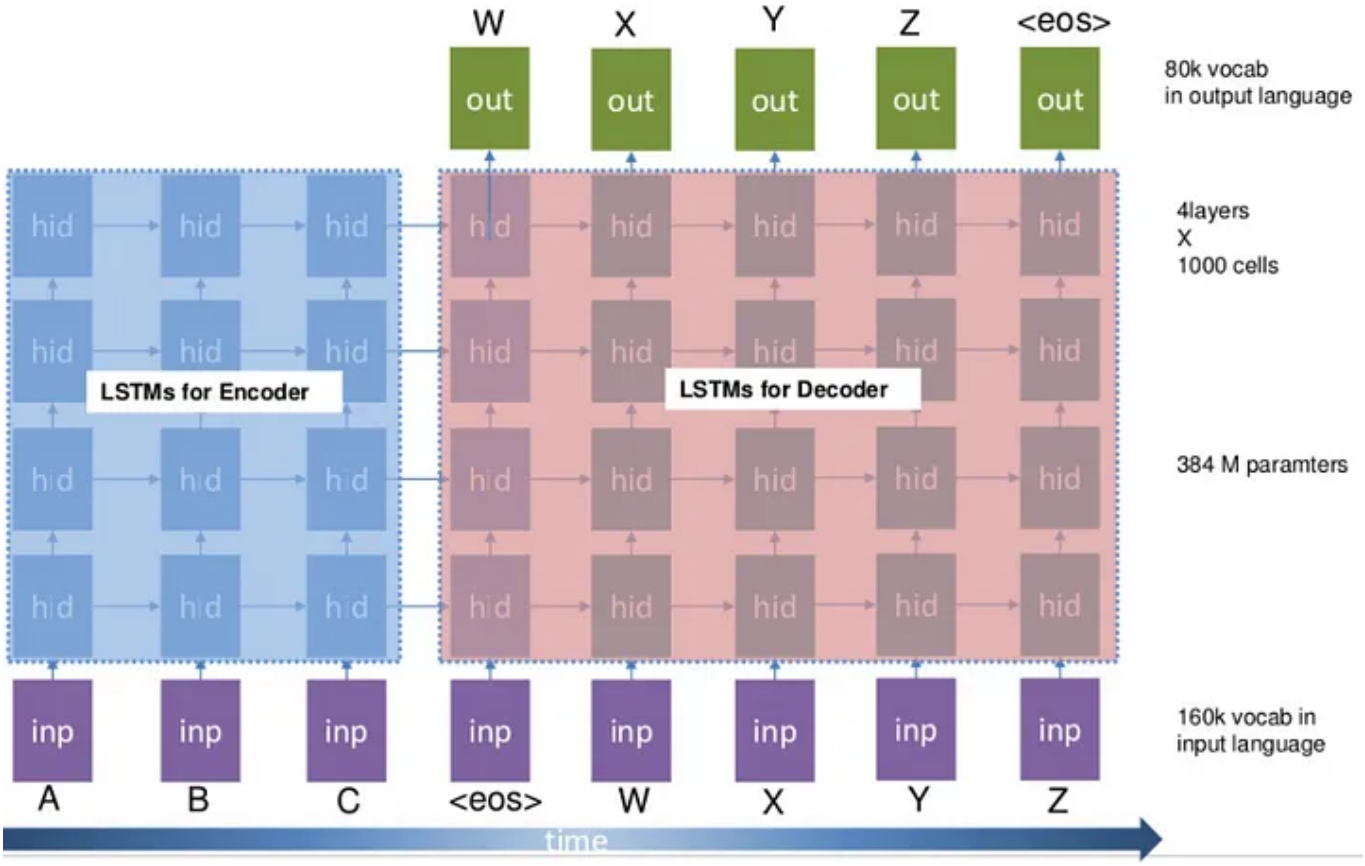
- **Dataset Details**

> *"We trained our models on a subset of 12M sentences consisting of 348M French words and 304M English words, which is a clean "selected" subset from [29]. We chose this translation task and this specific training set subset because of the public availability of a tokenized training and test set together with 1000-best lists from the baseline SMT [29]. As typical neural language models rely on a vector representation for each word, we used a fixed vocabulary for both languages. We used 160,000 of the most frequent words for the source language and 80,000 of the most frequent words for the target language. Every out-of-vocabulary word was replaced with a special "UNK" token."[1]*

- **Input-sequences were reversed.**

> *"While the LSTM is capable of solving problems with long term dependencies, we discovered that the LSTM learns much better when the source sentences are reversed (the target sentences are not reversed)" [1]*

- A 1000-dimensional word embedding layer was used to represent the input words. Softmax was used on the output layer.
- The input and output models had 4 layers with 1,000 units per layer.



https://www.slideshare.net/quangntta/sequence-to-sequence-learning-with-neural-networks

- A BLEU score of 34.81 was achieved.

> *"... we obtained a BLEU score of 34.81 [...] This is by far the best result achieved by direct translation with large neural networks. For comparison, the BLEU score of an SMT baseline on this dataset is 33.30" [1]*

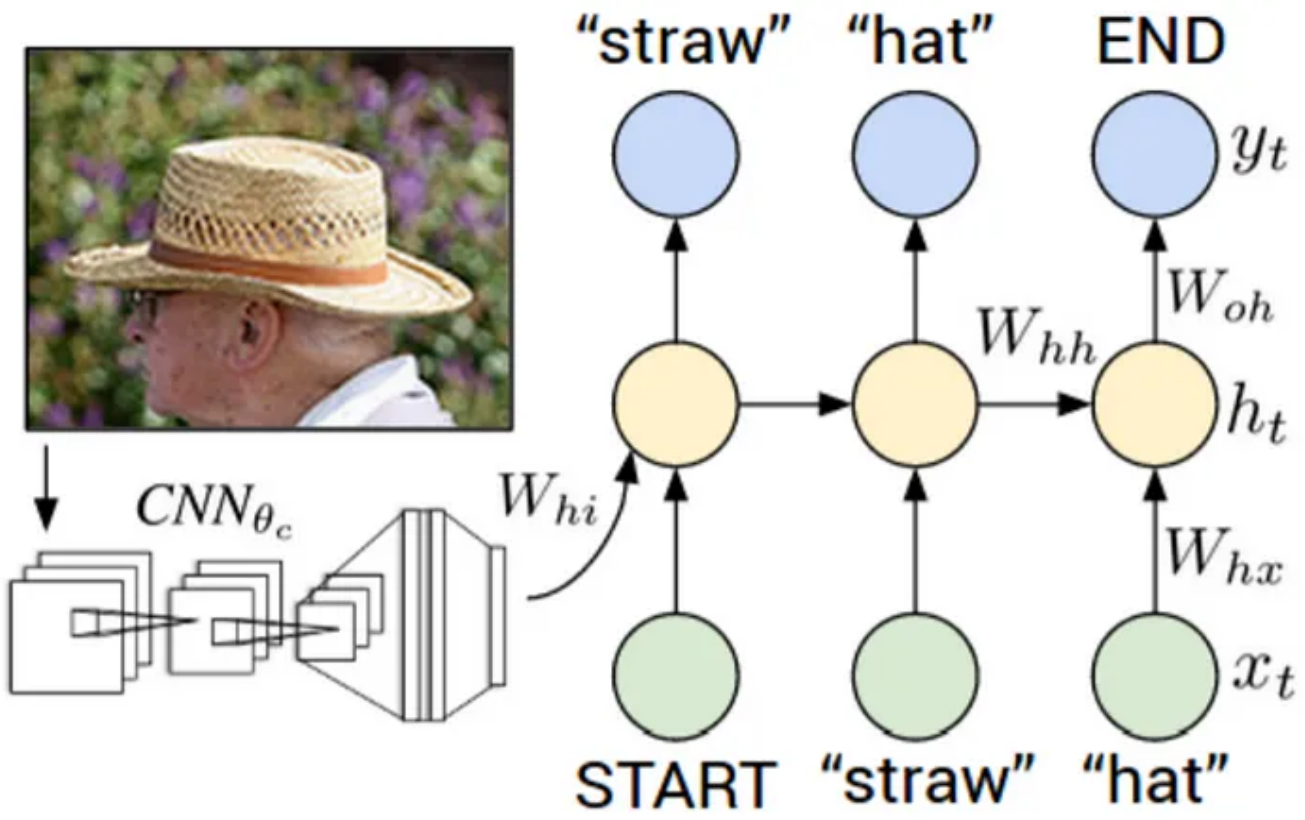
Here is a talk on the same paper for your reference.

https://www.youtube.com/watch?v=-uyXE7dY5H0

Image Captioning Intuition

The paper Deep Visual-Semantic Alignments for Generating Image Descriptions by Andrej Karpathy, Li Fei-Fei, et al used a different version of the encoder-decoder model for image captioning. Image Captioning is the process of generating a textual description of an image.

In the model proposed by them, the image is fed into a CNN network of your choice. Then the last Dense layer is fed into the LSTM network. This layer in some sense acts as a context vector as it captures the essence of the image.



Deep Visual-Semantic Alignments for Generating Image Descriptions

Further Reading

If you've found your way here, then congratulations!! I hope you've found this useful. Since this is my first attempt at blogging, I hope that the readers will be forgiving and overlook any slight errors I may have made. I am mentioning more resources on the topic if you are looking to go deeper.

1. I would suggest you to go through the paper we have discussed **Sequence to Sequence Learning with Neural Networks by Ilya Sutskever, et al** by yourself once.
2. Nothing helps you understand a concept better than implementing it yourself. I found this amazing blog **Neural Machine Translation with Keras** where an Encoder-Decoder model is trained from scratch for English to French translation.

3. As mentioned before, there are a lot of advanced techniques which have been developed over encoder-decoder models for Seq2Seq modelling. For them I strongly suggest you to refer to the following in order

- Attention Mechanism → *Jay Alammar,* **Visualizing A Neural Machine Translation Model** blog post, 2018.
- Transformer → *Jay Alammar,* **The Illustrated Transformer** blog post, 2018.
- BERT → Jay Alammar, **The Illustrated BERT** blog post, 2018.

References

[1] Sequence to Sequence Learning with Neural Networks by Ilya Sutskever, et al

[2] Deep Visual-Semantic Alignments for Generating Image Descriptions by Andrej Karpathy, Li Fei-Fei, et al

[3] https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

[4] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[6] https://machinelearningmastery.com/encoder-decoder-recurrent-neural-network-models-neural-machine-translation/

[7] https://www.slideshare.net/quangntta/sequence-to-sequence-learning-with-neural-networks

The diagrams in the post have been made using creately

PS: Feel free to provide comments/criticisms if you think they can improve this blog, I will definitely try to make the required changes.

Deep Learning | NLP | Sequence To Sequence | Machine Translation | Encoder Decoder

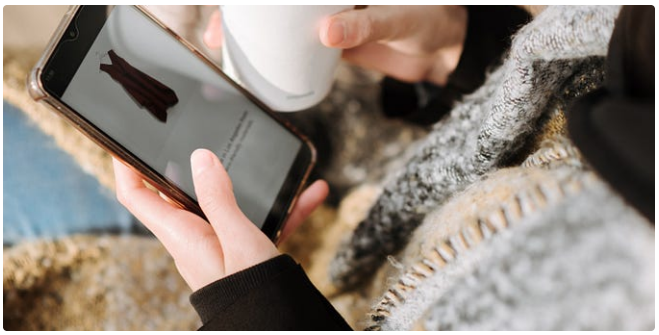
Written by Kriz Moses

Follow

73 Followers · Writer for Analytics Vidhya

https://www.linkedin.com/in/kriz-moses/

More from Kriz Moses and Analytics Vidhya



Kriz Moses in Towards Data Science

Modeling Product Search Relevance in e-Commerce: Home...

Predict the relevance of search results on homedepot.com using Machine Learning

31 min read · Jun 29, 2021

275

Sajal Rastogi in Analytics Vidhya

Wifi -Hacking using PyWifi

.

4 min read · Feb 6, 2021

252 1

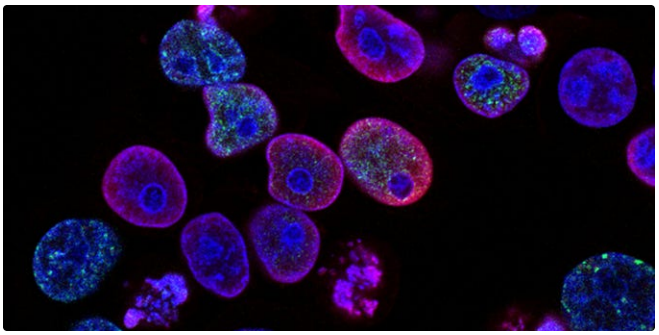
Kia Eisinga in Analytics Vidhya

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open...

7 min read · Jan 27, 2020

2K 24



Kriz Moses in Towards AI

Medical Image Segmentation: 2018 Data Science Bowl

A case study on Nucleus Segmentation across imaging experiments using Deep...

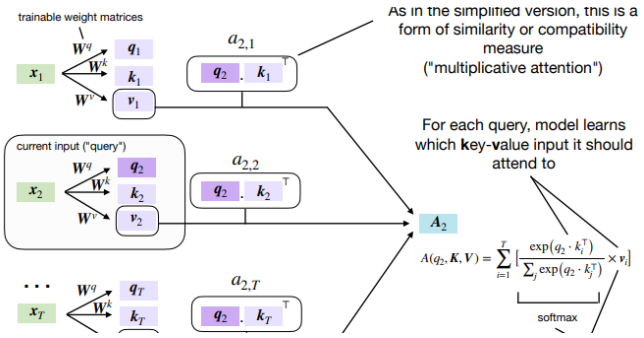
18 min read · Oct 25, 2021

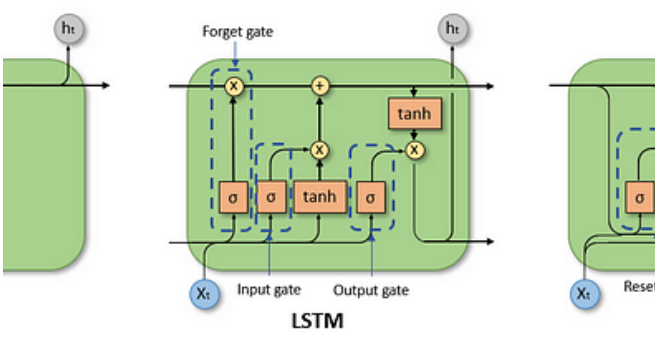
86

See all from Kriz Moses

See all from Analytics Vidhya

Recommended from Medium





Zain ul Abideen

Attention Is All You Need: The Core Idea of the Transformer

An overview of the Transformer model and its key components.

6 min read · Jun 26

144

Jonte Dancker in Towards Data Science

A Brief Introduction to Recurrent Neural Networks

An introduction to RNN, LSTM, and GRU and their implementation

12 min read · Dec 26, 2022

359 5

Lists

Natural Language Processing

669 stories · 283 saves

The New Chatbots: ChatGPT, Bard, and Beyond

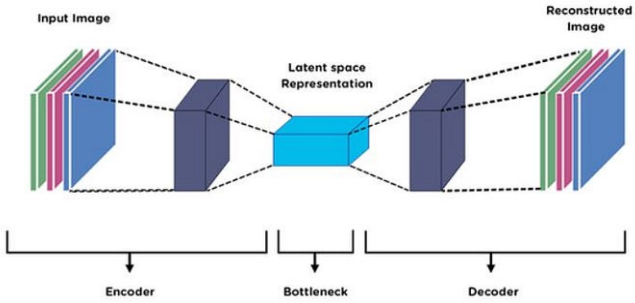
13 stories · 133 saves

Practical Guides to Machine Learning

10 stories · 519 saves

New_Reading_List

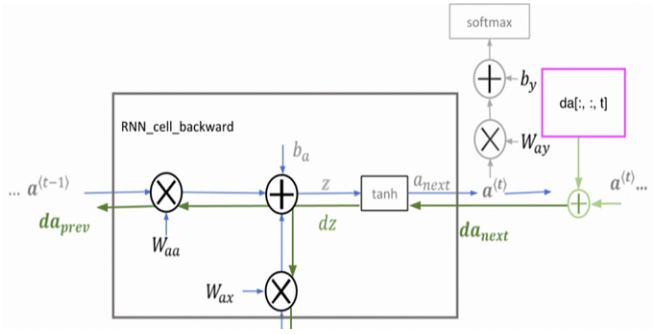
174 stories · 133 saves



Ahmadsabry

A Perfect guide to Understand Encoder Decoders in Depth with...

Introduction



Eugene Ku

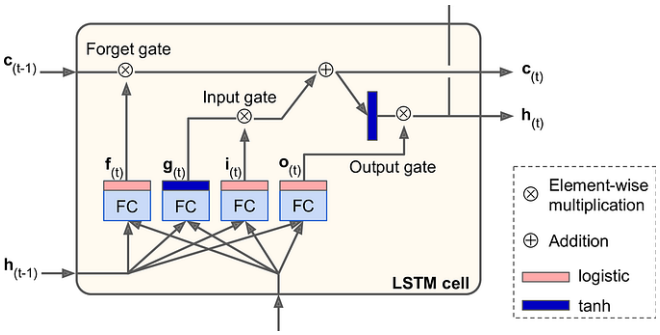
Matrix Calculus For Deep Learning: Taking Derivatives of Matrices...

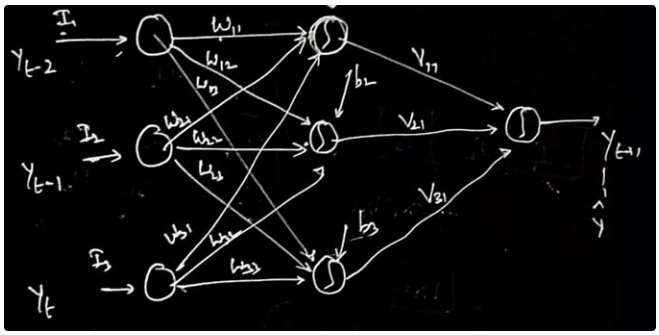
Hello everyone! Today, we are going to go through one of the most important...

6 min read · Jun 24

4 min read · Aug 4

10 1





Anishnama

Understanding LSTM: Architecture, Pros and Cons, and...

What is LSTM and How it works?

7 min read · Apr 28

153 2

Utsav Poudel in Level Up Coding

Time and Series Forecasting with LSTM- Recurrent Neural Networks

Every day, humans make passive predictions when performing tasks such as crossing a...

10 min read · May 10

306 2

See more recommendations