**Attention Mechanism in Deep Learning**

* **The attention mechanism is one of the most valuable breakthroughs in Deep Learning research in the last decade. It has spawned the rise of so many recent breakthroughs in natural language processing (NLP), including the**[**Transformer architecture**](https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)**and**[**Google’s BERT**](https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)
* **Attention is one of the most influential ideas in the Deep Learning community. Even though this mechanism is now used in various problems like image captioning and others,it was initially designed in the context of Neural Machine Translation using Seq2Seq Models.**
* **So what’s wrong with seq2seq models?**
* **The seq2seq models is normally composed of an encoder-decoder architecture, where the encoder processes the input sequence and encodes/compresses/summarizes the information into a context vector (also called as the “thought vector”) of a fixed length. This representation is expected to be a good summary of the entire input sequence. The decoder is then initialized with this context vector, using which it starts generating the transformed output.**
* **A critical and apparent disadvantage of this fixed-length context vector design is the incapability of the system to remember longer sequences. Often is has forgotten the earlier parts of the sequence once it has processed the entire the sequence. The attention mechanism was born to resolve this problem.**

## What is Attention?

* **Attention is the cognitive process of selectively concentrating on one or a few things while ignoring others.**
* **A neural network is considered to be an effort to mimic human brain actions in a simplified manner. Attention Mechanism is also an attempt to implement the same action of selectively concentrating on a few relevant things, while ignoring others in deep neural networks.**

**When we think about the English word “Attention”, we know that it means directing your focus at something and taking greater notice. The Attention mechanism in Deep Learning is based off this concept of directing your focus, and it pays greater attention to certain factors when processing the data.**

**In broad terms, Attention is one component of a network’s architecture, and is in charge of managing and quantifying the interdependence:**

* **Between the input and output elements (General Attention)**
* **Within the input elements (Self-Attention)**

**Let me give you an example of how Attention works in a translation task. Say we have the sentence “*How was your day*”, which we would like to translate to the French version - “*Comment se passe ta journée*”. What the Attention component of the network will do for each word in the output sentence is**map **the important and relevant words from the input sentence and assign higher weights to these words, enhancing the accuracy of the output prediction.**

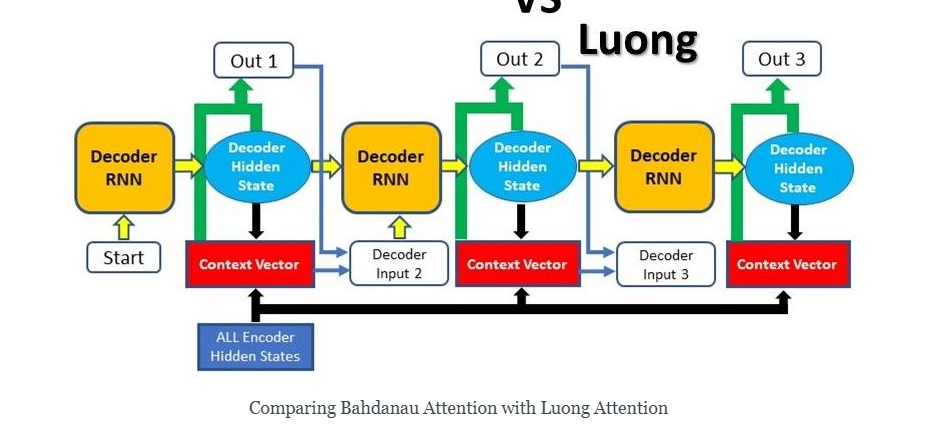
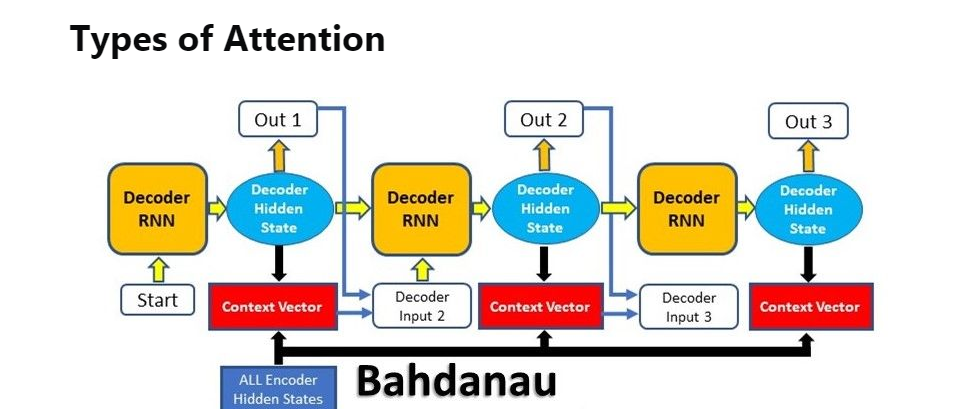
## C:\Users\dell\Desktop\1.jpg

## Attention in Sequence-to-Sequence Models

## C:\Users\dell\Desktop\aa.jpg

**The standard seq2seq model is generally unable to accurately process long input sequences, since only the last hidden state of the encoder RNN is used as the context vector for the decoder. On the other hand, the Attention Mechanism directly addresses this issue as it retains and utilises all the hidden states of the input sequence during the decoding process. It does this by creating a unique mapping between each time step of the decoder output to all the encoder hidden states. This means that for each output that the decoder makes, it has access to the entire input sequence and can selectively pick out specific elements from that sequence to produce the output.**

**Therefore, the mechanism allows the model to focus and place more “Attention” on the relevant parts of the input sequence as needed.**

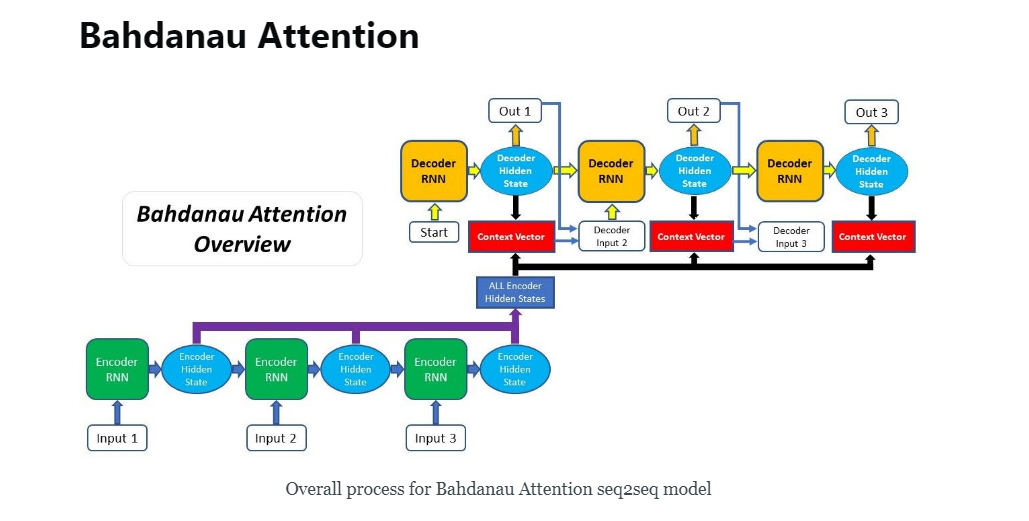
****

**Before we delve into the specific mechanics behind Attention, we must note that there are 2 different major types of Attention:**

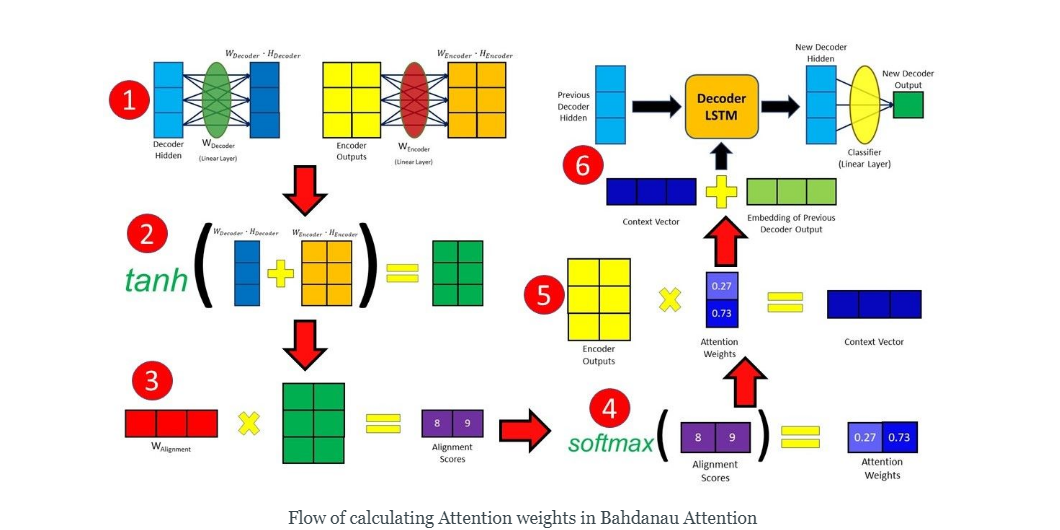
**1.**[**Bahdanau Attention**](https://blog.floydhub.com/attention-mechanism/#bahdanau-att)

**2.**[**Luong Attention**](https://blog.floydhub.com/attention-mechanism/#luong-att)

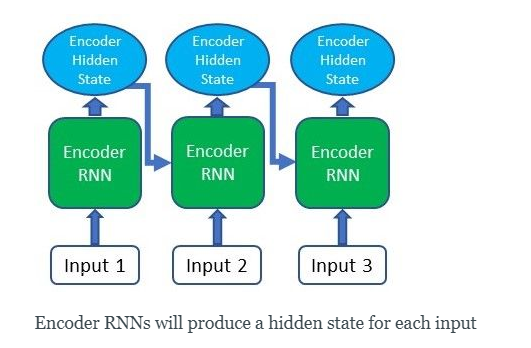
**While the underlying principles of Attention are the same in these 2 types, their differences lie mainly in their architectures and computations.**

****

1. [**Producing the Encoder Hidden States**](https://blog.floydhub.com/attention-mechanism/#bahdanau-att-step1)**- Encoder produces hidden states of each element in the input sequence**
2. [**Calculating Alignment Scores**](https://blog.floydhub.com/attention-mechanism/#bahdanau-att-step2)**between the previous decoder hidden state and each of the encoder’s hidden states are calculated (Note: The last encoder hidden state can be used as the first hidden state in the decoder)**
3. [**Softmaxing the Alignment Scores**](https://blog.floydhub.com/attention-mechanism/#bahdanau-att-step3)**- the alignment scores for each encoder hidden state are combined and represented in a single vector and subsequently softmaxed**
4. [**Calculating the Context Vector**](https://blog.floydhub.com/attention-mechanism/#bahdanau-att-step4)**- the encoder hidden states and their respective alignment scores are multiplied to form the context vector**
5. [**Decoding the Output**](https://blog.floydhub.com/attention-mechanism/#bahdanau-att-step5)**- the context vector is concatenated with the previous decoder output and fed into the Decoder RNN for that time step along with the previous decoder hidden state to produce a new output**
6. **The process (steps 2-5) repeats itself for each time step of the decoder until an token is produced or output is past the specified maximum length**



### 1. Producing the Encoder Hidden States



**For our first step, we’ll be using an RNN or any of its variants (e.g. LSTM, GRU) to encode the input sequence. After passing the input sequence through the encoder RNN, a hidden state/output will be produced for each input passed in. Instead of using only the hidden state at the final time step, we’ll be carrying forward all the hidden states produced by the encoder to the next step.**

**In the code implementation of the encoder above, we’re first embedding the input words into word vectors (assuming that it’s a language task) and then passing it through an LSTM. The encoder over here is exactly the same as a normal encoder-decoder structure without Attention.**

### 2. Calculating Alignment Scores

**For these next 3 steps, we will be going through the processes that happen in the Attention Decoder and discuss how the Attention mechanism is utilised. The class BahdanauDecoderLSTM defined below encompasses these 3 steps in the forward function.**

**After obtaining all of our encoder outputs, we can start using the decoder to produce outputs. At each time step of the decoder, we have to calculate the alignment score of each encoder output with respect to the decoder input and hidden state at that time step. The alignment score is the essence of the Attention mechanism, as it quantifies the amount of “Attention” the decoder will place on each of the encoder outputs when producing the next output.**

**The alignment scores for Bahdanau Attention are calculated using the hidden state produced by the decoder in the previous time step and the encoder outputs with the following equation:**

**C:\Users\dell\Desktop\aa.jpg**

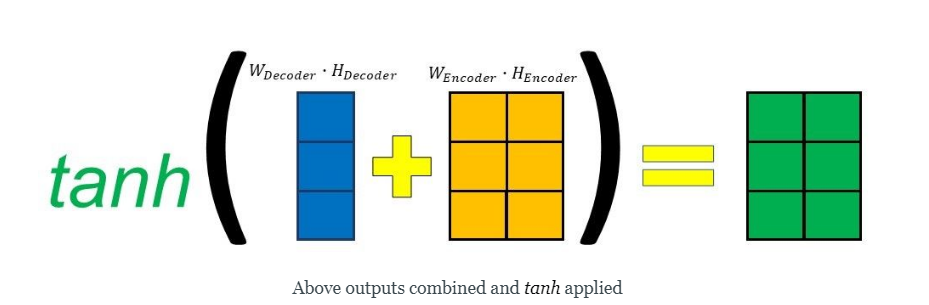
**The decoder hidden state and encoder outputs will be passed through their individual Linear layer and have their own individual trainable weights.**

**The decoder hidden state and encoder outputs will be passed through their individual Linear layer and have their own individual trainable weights.**

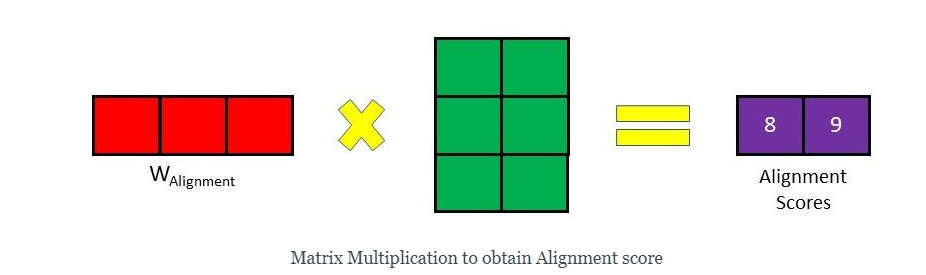


**the hidden size is 3 and the number of encoder outputs is 2.**

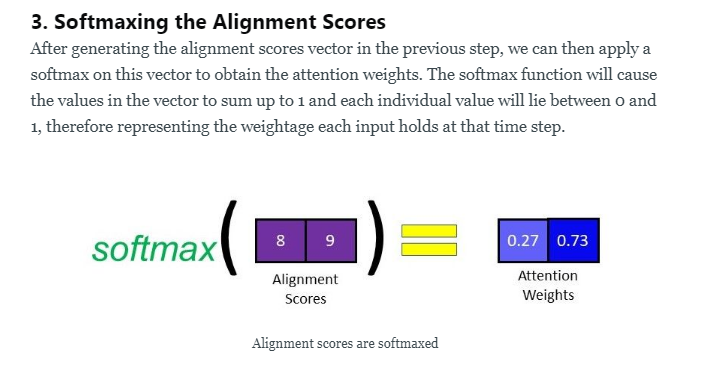
**Thereafter, they will be added together before being passed through a *tanh*activation function. The decoder hidden state is added to each encoder output in this case.**

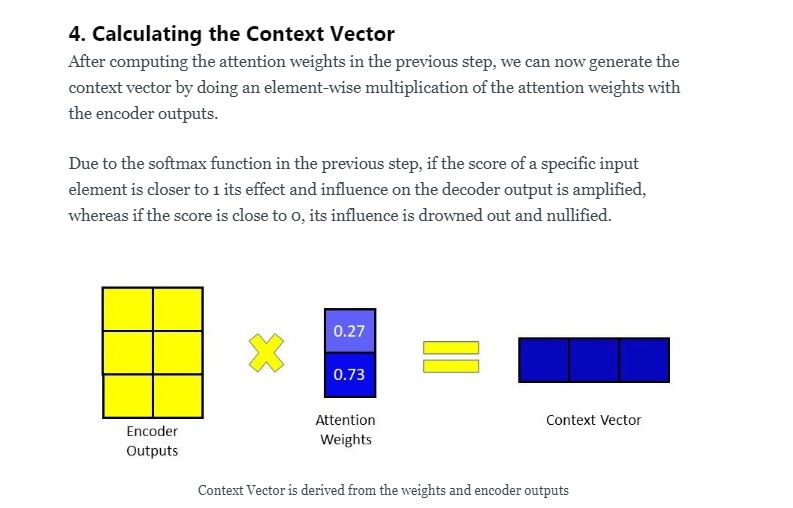


**Lastly, the resultant vector from the previous few steps will undergo matrix multiplication with a trainable vector, obtaining a final alignment score vector which holds a score for each encoder output.**

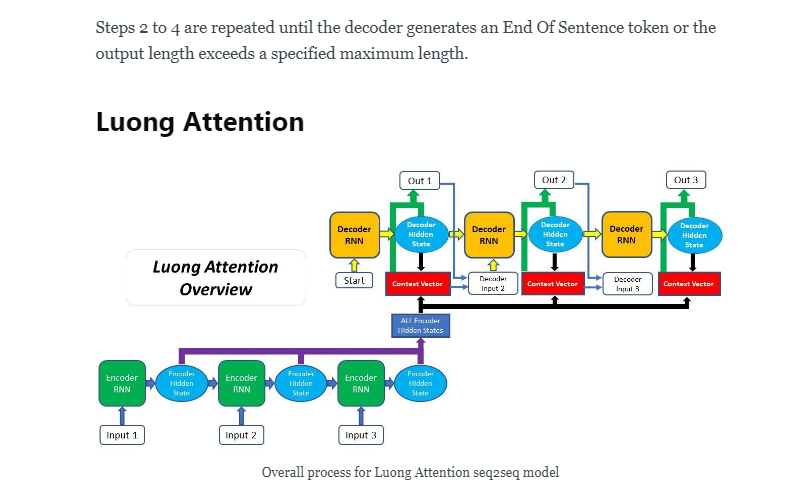


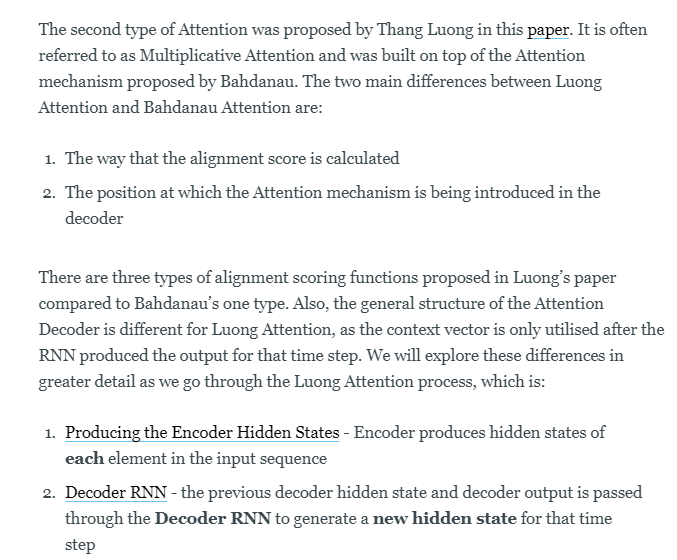
***Note: As there is no previous hidden state or output for the first decoder step, the last encoder hidden state and a Start Of String (<SOS>) token can be used to replace these two, respectively.***

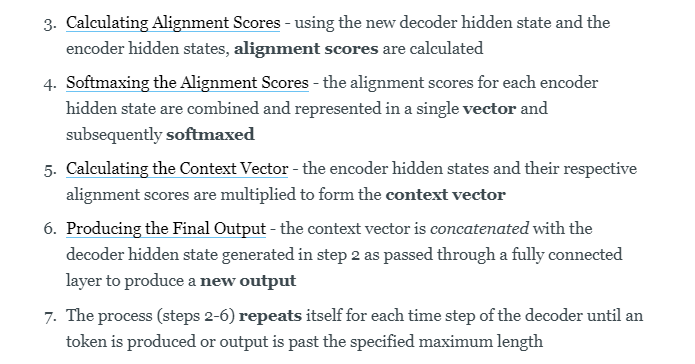


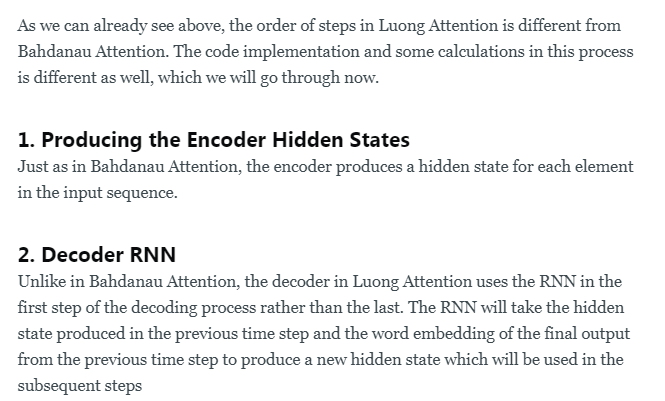


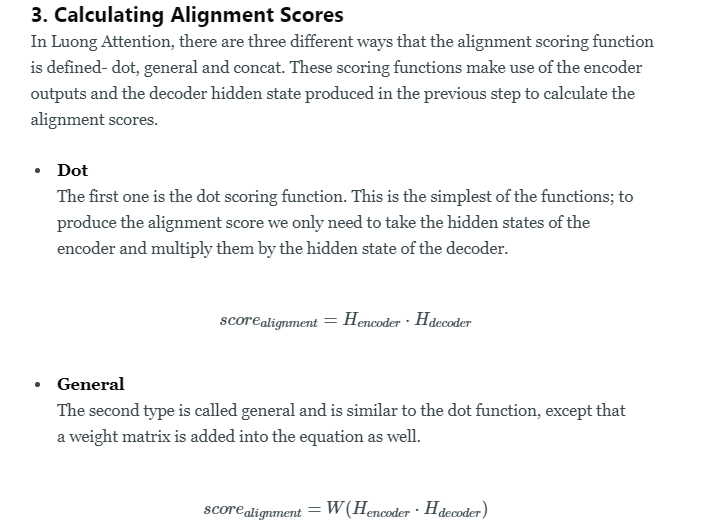


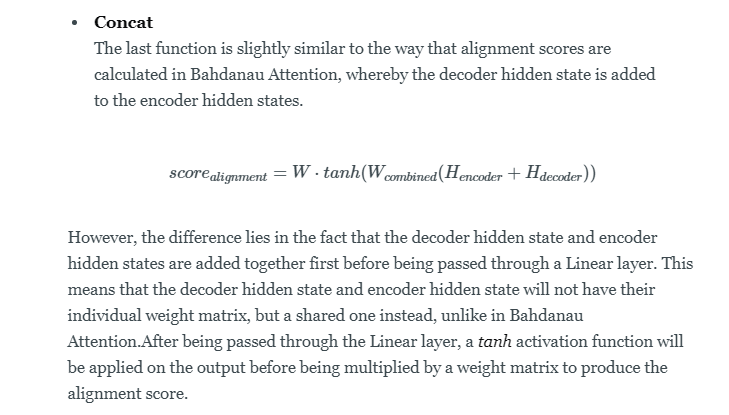














## How Attention Mechanism was Introduced in Deep Learning

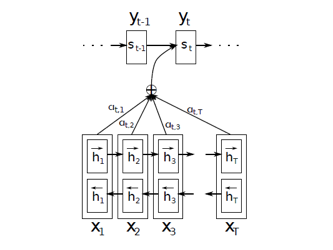
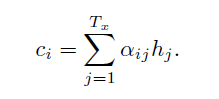
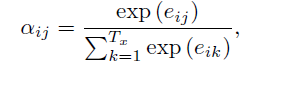
* **The attention mechanism emerged as an improvement over the encoder decoder-based**[**neural machine translation system**](https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)**in**[**natural language processing (NLP)**](https://courses.analyticsvidhya.com/courses/natural-language-processing-nlp?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)**. Later, this mechanism, or its variants, was used in other applications, including**[**computer vision**](https://courses.analyticsvidhya.com/courses/computer-vision-using-deep-learning-version2?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)**, speech processing, etc.**
* **Before**[Bahdanau et al](https://arxiv.org/abs/1409.0473" \t "_blank)[**proposed the first Attention model in 2015**](https://arxiv.org/abs/1409.0473)**, neural machine translation was based on encoder-decoder**[**RNNs**](https://www.analyticsvidhya.com/blog/2019/01/fundamentals-deep-learning-recurrent-neural-networks-scratch-python/?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)**/**[**LSTMs**](https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning)**. Both encoder and decoder are stacks of LSTM/RNN units. It works in the two following steps:**

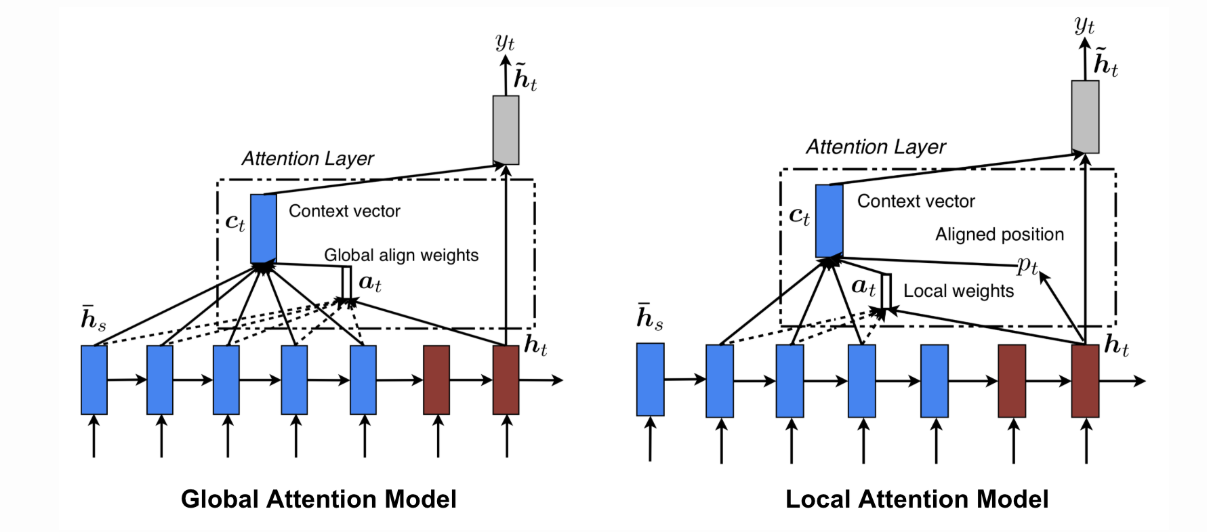
**1.The encoder LSTM is used to process the entire input sentence and encode it into a context vector, which is the last hidden state of the LSTM/RNN. This is expected to be a good summary of the input sentence. All the intermediate states of the encoder are ignored, and the final state id supposed to be the initial hidden state of the decoder**

**2.The decoder LSTM or RNN units produce the words in a sentence one after another**

* **In short, there are two RNNs/LSTMs. One we call the encoder – this reads the input sentence and tries to make sense of it, before summarizing it. It passes the summary (context vector) to the decoder which translates the input sentence by just seeing it.**
* **The main drawback of this approach is evident. If the encoder makes a bad summary, the translation will also be bad. And indeed it has been observed that the encoder creates a bad summary when it tries to understand longer sentences. It is called the long-range dependency problem of RNN/LSTMs.**
* **RNNs cannot remember longer sentences and sequences due to the vanishing/exploding gradient problem. It can remember the parts which it has just seen. Even**[**Cho et al (2014)**](https://arxiv.org/abs/1406.1078)**, who proposed the encoder-decoder network, demonstrated that**the performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases.
* **Although an LSTM is supposed to capture the long-range dependency better than the RNN, it tends to become forgetful in specific cases. Another problem is that there is no way to give more importance to some of the input words compared to others while translating the sentence.**
* **Now, let’s say, we want to predict the next word in a sentence, and its context is located a few words back. Here’s an example – “Despite originally being from Uttar Pradesh, as he was brought up in Bengal, he is more comfortable in Bengali”. In these groups of sentences, if we want to predict the word “Bengali”, the phrase “brought up” and “Bengal”- these two should be given more weight while predicting it. And although Uttar Pradesh is another state’s name, it should be “ignored”.**
* **So, whenever the proposed model generates a sentence, it searches for a set of positions in the encoder hidden states where the most relevant information is available. This idea is called ‘Attention’.**

### Understanding the Attention Mechanism

* ****
* **This is the diagram of the Attention model shown in [Bahdanau’s paper](https://arxiv.org/abs/1409.0473" \t "_blank). The Bidirectional LSTM used here generates a sequence of annotat*ions (h1, h2,….., hTx)* for each input sentence. All the vectors h1,h2.., etc., used in their work are basically the concatenation of forward and backward hidden states in the encoder.**
* **[attention deep learning](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image9.png)**
* **To put it in simple terms, all the vectors h1,h2,h3…., hTx are representations of Tx number of words in the input sentence. In the simple encoder and decoder model, only the last state of the encoder LSTM was used (hTx in this case) as the context vector.**
* **Now, the question is how should the weights be calculated? Well, the weights are also learned by a feed-forward neural network and I’ve mentioned their mathematical equation below.**
* **The context vector ci for the output word yi is generated using the weighted sum of the annotations:**
* **[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image8.png)**
* **The weights αij are computed by a softmax function given by the following equation:**
* **[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image11.png)**
* **[https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image10.png](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image10.png)**
* **eij is the output score of a feedforward neural network described by the function a that attempts to capture the alignment between input at j and output at i.**
* **Basically, if the encoder produces Tx number of “annotations” (the hidden state vectors) each having dimension d, then the input dimension of the feedforward network is (Tx , 2d) (assuming the previous state of the decoder also has d dimensions and these two vectors are concatenated). This input is multiplied with a matrix Wa of (2d, 1) dimensions (of course followed by addition of the bias term) to get scores eij (having a dimension (Tx , 1)).**
* **On the top of these eij scores, a tan hyperbolic function is applied followed by a softmax to get the normalized alignment scores for output j:**
* **E = I [Tx\*2d] \* Wa [2d \* 1] + B[Tx\*1]**
* **α = softmax(tanh(E))**
* **C= IT \* α**
* **So, α is a (Tx, 1) dimensional vector and its elements are the weights corresponding to each word in the input sentence.**
* **Let α is [0.2, 0.3, 0.3, 0.2] and the input sentence is “I am doing it”. Here, the context vector corresponding to it will be:**
* **C=0.2\*I”I” + 0.3\*I”am”  + 0.3\*I”doing” + + 0.3\*I”it”  [Ix is the hidden state corresponding to the word x]**
* **Different researchers have tried different techniques for score calculation. There are different variants of Attention model(s) according to how the score, as well as the context vector, are calculated. There are other variants also, which we will discuss next.**
* **The term “global” Attention is appropriate because all the inputs are given importance. Originally, the Global Attention (defined by Luong et al 2015) had a few subtle differences with the Attention concept we discussed previously.**
* **The differentiation is that it considers all the hidden states of both the encoder LSTM and decoder LSTM to calculate a “variable-length context vector ct, whereas Bahdanau et al. used the previous hidden state of the unidirectional decoder LSTM and all the hidden states of the encoder LSTM to calculate the context vector.**
* **In encoder-decoder architectures, the score generally is a function of the encoder and the decoder hidden states. Any function is valid as long as it captures the relative importance of the input words with respect to the output word.**
* **When a “global” Attention layer is applied, a lot of computation is incurred. This is because all the hidden states must be taken into consideration, concatenated into a matrix, and multiplied with a weight matrix of correct dimensions to get the final layer of the feedforward connection.**
* **So, as the input size increases, the matrix size also increases. In simple terms, the number of nodes in the feedforward connection increases and in effect it increases computation.**
* **Soft Attention is the global Attention where all image patches are given some weight; but in hard Attention, only one image patch is considered at a time.**
* **But local Attention is not the same as the hard Attention used in the image captioning task. On the contrary, it is a blend of both the concepts, where instead of considering all the encoded inputs, only a part is considered for the context vector generation. This not only avoids expensive computation incurred in soft Attention but is also easier to train than hard Attention.**
* **How can this be achieved in the first place? Here, the model tries to predict a position pt in the sequence of the embeddings of the input words. Around the position pt, it considers a window of size, say, 2D. Therefore, the context vector is generated as a weighted average of the inputs in a position [pt – D,pt + D] where D is empirically chosen.**
* **Furthermore, there can be two types of alignments:**
* **Monotonic alignment, where pt is set to t, assuming that at time t, only the information in the neighborhood of t matters**
* **Predictive alignment where the model itself predicts the alignment position as follows:**
* **[https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image25.png](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image25.png)**
* **where ‘Vp’ and ‘Wp’ are the model parameters that are learned during training and ‘S’ is the source sentence length. Clearly, pt ε [0,S].**
* **The figures below demonstrate the difference between the Global and Local Attention mechanism. Global Attention considers all hidden states (blue) whereas local Attention considers only a subset:**

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image26.png)

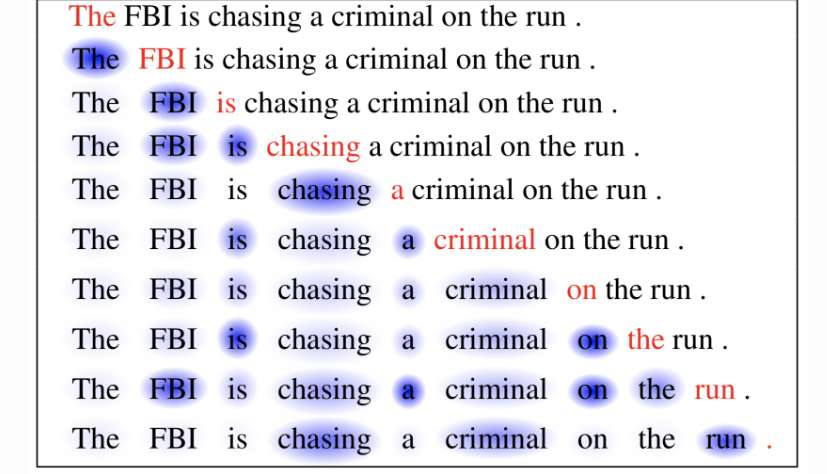
## Transformers – Attention is All You Need

**The paper named “**[Attention is All You Need](https://arxiv.org/abs/1706.03762)**” by Vaswani et al is one of the most important contributions to Attention so far. They have redefined Attention by providing a very generic and broad definition of Attention based on key, query, and values. They have referenced another concept called multi-headed Attention. Let’s discuss this briefly.**

**First, let’s define what “self-Attention” is. Cheng et al, in their paper named “**[Long Short-Term Memory-Networks for Machine Reading](https://arxiv.org/abs/1601.06733)**”, defined self-Attention as the mechanism of relating different positions of a single sequence or sentence in order to gain a more vivid representation.**

**Machine reader is an algorithm that can automatically understand the text given to it.  We have taken the below picture from the paper. The red words are read or processed at the current instant, and the blue words are the memories. The different shades represent the degree of memory activation.**

**When we are reading or processing the sentence word by word, where previously seen words are also emphasized on, is inferred from the shades, and this is exactly what self-Attention in a machine reader does.**

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image2.png)

**Previously, to calculate the Attention for a word in the sentence, the mechanism of score calculation was to either use a dot product or some other function of the word with the hidden state representations of the previously seen words. In this paper, a fundamentally same but a more generic concept altogether has been proposed.**

**Let’s say we want to calculate the Attention for the word “chasing”. The mechanism would be to take a dot product of the embedding of “chasing” with the embedding of each of the previously seen words like “The”, “FBI”, and “is”.**

**Now, according to the generalized definition, each embedding of the word should have three different vectors corresponding to it, namely Key, Query, and Value. We can easily derive these vectors using matrix multiplications.**

**Whenever we are required to calculate the Attention of a target word with respect to the input embeddings, we should use the Query of the target and the Key of the input to calculate a matching score, and these matching scores then act as the weights of the Value vectors during summation.**

**Now, you might ask what these Key, Query and Value vectors are. These are basically abstractions of the embedding vectors in different subspaces. Think of it in this way: you raise a query; the query hits the key of the input vector. The Key can be compared with the memory location read from, and the value is the value to be read from the memory location. Simple, right?**

**If the dimension of the embeddings is (D, 1) and we want a Key vector of dimension (D/3, 1), we must multiply the embedding by a matrix Wk of dimension (D/3, D). So, the key vector becomes K=Wk\*E. Similarly, for Query and Value vectors, the equations will be Q=Wq\*E, V=Wv\*E (E is the embedding vector of any word).**

**Now, to calculate the Attention for the word “chasing”, we need to take the dot product of the query vector of the embedding of “chasing” to the key vector of each of the previous words, i.e., the key vectors corresponding to the words “The”, “FBI” and “is”. Then these values are divided by D (the dimension of the embeddings) followed by a softmax operation. So, the operations are respectively:**

**softmax(Q”chasing” . K”The” / D)**

**softmax(Q”chasing” .K”FBI” / D)**

**softmax(Q”chasing” . K”is” / D)**

**Basically, this is a function f(Qtarget, Kinput) of the query vector of the target word and the key vector of the input embeddings. It doesn’t necessarily have to be a dot product of Q and K. Anyone can choose a function of his/her own choice.**

**Next, let’s say the vector thus obtained is [0.2, 0.5, 0.3]. These values are the “alignment scores” for the calculation of Attention. These alignment scores are multiplied with the value vector of each of the input embeddings and these weighted value vectors are added to get the context vector:**

**C”chasing”= 0.2 \* VThe + 0.5\* V”FBI” + 0.3 \* V”is”**

**Practically, all the embedded input vectors are combined in a single matrix X, which is multiplied with common weight matrices Wk, Wq, Wv to get K, Q and V matrices respectively. Now the compact equation becomes:**

**Z=Softmax(Q\*KT/D)V**

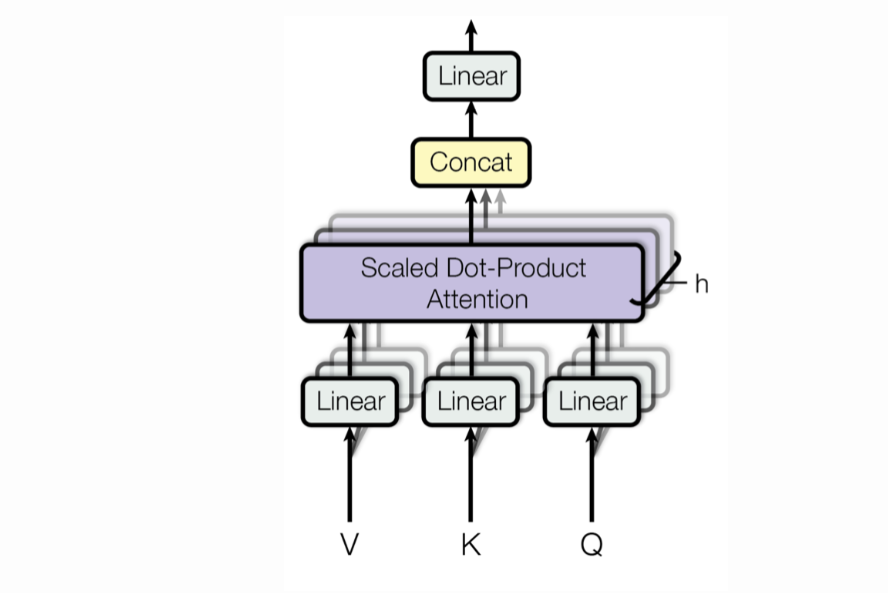
**Therefore, the context vector is a function of Key, Query and Value F(K, Q, V).**

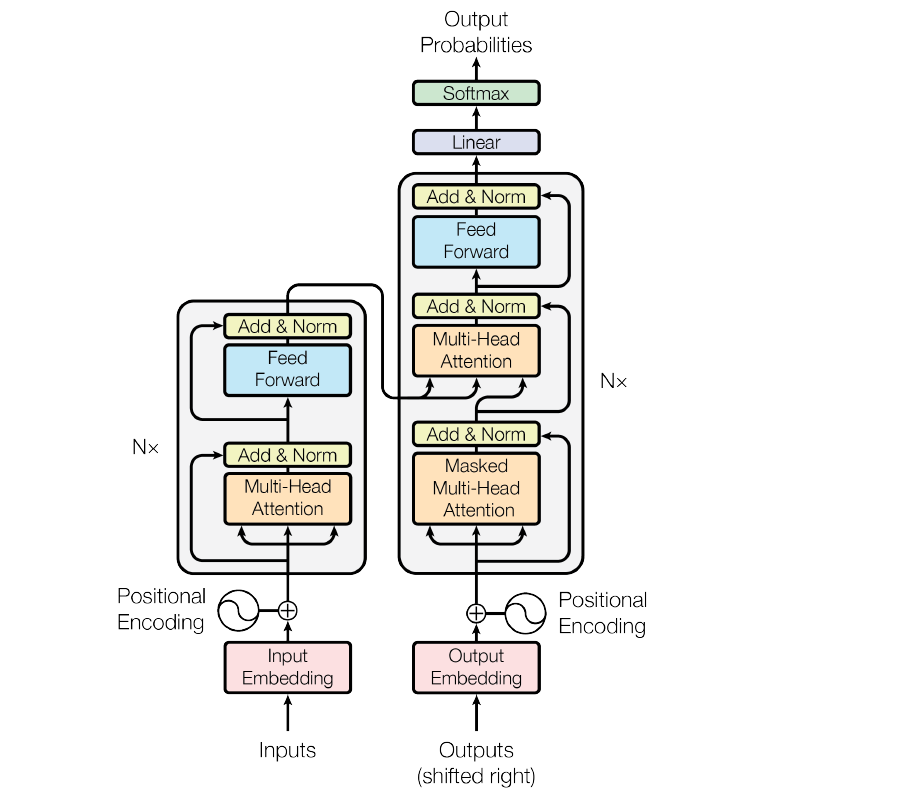
**The Bahdanau Attention or all other previous works related to Attention are the special cases of the Attention Mechanisms described in this work. The salient feature/key highlight is that the single embedded vector is used to work as Key, Query and Value vectors simultaneously.**

**In multi-headed Attention, matrix X is multiplied by different Wk, Wq and Wv matrices to get different K, Q and V matrices respectively. And we end up with different Z matrices, i.e., embedding of each input word is projected into different “representation subspaces”.**

**In, say, 3-headed self-Attention, corresponding to the “chasing” word, there will be 3 different Z matrices also called “Attention Heads”. These Attention heads are concatenated and multiplied with a single weight matrix to get a single Attention head that will capture the information from all the Attention heads.**

**The picture below depicts the multi-head Attention. You can see that there are multiple Attention heads arising from different V, K, Q vectors, and they are concatenated:**

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image3.png)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/11/image4.png)

**This image above is the transformer architecture. We see that something called ‘positional encoding’ has been used and added with the embedding of the inputs in both the encoder and decoder.**

**The models that we have described so far had no way to account for the order of the input words. They have tried to capture this through positional encoding.**This mechanism adds a vector to each input embedding, and all these vectors follow a pattern that helps to determine the position of each word, or the distances between different words in the input.

**As shown in the figure, on top of this positional encoding + input embedding layer, there are two sublayers:**

1. **In the first sublayer, there is a multi-head self-attention layer. There is an additive residual connection from the output of the positional encoding to the output of the multi-head self-attention, on top of which they have applied a layer normalization layer. The layer normalization is a technique**(Hinton, 2016)**similar to batch normalization where instead of considering the whole minibatch of data for calculating the normalization statistics, all the hidden units in the same layer of the network have been considered in the calculations. This overcomes the drawback of estimating the statistics for the summed input to any neuron over a minibatch of the training samples. Thus, it is convenient to use in RNN/LSTM**
2. **In the second sublayer, instead of the multi-head self-attention, there is a feedforward layer (as shown), and all other connections are the same**

**On the decoder side, apart from the two layers described above, there is another layer that applies multi-head Attention on top of the encoder stack. Then, after a sublayer followed by one linear and one softmax layer, we get the output probabilities from the decoder.**

**The central idea behind Attention**

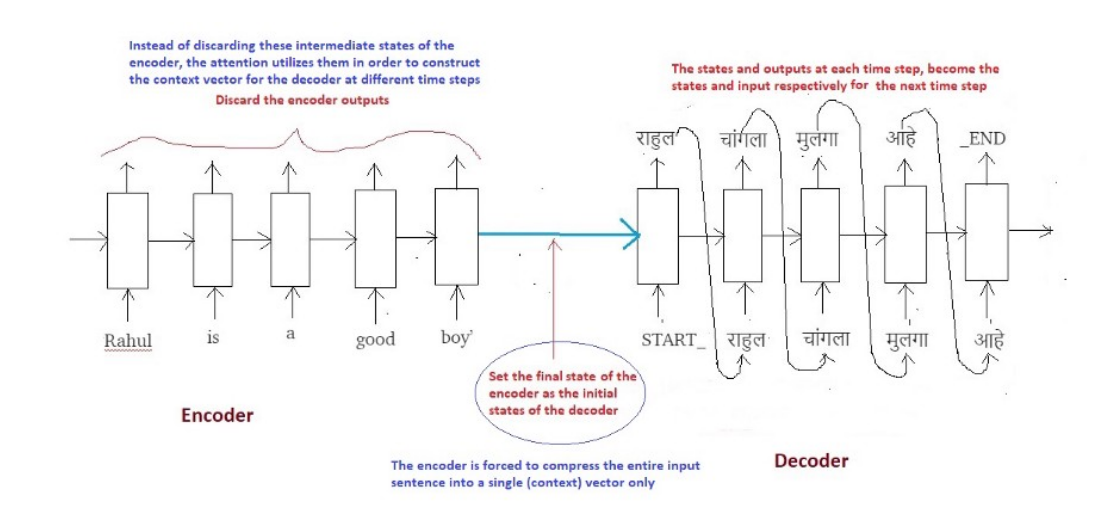
**For the illustrative purposes, I will borrow the same example that I used to explain Seq2Seq models** **in my previous**[**blog**](https://towardsdatascience.com/word-level-english-to-marathi-neural-machine-translation-using-seq2seq-encoder-decoder-lstm-model-1a913f2dc4a7)**.**

**Input (English) Sentence: “Rahul is a good boy”**

**Target (Marathi) Sentence: “राहुल चांगला मुलगा आहे”**

**The only change will be that instead of an LSTM layer that I used in my previous explanation, here I will use a GRU layer. The reason being that LSTM has two internal states (hidden state and cell state) and GRU has only one internal state (hidden state). This will help simplify the the concept and explanation.**

**Recall the below diagram in which I summarized the entire process procedure of Seq2Seq modelling.**

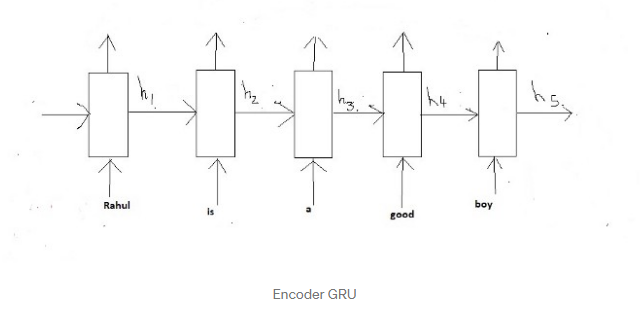
****

**In the traditional Seq2Seq model, we discard all the intermediate states of the encoder and use only its final states (vector) to initialize the decoder. This technique works good for smaller sequences, however as the length of the sequence increases, a single vector becomes a bottleneck and it gets very difficult to summarize long sequences into a single vector. This observation was made empirically as it was noted that the performance of the system decreases drastically as the size of the sequence increases.**

**The central idea behind Attention is not to throw away those intermediate encoder states but to utilize all the states in order to construct the context vectors required by the decoder to generate the output sequence.**

**3. Why the name Attention?**

**Let’s name each of the intermediate states of the encoder as below:**

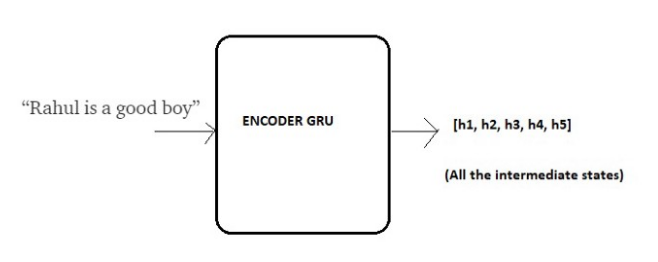
****

**Notice that since we are using a GRU instead of an LSTM, we only have a single state at each time step and not two states, which thus helps to simplify the illustration. Also note that attention is useful specially in case of longer sequences but for the sake of simplicity we will consider the same above example for illustration.**

**Recall that these states (h1 to h5) are nothing but vectors of fixed length. To develop some intuition think of these states as vectors which store local information within the sequence. For example;**

**h1 stores the information present in the start of the sequence (words like ‘Rahul’ and ‘is’) while h5 stores the information present in the later part of the sequence (words like ‘good’ and ‘boy’).**

**Lets represent our Encoder GRU with the below simplified diagram:**

****

**Now the idea is to utilize all of these local information collectively in order to decide the next sequence while decoding the target sentence.**

**Imagine you are translating “Rahul is a good boy” to “राहुल चांगला मुलगा आहे”. Ask yourself, how do you do it in your mind?**

**When you predict “राहुल”, its obvious that this name is the result of the word “Rahul” present in the input English sentence regardless of the rest of the sentence. We say that while predicting “राहुल”, we pay more attention to the word “Rahul” in the input sentence.**

**Similarly while predicting the word “चांगला”, we pay more attention to the word “good” in the input sentence.**

**Similarly while predicting the word “मुलगा”, we pay more attention to the word “boy” in the input sentence. And so on..**

Hence the name **“ATTENTION”.**

As human beings we are quickly able to understand these mappings between different parts of the input sequence and corresponding parts of the output sequence. However its not that straight forward for artificial neural network to automatically detect these mappings.

Thus the Attention mechanism is developed to **“learn”** these mappings through Gradient Descent and Back-propagation.

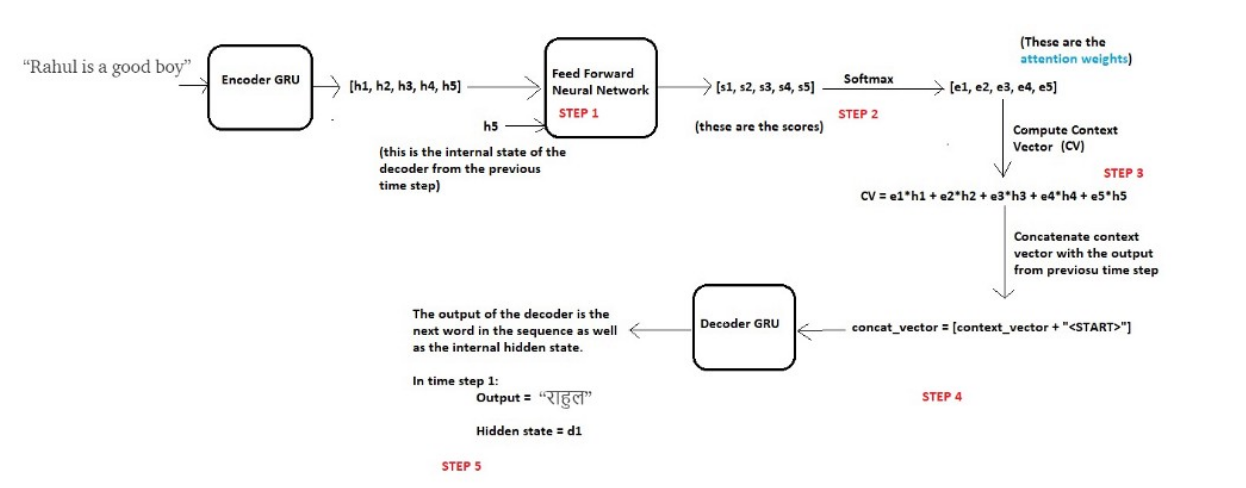
# 4. How does Attention work?

Let’s get technical and dive into the nitty gritty of Attention mechanism.

## Decoding at time step 1

Continuing the above example, let’s say we now want our decoder to start predicting the first word of the target sequence i.e. “राहुल”

At time step 1, we can break the entire process into **five steps** as below:

****

Before we start decoding, we first need to encode the input sequence into a set of internal states (in our case h1, h2, h3, h4 and h5).

Now the hypothesis is that, the next word in the output sequence is dependent on the current state of the decoder (decoder is also a GRU) as well as on the hidden states of the encoder. Thus at each time step, we consider these two things and follow the below steps:

**Step 1 — Compute a score each encoder state**

Since we are predicting the first word itself, the decoder does not have any current internal state. For this reason, we will consider the last state of the encoder (i.e. h5) as the previous decoder state.

Now using these two components (all the encoder states and the current state of the decoder), we will train a simple feed forward neural network.

Why?

Recall we are trying to predict the first word in the target sequence i.e. “राहुल”. As per the idea behind attention, we do not need all the encoder states to predict this word, but we need those encoder states which store information about the word “Rahul” in the input sequence.

As discussed previously these intermediate encoder states store the local information of the input sequence. So it is highly likely that the information of the word “Rahul” will be present in the states, let’s say, h1 and h2.

Thus we want our decoder to pay more attention to the states h1 and h2 while paying less attention to the remaining states of the encoder.

For this reason we train a feed forward neural network which will **learn** to identify relevant encoder states by generating a high score for the states for which attention is to be paid while low score for the states which are to be ignored.

Let s1, s2, s3, s4 and s5 be the scores generated for the states h1, h2, h3, h4 and h5 correspondingly. Since we assumed that we need to pay more attention to the states h1 and h2 and ignore h3, h4 and h5 in order to predict “राहुल”, we expect the above neural to generate scores such that s1 and s2 are high while s3, s4 and s5 are relatively low.

**Step 2— Compute the attention weights**

Once these scores are generated, we apply a softmax on these scores to produce the attention weights e1, e2, e3 ,e4 and e5 as shown above. The advantage of applying softmax is as below:

a) All the weights lie between 0 and 1, i.e., 0 ≤ e1, e2, e3, e4, e5 ≤ 1

b) All the weights sum to 1, i.e., e1+e2+3+e4+e5 = 1

Thus we get a nice probabilistic interpretation of the attention weights.

In our case we would expect values like below: (just for intuition)

e1 = 0.75, e2 = 0.2, e3 = 0.02, e4 = 0.02, e5 = 0.01

This means that while predicting the word “राहुल”, the decoder needs to put more attention on the states h1 and h2 (since values of e1 and e2 are high) while ignoring the states h3, h4 and h5 (since the values of e3, e4 and e5 are very small).

**Step 3— Compute the context vector**

Once we have computed the attention weights, we need to compute the context vector (thought vector) which will be used by the decoder in order to predict the next word in the sequence. Calculated as follows:

context\_vector = e1 \* h1 + e2 \* h2 + e3 \* h3 + e4 \* h4 + e5 \* h5

Clearly if the values of e1 and e2 are high and those of e3, e4 and e5 are low then the context vector will contain more information from the states h1 and h2 and relatively less information from the states h3, h4 and h5.

**Step 4— Concatenate context vector with output of previous time step**

Finally the decoder uses the below two input vectors to generate the next word in the sequence

a) The context vector

b) The output word generated from the previous time step.

We simply concatenate these two vectors and feed the merged vector to the decoder. **Note that for the first time step, since there is no output from the previous time step, we use a special <START> token for this purpose**. This concept is already discussed in detail in my previous [blog](https://towardsdatascience.com/word-level-english-to-marathi-neural-machine-translation-using-seq2seq-encoder-decoder-lstm-model-1a913f2dc4a7).

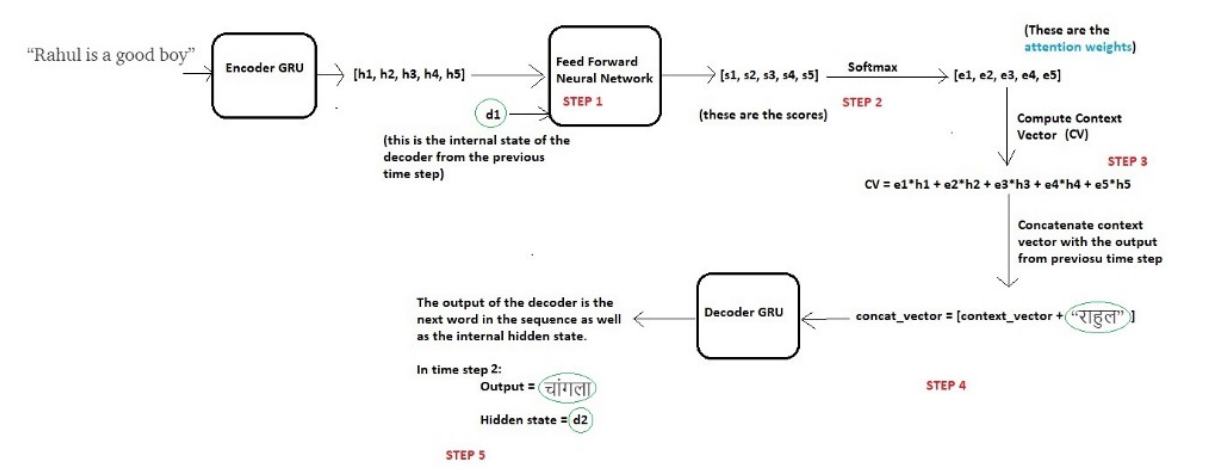
**Step 5— Decoder Output**

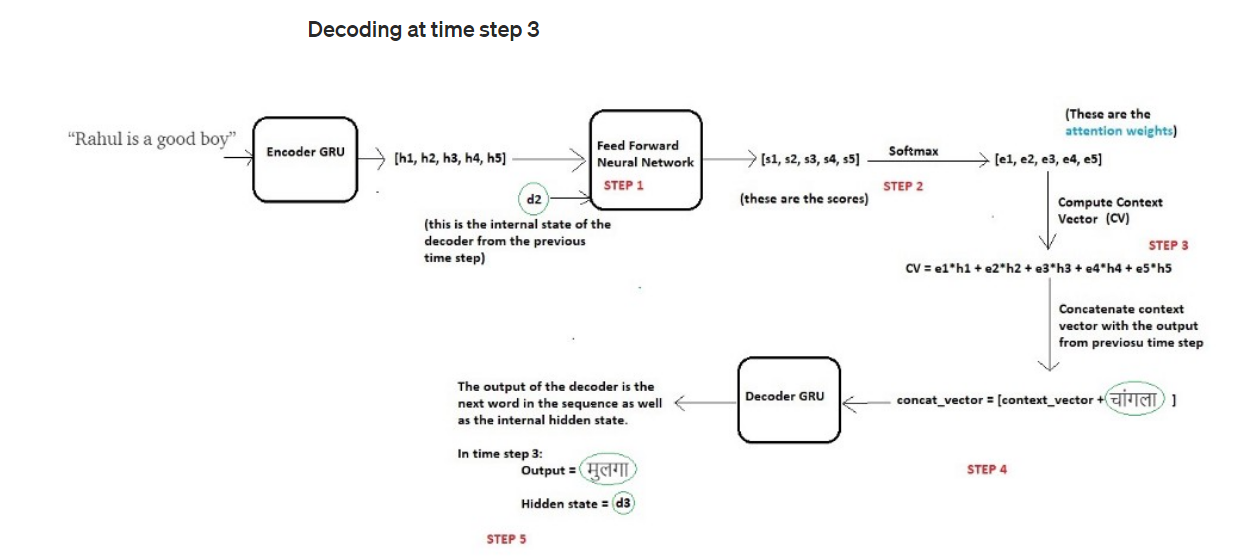
The decoder then generates the next word in the sequence (in this case, it is expected to generate “राहुल”) and along with the output, the decoder will also generate an internal hidden state, and lets call it as “d1”.

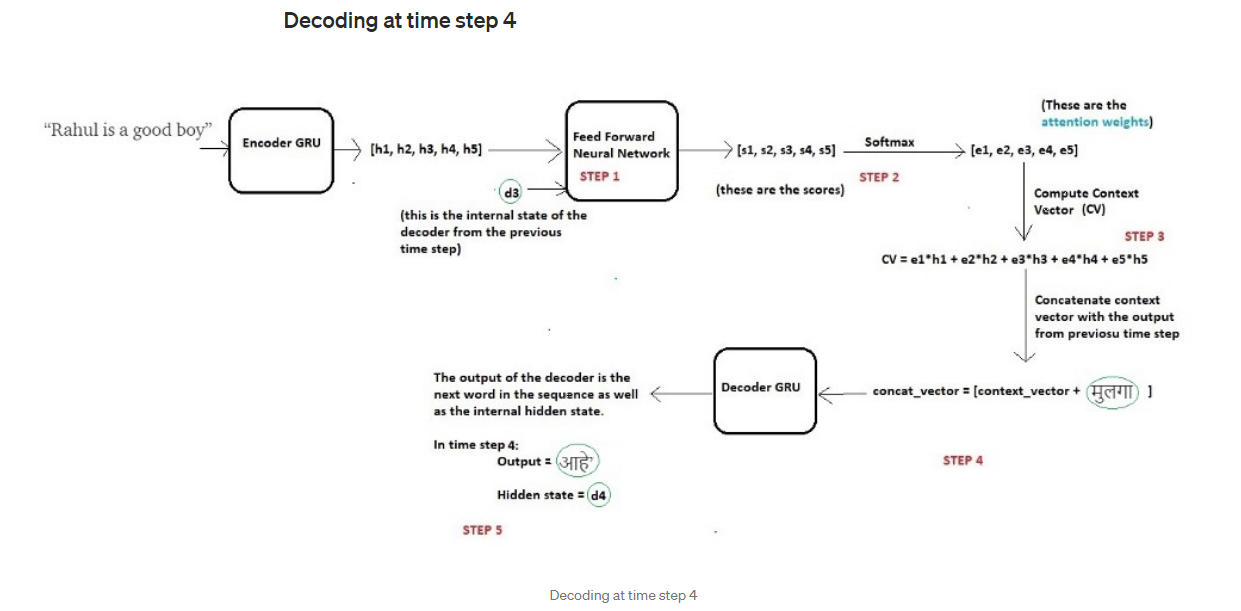
## Decoding at time step 2

Now in order to generate the next word “चांगला”, the decoder will repeat the same procedure which can be summarized in the below diagram:

The changes are highlighted in **green circles**

****

****

****

****

Once the decoder outputs the <END> token, we stop the generation process.

Note that unlike the fixed context vector used for all the decoder time steps in case of the traditional Seq2Seq models, here in case of Attention, we compute a separate context vector for each time step by computing the attention weights every time.

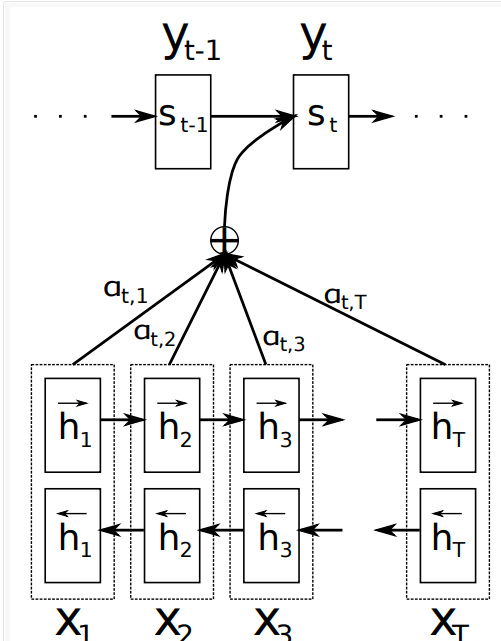
Thus using this mechanism our model is able to find interesting mappings between different parts of the input sequence and corresponding parts of the output sequence.

Note that during the training of the network, we use teacher forcing in order to input the actual word rather than the predicted word from the previous time step.

## Attention Model

* Attention is proposed as a solution to the limitation of the Encoder-Decoder model encoding the input sequence to one fixed length vector from which to decode each output time step. This issue is believed to be more of a problem when decoding long sequences.
* 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.
* Attention is proposed as a method to both align and translate.
* Alignment is the problem in machine translation that identifies which parts of the input sequence are relevant to each word in the output, whereas translation is the process of using the relevant information to select the appropriate output.
* … we introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

Instead of encoding the input sequence into a single fixed context vector, the attention model develops a context vector that is filtered specifically for each output time step.

****

As with the Encoder-Decoder paper, the technique is applied to a machine translation problem and uses GRU units rather than LSTM memory cells. In this case, a bidirectional input is used where the input sequences are provided both forward and backward, which are then concatenated before being passed on to the decoder.

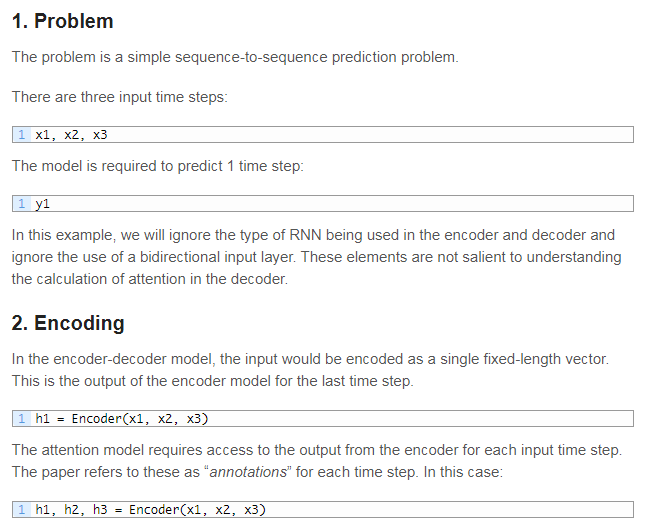
## Worked Example of Attention

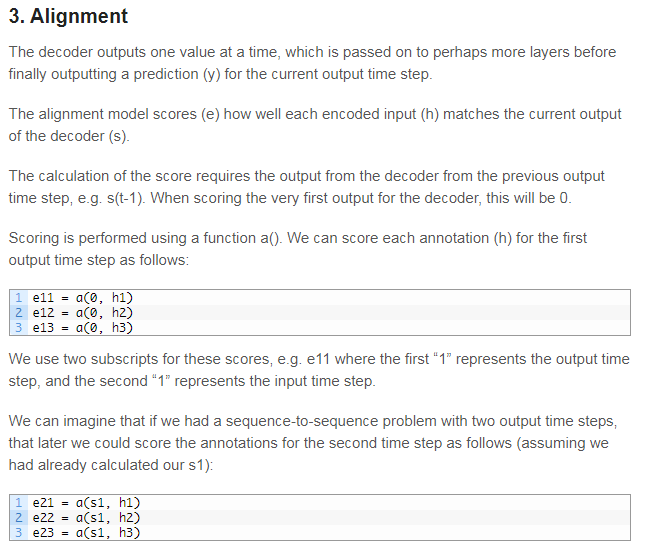
In this section, we will make attention concrete with a small worked example. Specifically, we will step through the calculations with un-vectorized terms.

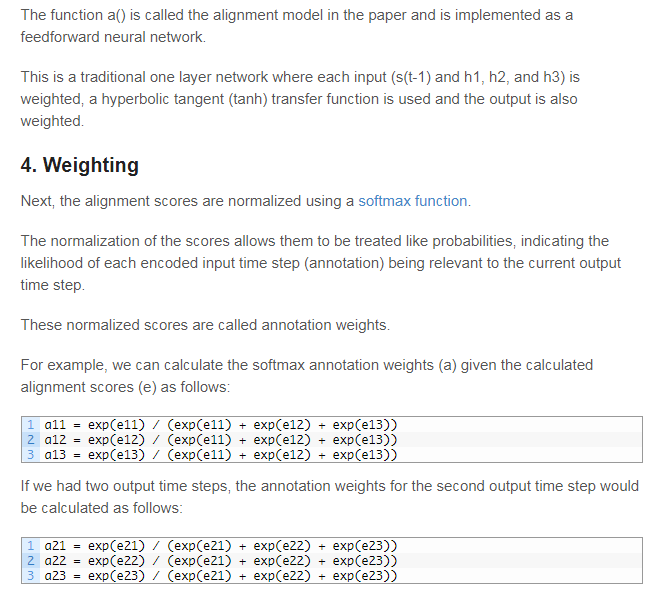
This will give you a sufficiently detailed understanding that you could add attention to your own encoder-decoder implementation.

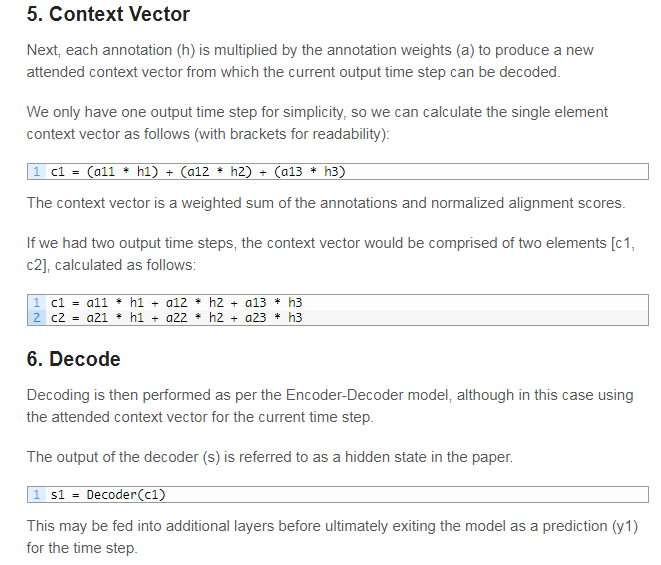
This worked example is divided into the following 6 sections:

1. Problem
2. Encoding
3. Alignment
4. Weighting
5. Context Vector
6. Decode

****

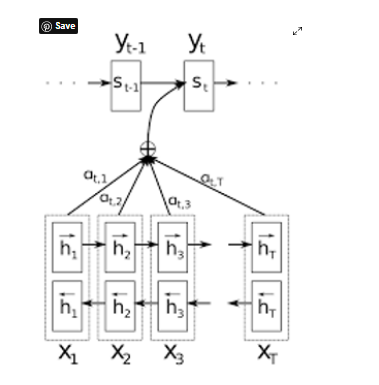
****

****

****

**Attention mechanism in Deep Learning, Explained**

*Attention is a powerful mechanism developed to enhance the performance of the Encoder-Decoder architecture on neural network-based machine translation tasks. Learn more about how this process works and how to implement the approach into your work.*

****

Attention is one of the most prominent ideas in the Deep Learning community. Even though this mechanism is now used in various problems like image captioning and others, it was originally designed in the context of Neural Machine Translation using Seq2Seq Models.

### Seq2Seq model

The seq2seq model is normally composed of an encoder-decoder architecture, where the encoder processes the input sequence and encodes/compresses the information into a context vector (or “thought vector”) of fixed length. This representation is anticipated to be a good summary of the complete input sequence. The decoder is then initialized with this context vector, using which it starts producing the transformed or translated output.

### The disadvantage of the Seq2Seq model

A critical disadvantage of this fixed-length context vector design is the inability of the system to retain longer sequences. Often it has forgotten the earlier elements of the input sequence once it has processed the complete sequence. The attention mechanism was created to resolve this problem of long dependencies.

**BLEU (Bilingual Evaluation Understudy)** is a score for comparing a candidate translation of text to one or more reference translations. The above graph shows that the encoder-decoder unit fails to memorize the whole long sentence. Hence, what’s reflected from the graph above is that the encoder-decoder unit works well for shorter sentences (high bleu score).

### The basic idea behind Attention

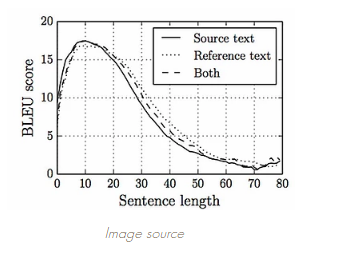
Attention was presented by Dzmitry Bahdanau, et al. in their 2014 paper “[Neural Machine Translation by Jointly Learning to Align and Translate](https://arxiv.org/abs/1409.0473),” which reads as a natural extension of their previous work on the Encoder-Decoder model. This very paper laid the foundation of the famous paper [Attention is All You Need by Vaswani et al](https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf)., on transformers that revolutionized the deep learning arena with the concept of parallel processing of words instead of processing them sequentially.

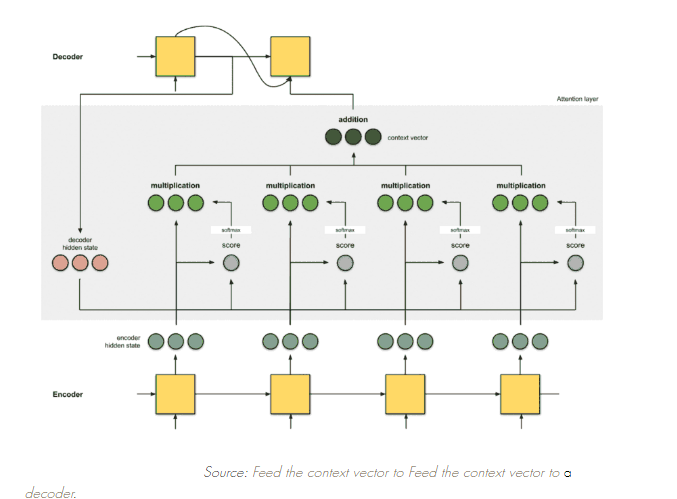
So coming back, the core idea is each time the model predicts an output word, it only uses parts of the input where the most relevant information is concentrated instead of the entire sequence. In simpler words, it only pays attention to some input words.

Attention is an interface connecting the encoder and decoder that provides the decoder with information from every encoder's hidden state. With this framework, the model is able to selectively focus on valuable parts of the input sequence and hence, learn the association between them. This helps the model to cope efficiently with long input sentences.

### Intuition

The below figure demonstrates an Encoder-Decoder architecture with an attention layer.

****

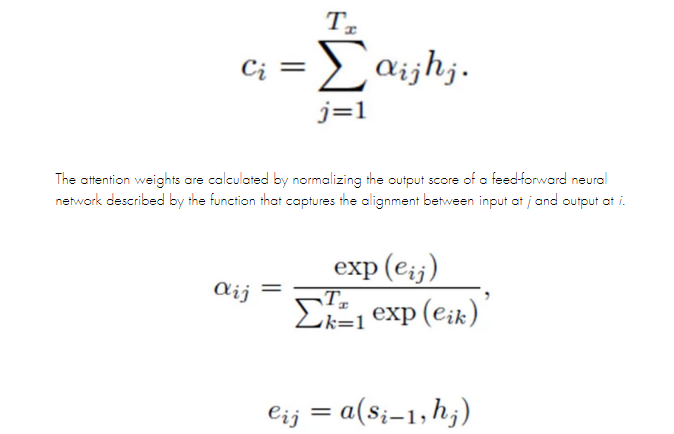
****

The idea is to keep the decoder as it is, and we just replace sequential RNN/LSTM with bidirectional RNN/LSTM in the encoder.

Here, we give attention to some words by considering a window size *Tx* (say four words *x1*, *x2*, *x3*, and *x4*). Using these four words, we’ll create a context vector *c1*, which is given as input to the decoder. Similarly, we’ll create a context vector *c2* using these four words. Also, we have α1, α2, and α3 as weights, and the sum of all weights within one window is equal to 1.

Similarly, we create context vectors from different sets of words with different α values.

The attention model computes a set of attention weights denoted by α(t,1), α(t,2),..,α(t,t) because not all the inputs would be used in generating the corresponding output. The context vector *ci* for the output word *yi* is generated using the weighted sum of the annotations:

****

### Implementation

Let's take an example where a translator reads the English(input language) sentence while writing down the keywords from the start till the end, after which it starts translating to Portuguese (the output language). While translating each English word, it makes use of the keywords it has understood.

Attention places different focus on different words by assigning each word with a score. Then, using the softmax scores, we aggregate the encoder hidden states using a weighted sum of the encoder hidden states to get the context vector.

The implementations of an attention layer can be broken down into 4 steps.

**Step 0: Prepare hidden states.**

First, prepare all the available encoder hidden states (green) and the first decoder hidden state (red). In our example, we have 4 encoder hidden states and the current decoder hidden state. (Note: the last consolidated encoder hidden state is fed as input to the first time step of the decoder. The output of this first time step of the decoder is called the first decoder hidden state.)

**Step 1: Obtain a score for every encoder hidden state.**

A score (scalar) is obtained by a score function (also known as alignment score function or alignment model). In this example, the score function is a dot product between the decoder and encoder hidden states.

**Step 2: Run all the scores through a softmax layer.**

We put the scores to a softmax layer so that the softmax scores (scalar) add up to 1. These softmax scores represent the attention distribution.

**Step 3: Multiply each encoder's hidden state by its softmax score.**

By multiplying each encoder's hidden state with its softmax score (scalar), we obtain the alignment vector or the annotation vector. This is exactly the mechanism where alignment takes place.

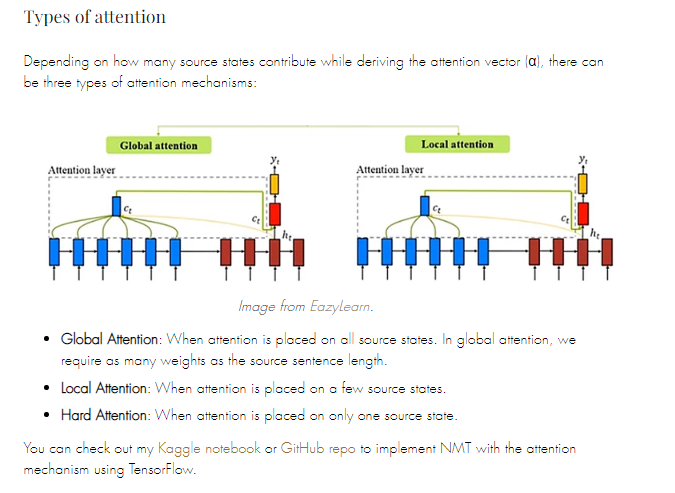
**Step 4: Sum the alignment vectors.**

The alignment vectors are summed up to produce the context vector. A context vector is an aggregated information of the alignment vectors from the previous step.

**Step 5: Feed the context vector into the decoder.**

### Types of attention

Depending on how many source states contribute while deriving the attention vector (α), there can be three types of attention mechanisms:

****

**https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/**

[**https://blog.floydhub.com/attention-mechanism/**](https://blog.floydhub.com/attention-mechanism/)

**https://towardsdatascience.com/intuitive-understanding-of-attention-mechanism-in-deep-learning-6c9482aecf4f**

**https://machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/**