



Search Medium



Write

**DDI • gain access to expert views**

Member-only story

# Attention in RNNs

Understanding the mechanism with a detailed example



Nir Arbel · Follow

Published in DataDrivenInvestor · 8 min read · Mar 15, 2019



748



7



...



Montepulciano Italy

Recurrent Neural Networks (RNNs) have been used successfully for many tasks involving sequential data such as machine translation, sentiment analysis, image captioning, time-series prediction etc. Improved RNN models such as Long Short-Term Memory networks (LSTMs) enable training on long sequences overcoming problems like vanishing gradients. However, even the more advanced models have their limitations and researchers had a hard time developing high-quality models when working with long data sequences. In machine translation, for example, the RNN has to find connections between long input and output sentences composed of dozens of words. It seemed that the existing RNN architectures needed to be changed and adapted to better deal with such tasks.

**DDI Editor's Pick: 5 Machine Learning Books That Turn You from Novice to Expert - Data Driven...**

The booming growth in the Machine Learning industry has brought renewed interest in people about Artificial...

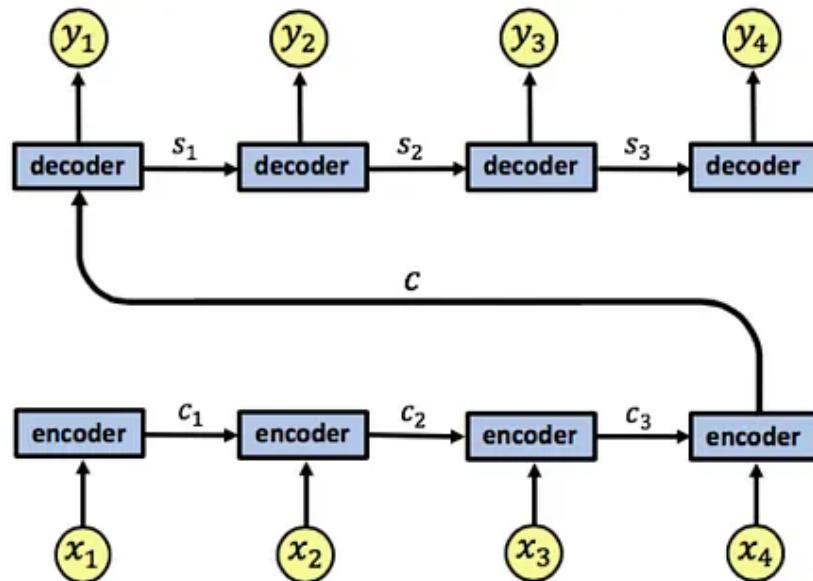
[www.datadriveninvestor.com](http://www.datadriveninvestor.com)

Attention is a mechanism combined in the RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and of higher quality.

Combination of attention mechanisms enabled improved performance in many tasks making it an integral part of modern RNN networks.

This work is based on the paper Neural machine translation by jointly learning to align and translate by Bahdanau, Cho, and Bengio [1].

We start by briefly going over basic RNNs. The RNN encoder-decoder architecture we will focus on looks like this:



An RNN encoder-decoder architecture, we take an architecture with 4 time steps for simplicity

The RNN encoder has an input sequence  $x_1, x_2, x_3, x_4$ . We denote the encoder states by  $c_1, c_2, c_3$ . The encoder outputs a single output vector  $c$

which is passed as input to the decoder. Like the encoder, the decoder is also a single-layered RNN, we denote the decoder states by  $s_1, s_2, s_3$  and the network's output by  $y_1, y_2, y_3, y_4$ .

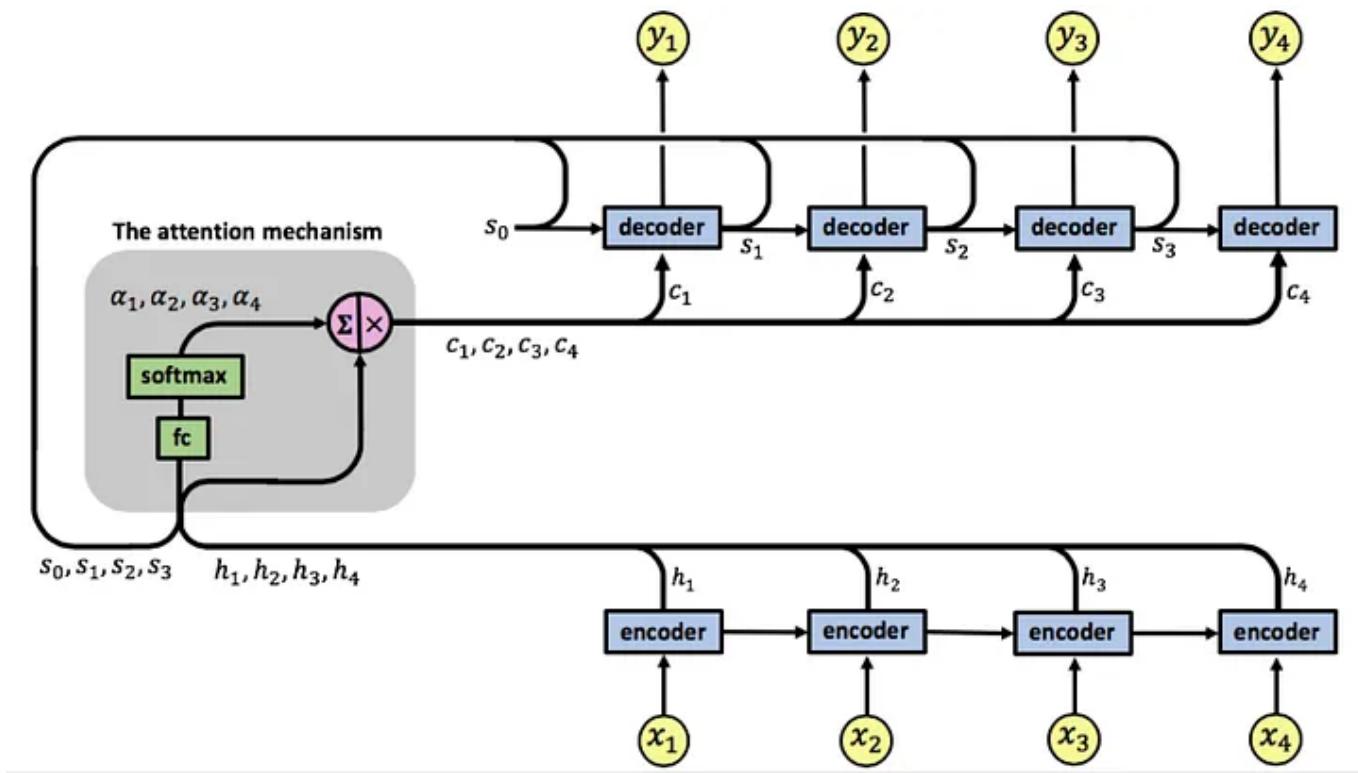
A problem with this architecture lies in the fact that the decoder needs to represent the entire input sequence  $x_1, x_2, x_3, x_4$  as a single vector  $c$ , which can cause information loss. Moreover, the decoder needs to decipher the passed information from this single vector, a complex task in itself.

As detailed in [1]:

*A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus.*

## RNNs with an attention mechanism

An attention RNN looks like this:



Our attention model has a single layer RNN encoder, again with 4-time steps. We denote the encoder's input vectors by  $x_1, x_2, x_3, x_4$  and the output vectors by  $h_1, h_2, h_3, h_4$ .

The attention mechanism is located between the encoder and the decoder, its input is composed of the encoder's output vectors  $h_1, h_2, h_3, h_4$  and the states of the decoder  $s_0, s_1, s_2, s_3$ , the attention's output is a sequence of vectors called context vectors denoted by  $c_1, c_2, c_3, c_4$ .

## The context vectors

The context vectors enable the decoder to focus on certain parts of the input when predicting its output. Each context vector is a weighted sum of the encoder's output vectors  $h_1, h_2, h_3, h_4$ , each vector  $h_i$  contains information about the whole input sequence (since it has access to the encoder states during its computation) with a strong focus on the parts surrounding the  $i$ -th vector of the input sequence. The vectors  $h_1, h_2, h_3, h_4$

are scaled by weights  $\alpha_{ij}$  capturing the degree of relevance of input  $x_j$  to output at time  $i$ ,  $y_i$ .

The context vectors  $c_1, c_2, c_3, c_4$  are given by:

$$c_i = \sum_{j=1}^4 \alpha_{ij} h_j$$

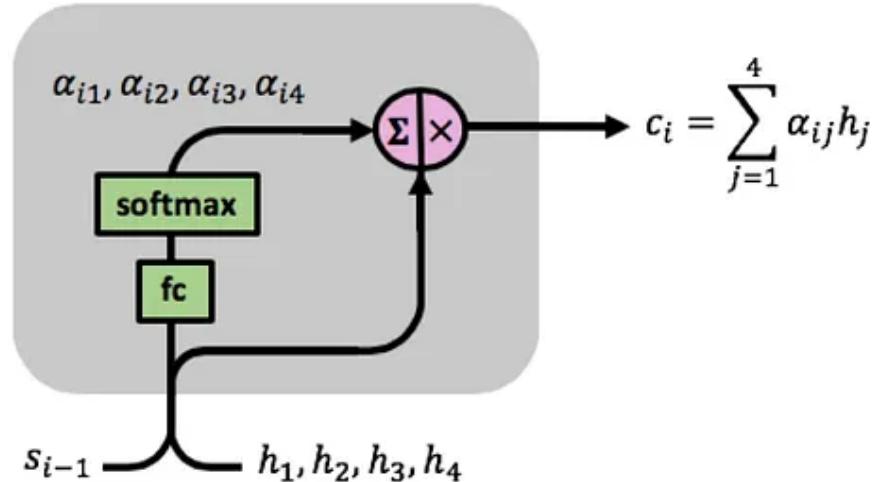
The attention weights are learned using an additional fully-connected shallow network, denoted by  $fc$ , this is where the  $s_0, s_1, s_2, s_3$  part of the attention mechanism's input comes into play. Computation of the attention weights is given by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^4 \exp(e_{ik})}$$

*where*       $e_{ij} = fc(s_{i-1}, h_j)$

The attention weights are learned using the attention fully-connected network and a softmax function:

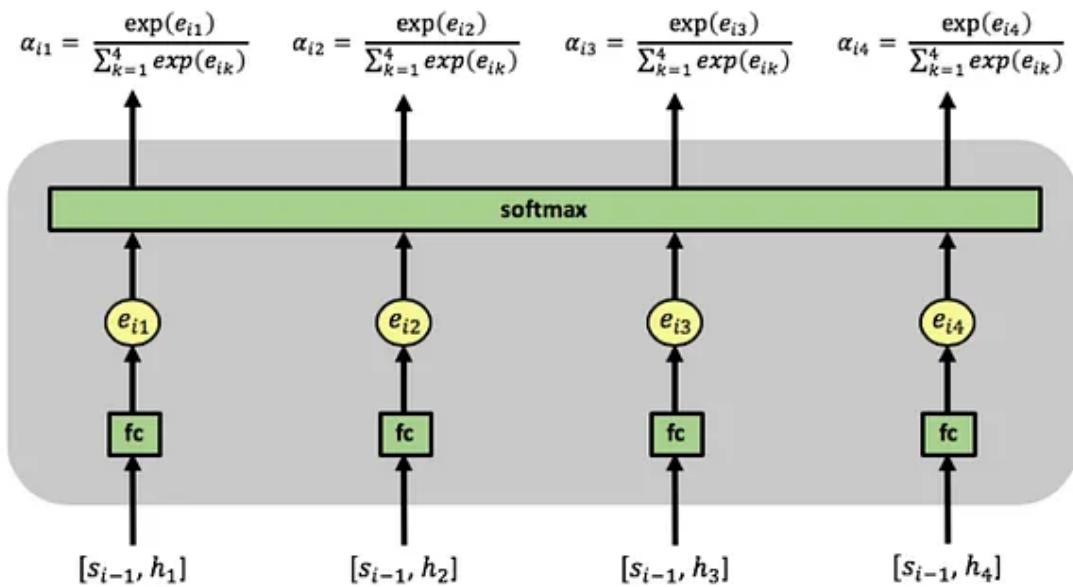
## The attention mechanism



At time step i, the mechanism has  $h_1, h_2, h_3, h_4$  and  $s_{i-1}$  as inputs, it uses the fc neural network and the softmax function to compute the attention weights  $\alpha_{i1}, \alpha_{i2}, \alpha_{i3}, \alpha_{i4}$ , these are then used in the computation of the context vector  $c_i$ .

As can be seen in the above image, the fully-connected network receives the concatenation of vectors  $[s_{i-1}, h_i]$  as input at time step i. The network has a single fully-connected layer, the outputs of the layer, denoted by  $e_{ij}$ , are passed through a softmax function computing the attention weights, which lie in  $[0,1]$ .

Notice that we are using the same fully-connected network for all the concatenated pairs  $[s_{i-1}, h_1], [s_{i-1}, h_2], [s_{i-1}, h_3], [s_{i-1}, h_4]$ , meaning **there is a single network learning the attention weights**.



Computation of attention weights at time step  $i$ , notice how this needs to be computed separately on every time step since the computation at time step  $i$  involves  $s_{i-1}$ , the decoder's state from time  $i-1$

The attention weights  $a_{ij}$  reflect the importance of  $h_j$  with respect to the previous hidden state  $s_{i-1}$  in deciding the next state  $s_i$  and generating  $y_i$ . A large  $a_{ij}$  attention weight causes the RNN to focus on input  $x_j$  (represented by the encoder's output  $h_j$ ), when predicting the output  $y_i$ .

The fc network is trained along with the encoder and decoder using backpropagation, the RNN's prediction error terms are backpropagated backward through the decoder, then through the fc attention network and from there to the encoder.

Notice that since the attention weights are learned using an additional neural network fc, we have an additional set of weights allowing this learning to take place, we denote this weight matrix by  $W_a$ .

An RNN with 4 input time steps and 4 output time steps will have the following weight matrices fine-tuned during the training process. Note the dimensions  $4 \times 4$  of the attention matrix, connecting between every input to every output:

*RNN encoder weights matrix  $W_e$*

*RNN decoder weights matrix  $W_d$*

*RNN attention weights matrix  $\alpha_{4 \times 4}$*

*fc weights matrix  $W_a$*

This mechanism enables the decoder to decide which parts of the input sequence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from having to encode all information in the input sequence into a single vector. The information can be spread throughout the sequence  $h_1, h_2, h_3, h_4$  which can be selectively retrieved by the decoder.

As written in [1]:

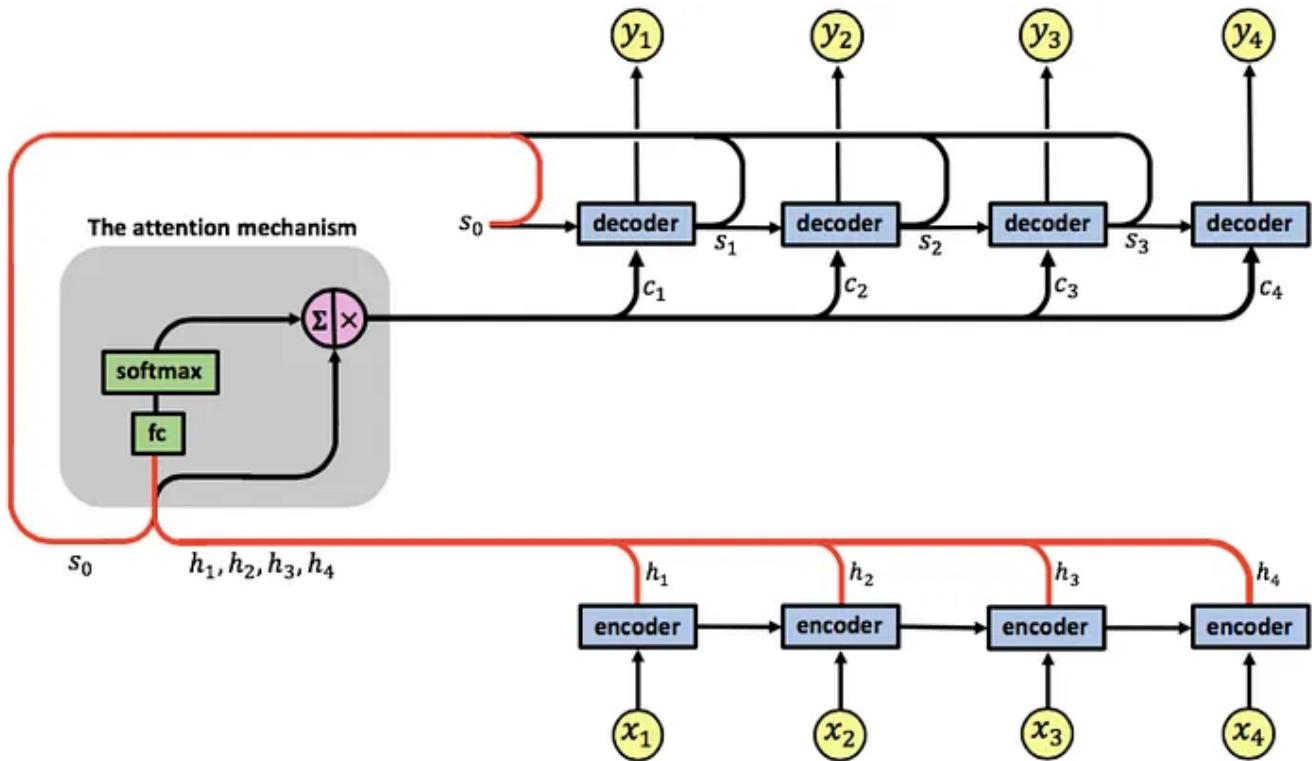
*The most important distinguishing feature of this approach from the basic encoder-decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector. We show this allows a model to cope better with long sentences.*

## **Computing the attention weights and context vectors**

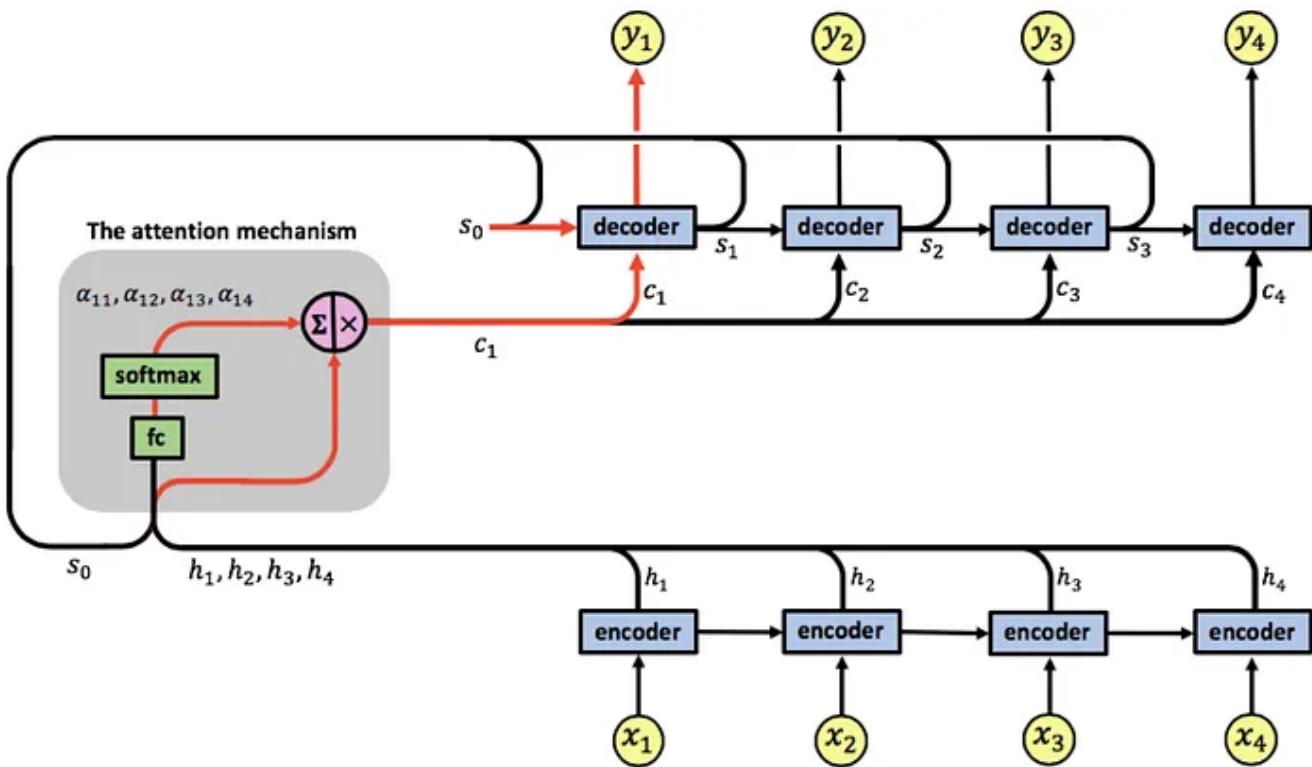
let's go over a detailed example and see how the context vectors are computed.

The first act performed is the computation of vectors  $h_1, h_2, h_3, h_4$  by the encoder. These are then used as inputs of the attention mechanism. This is

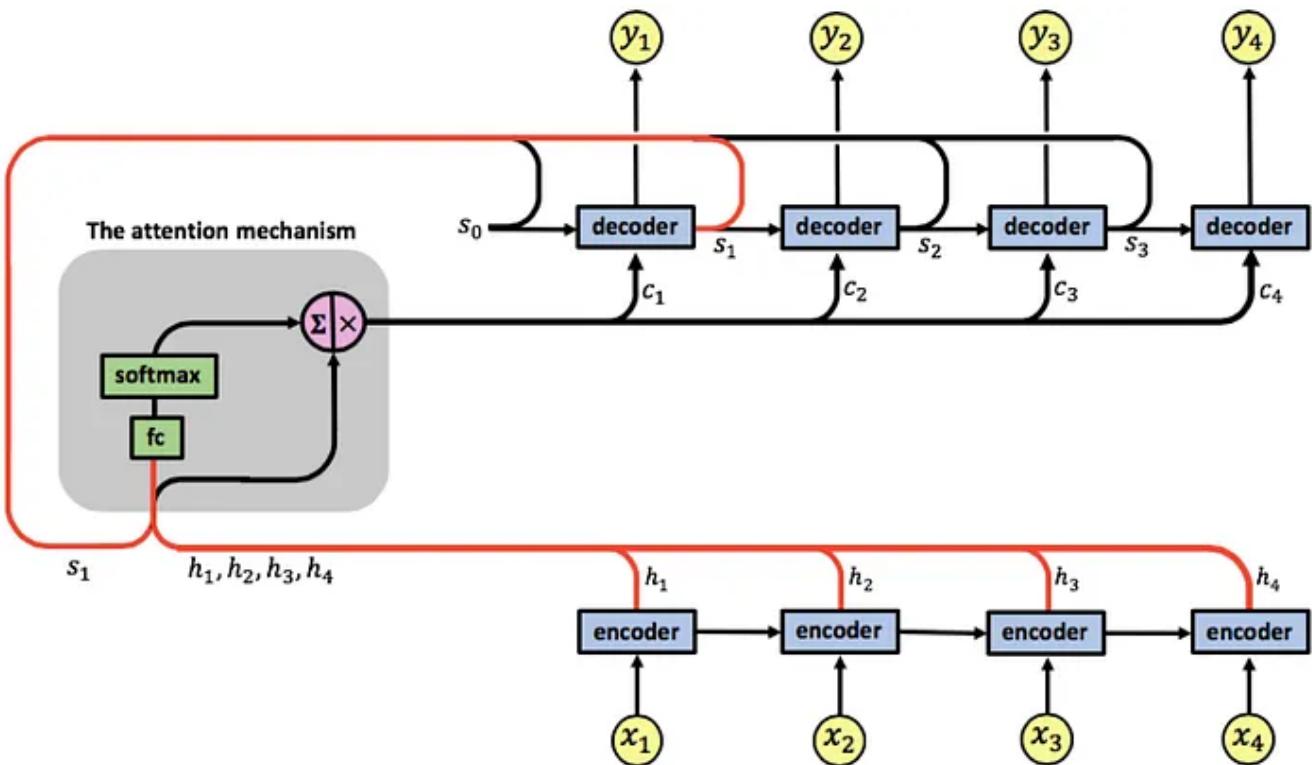
where the decoder is first involved by inputting its initial state vector  $s_0$  and we have the first attention input sequence  $[s_0, h_1], [s_0, h_2], [s_0, h_3], [s_0, h_4]$ .



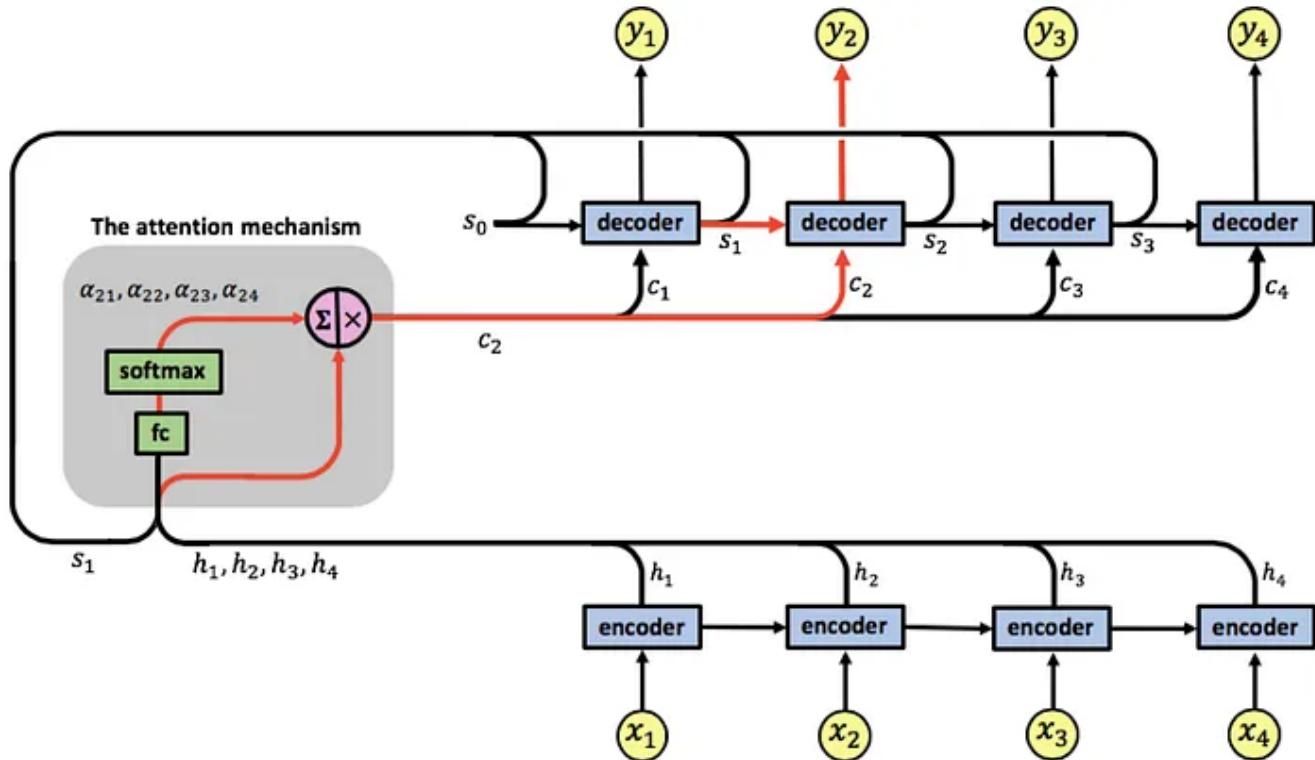
The attention mechanism computes the first set of attention weights  $\alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{14}$  enabling the computation of the first context vector  $c_1$ . The decoder now uses  $[s_0, c_1]$  and computes the first RNN output  $y_1$



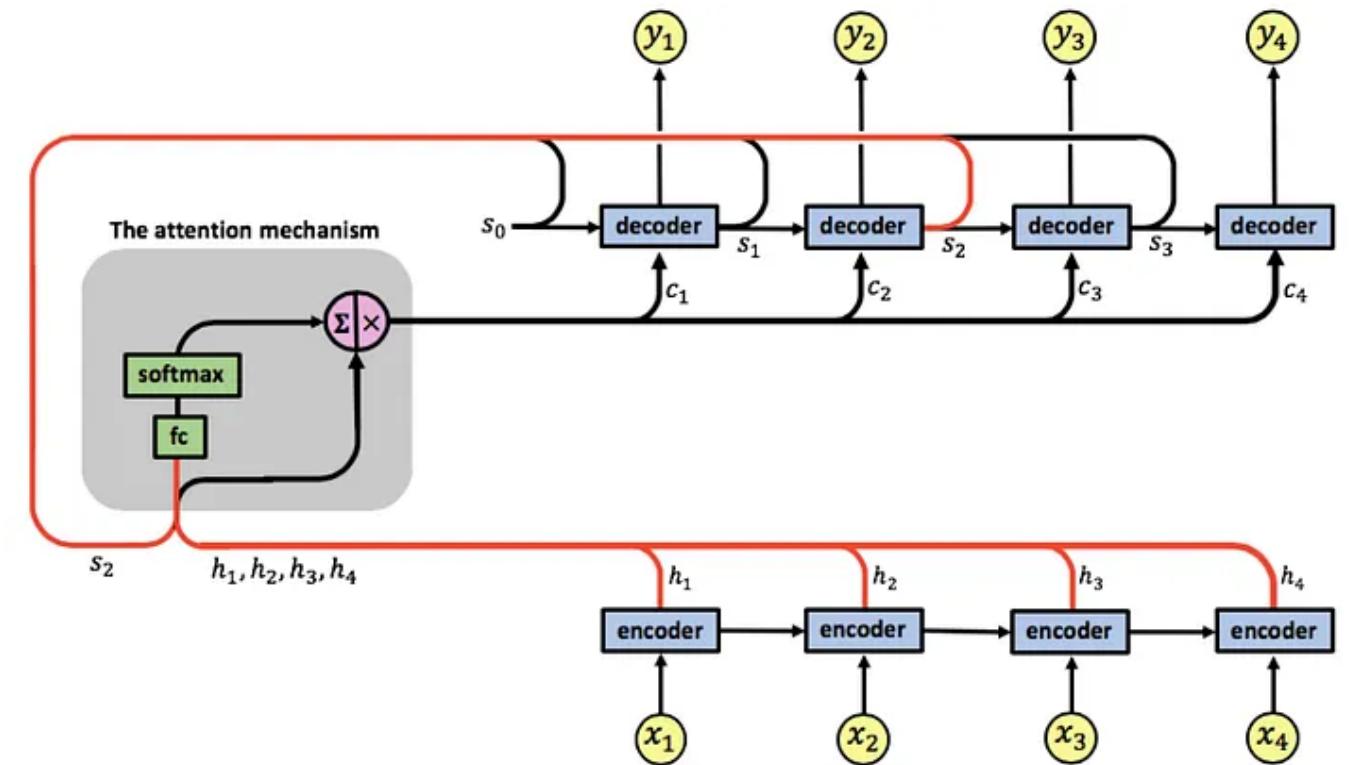
At the following time step the attention mechanism has as input the sequence  $[s_1, h_1], [s_1, h_2], [s_1, h_3], [s_1, h_4]$ .



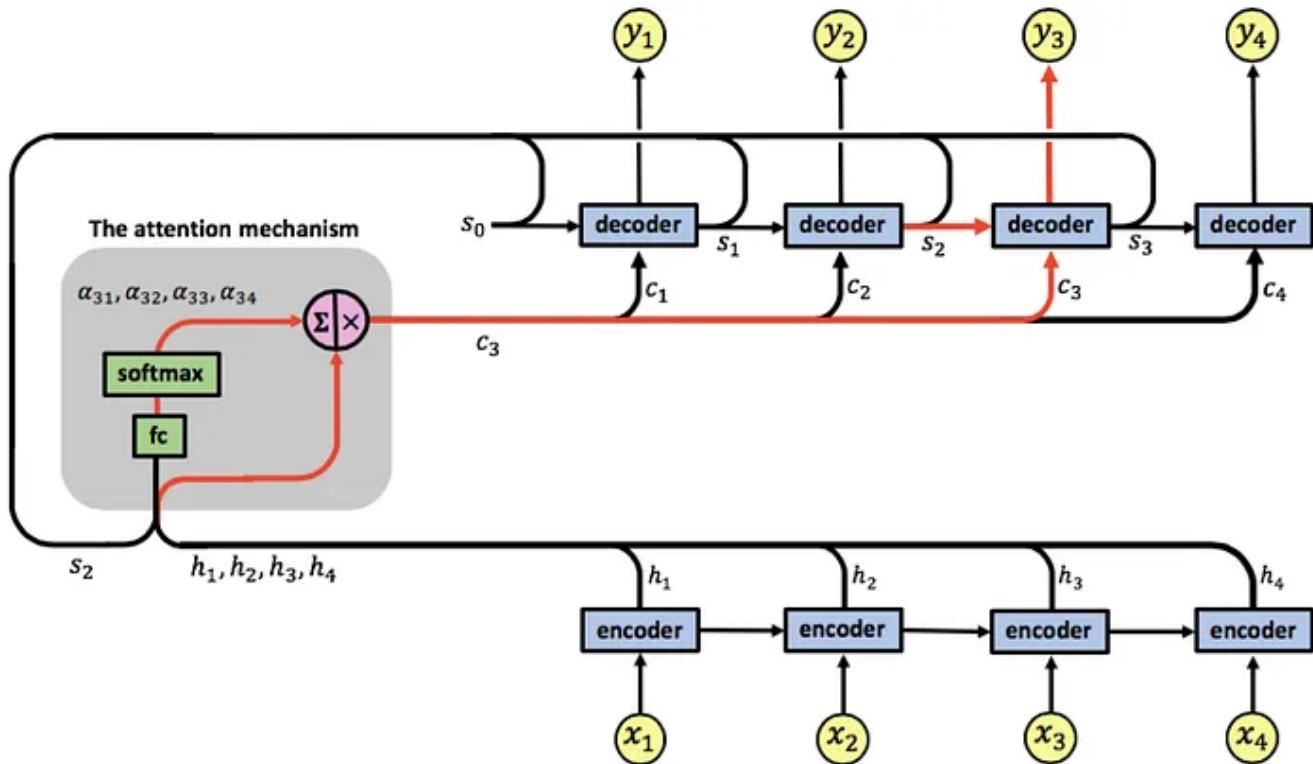
It computes a second set of attention weights  $\alpha_{21}, \alpha_{22}, \alpha_{23}, \alpha_{24}$  enabling the computation of the first context vector  $c_2$ . The decoder uses  $[s_1, c_2]$  and computes the second RNN output  $y_2$ .



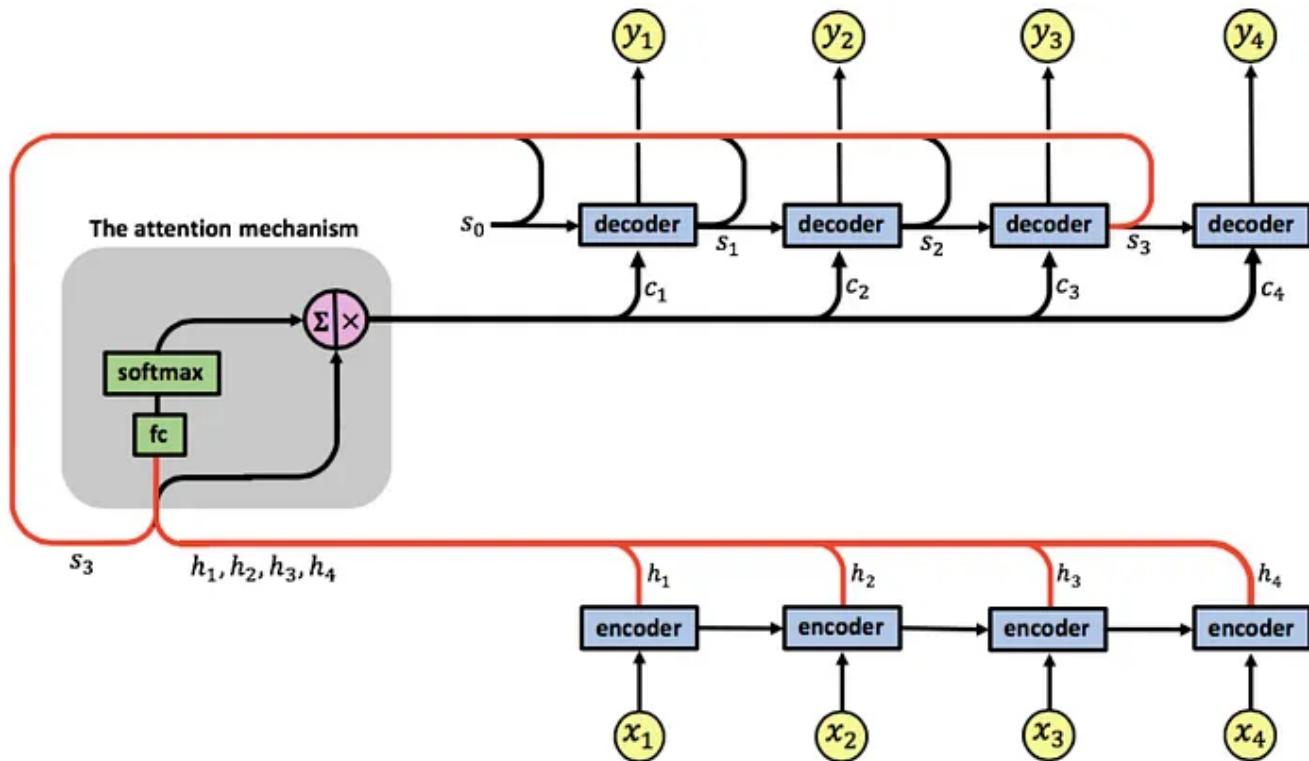
At the next time step the attention mechanism has an input sequence  $[s_2, h_1], [s_2, h_2], [s_2, h_3], [s_2, h_4]$ .



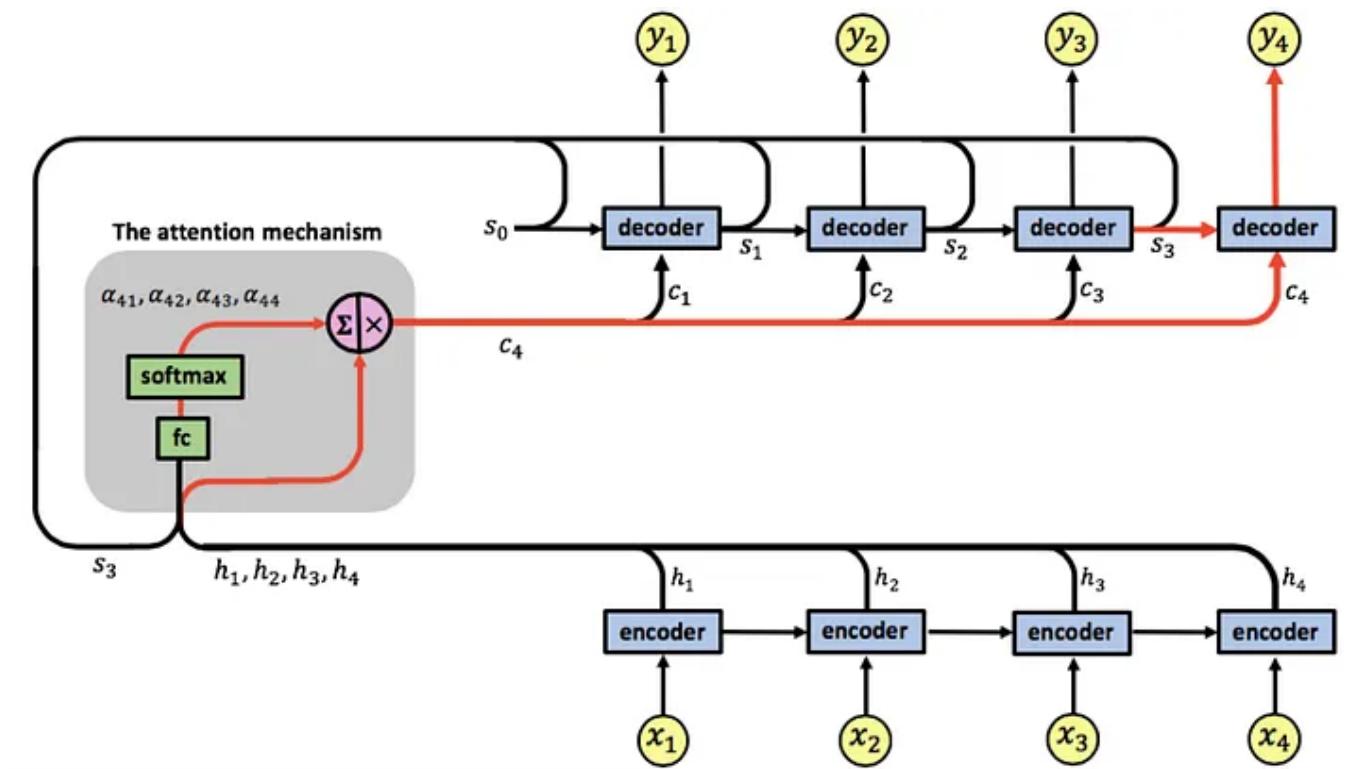
And computes a third set of attention weights  $\alpha_{31}, \alpha_{32}, \alpha_{33}, \alpha_{34}$  enabling the computation of the third context vector  $c_3$ . The decoder uses  $[s_2, c_3]$  and computes the following output  $y_3$ .



At the next time step the attention mechanism has an input sequence [s<sub>3</sub>, h<sub>1</sub>], [s<sub>3</sub>, h<sub>2</sub>], [s<sub>3</sub>, h<sub>3</sub>], [s<sub>3</sub>, h<sub>4</sub>].

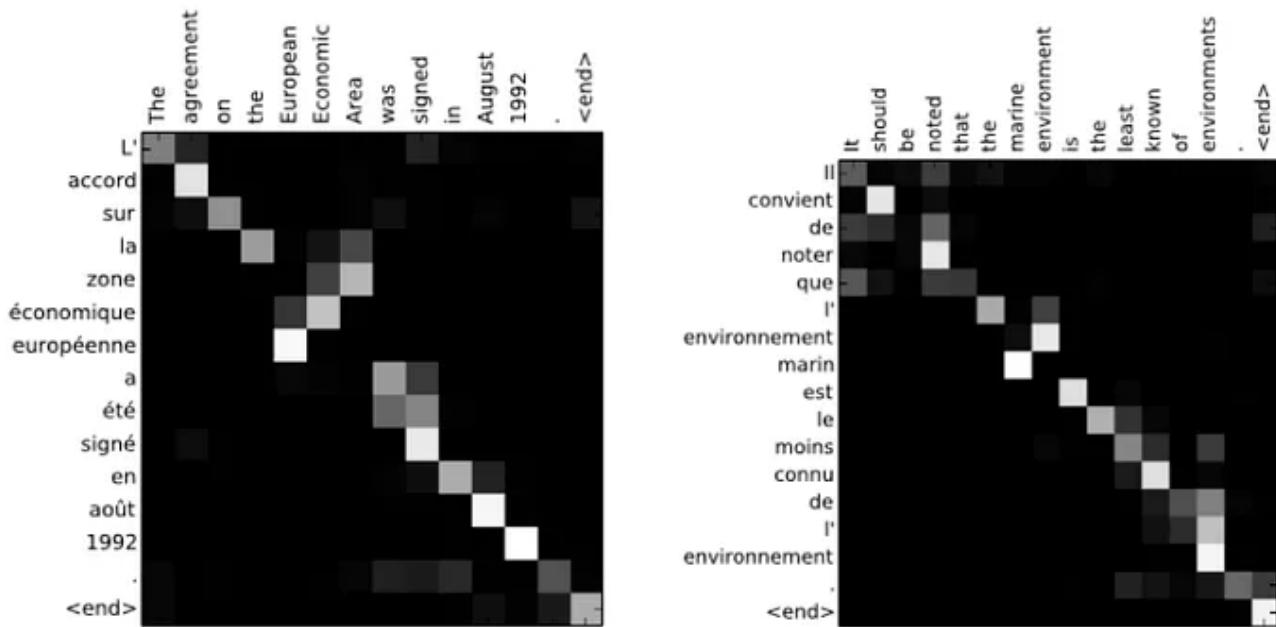


It computes a fourth set of attention weights  $\alpha_{41}, \alpha_{42}, \alpha_{43}, \alpha_{44}$  enabling the computation of the fourth context vector  $c_4$ . The decoder uses  $[s_3, c_4]$  and computes the final output  $y_4$ .



We end with an example from [1], the task is English-French machine translation. Below are two alignments found by the attention RNN. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively.

Each pixel shows the weight  $a_{ij}$  of the  $j$ -th source word and the  $i$ -th target word, in grayscale (0: black, 1: white).



We see how the attention mechanism allows the RNN to focus on a small part of the input sentence when outputting the translation. Notice how the larger attention parameters (given by the white pixels) connect corresponding parts of the English and French sentences enabling the network in [1] to achieve state of the art results.

**References:** [1] Neural Machine Translation By Jointly Learning To Align And Translate. Dzmitry Bahdanau, KyungHyun Cho and Yoshua Bengio.

[Machine Learning](#)[Deep Learning](#)[Neural Networks](#)[Artificial Intelligence](#)[Natural language processing](#)

## More from the list: "NLP"

Curated by [Himanshu Birla](#)



Jon Gi... in Towards Data ...

### Characteristics of Word Embeddings



· 11 min read · Sep 4, 2021



Jon Gi... in Towards Data ...

### The Word2vec Hyperparameters



· 6 min read · Sep 3, 2021



Jon Gi... in

### The Word2ve



· 15 min rea

[View list](#)

## Written by Nir Arbel

206 Followers · Writer for DataDrivenInvestor

Deep learning enthusiast

[Follow](#)


## More from Nir Arbel and DataDrivenInvestor



 Nir Arbel in DataDrivenInvestor

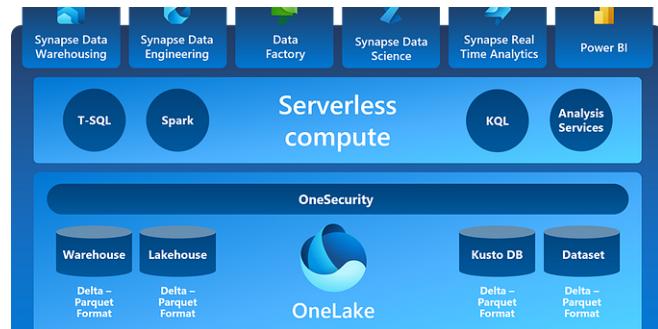
### How LSTM networks solve the problem of vanishing gradients

A simple, straightforward mathematical explanation

10 min read · Dec 21, 2018

 1.1K  12



 Gabe Araujo, M.Sc.  in DataDrivenInvestor

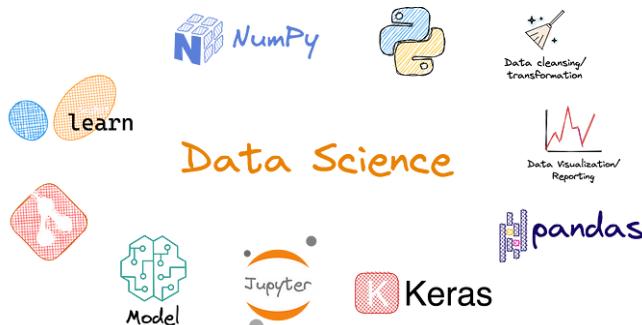
### Microsoft Fabric vs. Power BI: What's the Difference?

Microsoft Fabric vs. Power BI: Architecture, Capabilities, Data Governance, and Use Cases

10 min read · Aug 8

 286 



Avi Chawla in DataDrivenInvestor

## 250+ Python and Data Science Tips—Covering Pandas, NumPy,...

A self-curated collection of Python and Data Science tips to level up your data game.

6 min read · Jun 13

607 2

...

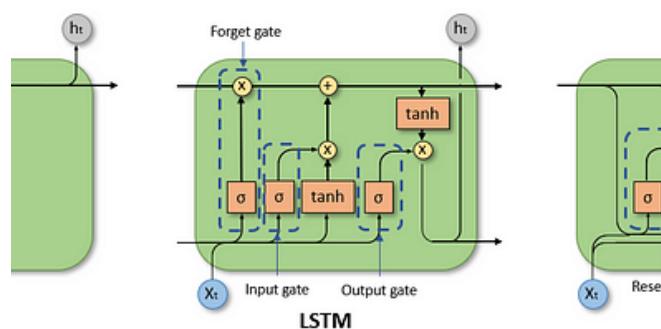
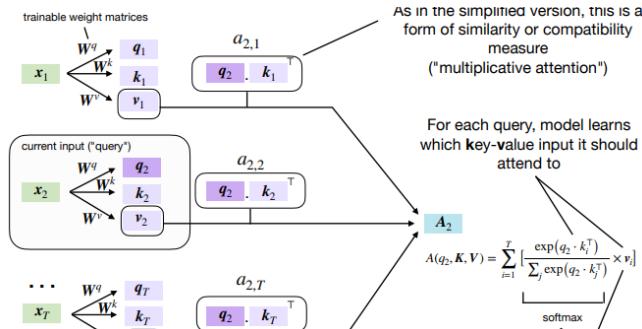
Nir Arbel in Analytics Vidhya

## Understanding Principle Component Analysis

Principle Component Analysis (PCA) is widely used in machine learning. PCA finds a...

5 min read · Oct 31, 2019

## 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



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



## Lists



### Predictive Modeling w/ Python

20 stories · 452 saves



### AI Regulation

6 stories · 138 saves



### Natural Language Processing

669 stories · 283 saves



### Practical Guides to Machine Learning

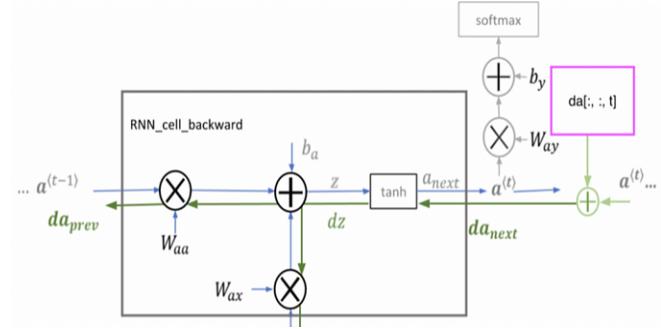
10 stories · 519 saves



Jim the AI Whisperer in The Generator

## The 19 tell-tale signs an article was written by AI

Avoid these AI cliches (or use them to detect AI writing in the wild)



Eugene Ku

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

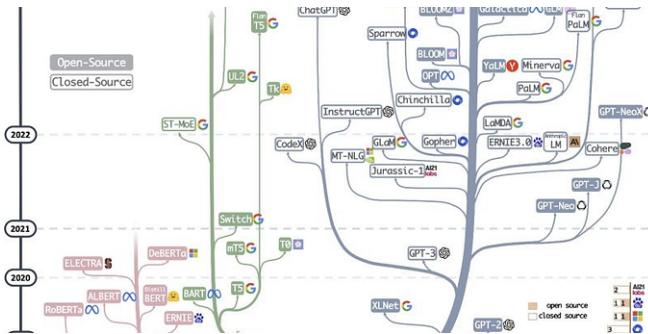
Hello everyone! Today, we are going to go through one of the most important...

◆ 16 min read · 5 days ago

4 min read · Aug 4

2.4K 55

+ ...



Haifeng Li

## A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372 0

+ ...

Anishnama

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

What is LSTM and How it works?

7 min read · Apr 28

153 2

+ ...

[See more recommendations](#)