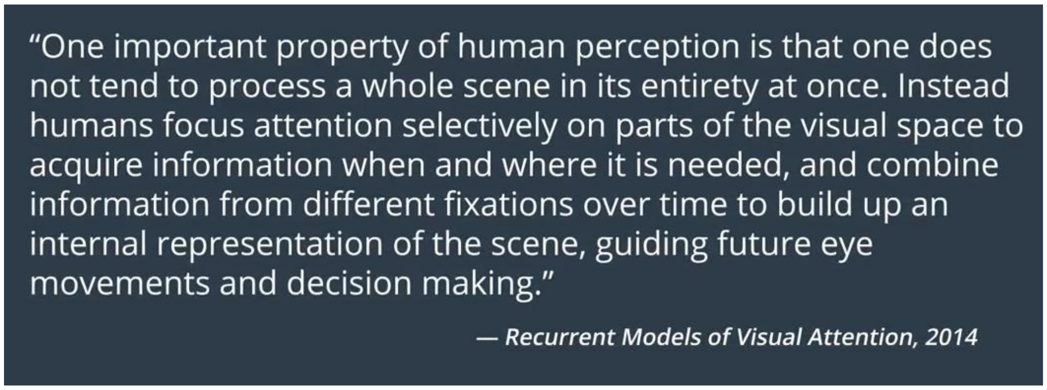
Attention started out in the field of Computer vision as an attempt to mimic human perception.



"One important property of human perception is that one does not tend to process a whole scene in its entirety at once. Instead humans focus attention selectively on parts of the visual space to acquire information when and where it is needed, and combine information from different fixations over time to build up an internal representation of the scene, guiding future eye movements and decision making."

— Recurrent Models of Visual Attention, 2014

This is quote of paper on visual attention 2014.

What that mean is that when we look at a scene in our daily lives, our brain does not just process a visual snapshot all at once.

Instead we selectivity focus on different parts of the image, and we sequentially collect and process that visual information over time.

Say, For example you are shopping in a shopping mall.

Now, the footage that we are going to see here is from an eye-tracking device.

If you haven`t seen those before, it's a **device** that records both what in front of you and it also records your eyes movement.

Then, we can overlay these two recordings so we can have and idea of where looking at each time in the video.

So, what we are seeing here is footage from a person the eye-tracking device,

Where the orange circle is highlighting where the person is looking at each moment.

So, we can see attention in general visual perception, but you can also it in reading and trying to process text one word at a time.

This type of device is used, for example in user experience testing.

If you wanted to build a user interface for an app or website and you wanted to track if the most important thing are actually grapping the attention of users, and since that`s how humans tend to understand a visual picture sequentially over time,

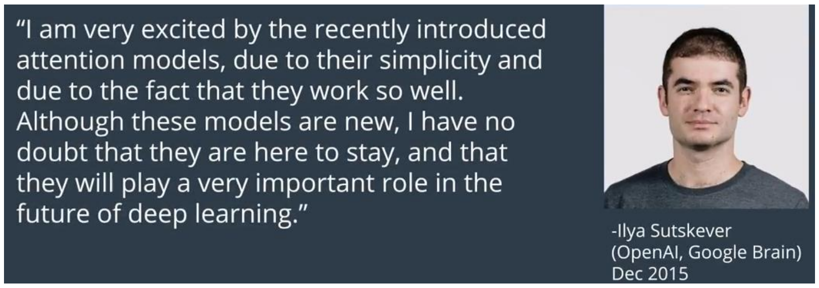
The idea from computer vision researchers was to try to adopt a method that does for computer vision models.

In machine learning attention methods give us a mechanism for adding selective focus into a machine learning model.

Typically, one that does its processing sequentially.

Attention is a concept that powers up some of the best performing models spanning both

1- **NLP = natural language processing**
2- **Computer vision**



"I am very excited by the recently introduced attention models, due to their simplicity and due to the fact that they work so well. Although these models are new, I have no doubt that they are here to stay, and that they will play a very important role in the future of deep learning."

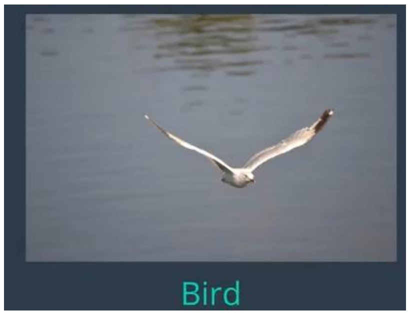
-Ilya Sutskever
(OpenAI, Google Brain)
Dec 2015

These models include:

1- **Neural machine translation**
2- **image captioning**
3- **Speech recognition**
4- **Text summarization**

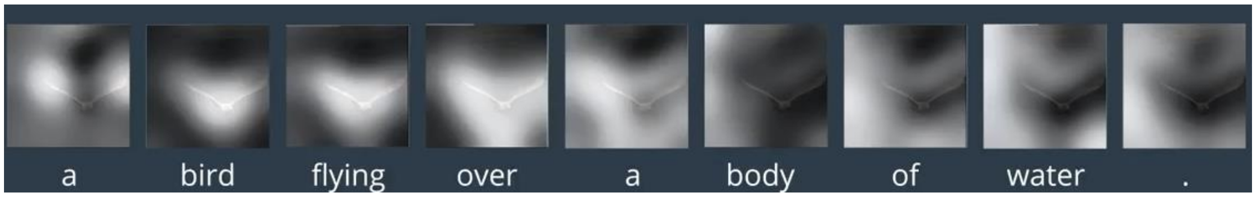
Take image classification in capturing as an example.

Before use attention, convolutional neural network were also to classify images by looking at the whole image and outputting a class label.



But not all of this image is necessary to produce that classifications, only some of these pixel are needed to identify a bird, and attention came out of the desire to attend to these most important pixels.

Now, not only that, but attention also improved our ability to describe images with full sentence by focusing on different part of the image as we generate our output sentence.

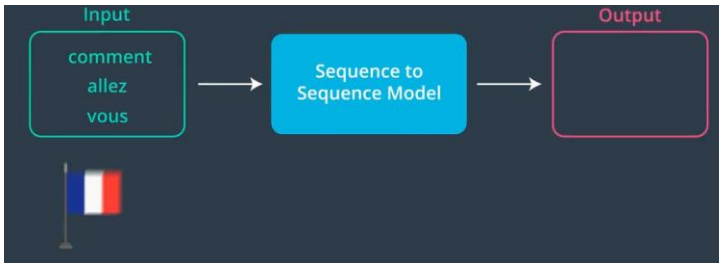


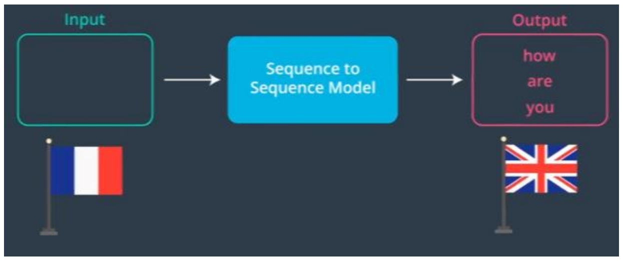
Attention achieved its rise to frame however, from how useful it became in tasks like neural machine translation.

As sequence to sequence models started to exhibits impressive results, they were held back by certain limitations that made it difficult for them to process long sentence for example,

Classic sequence to sequence models, without attention, have to look at the original sentence that you want to translate one time and then use that entire input to produce every single small outputted work.

Attention, however allows the model to look at this small relevant parts of the input as you generate the output over time.



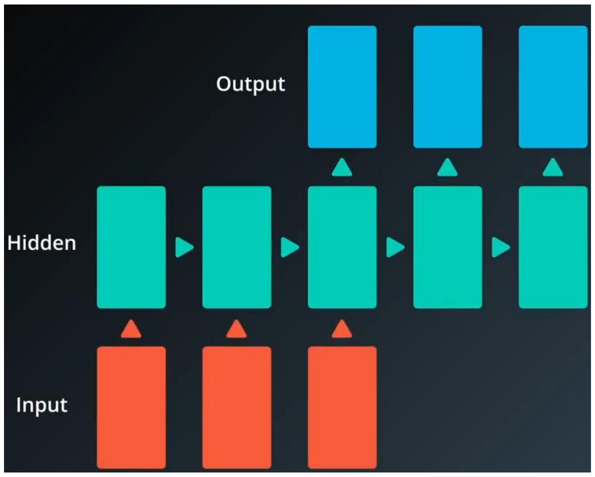


When attention was incorporated in sequence to sequence models, they become the state of the art in neural machine translation.

This is what like Google to adopt neural machine translation with attention as the translation engine for google translate in the end 2016.

What is the application ?

That`s because the term sequence-to-sequence RNN is a little bit abstract and doesn`t relay how many amazing things we can do with this type of model.



So, let`s think of it like this.

We have a model that can learn to generate any sequence of vector. And these can be letters.

They can be words or images or anything , really.

If you can represent it as a vector,

It can be used in a sequence-to-sequence model.

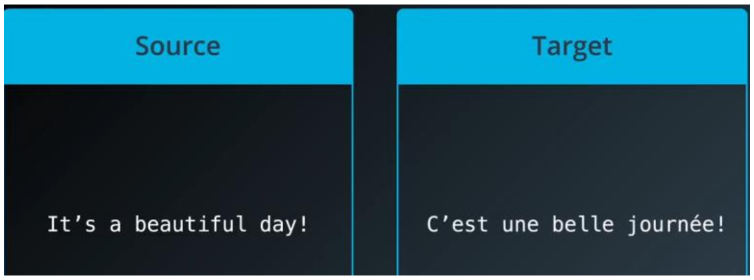
So this model can learn to generate any sequence of vectors after we feed it a sequence of input vectors.

What can we do with that?

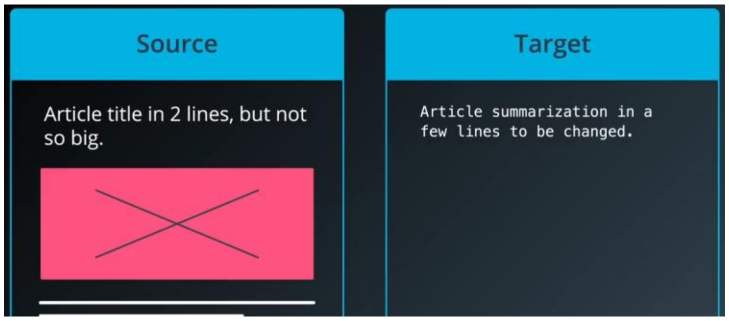
Say you train it on a dataset where the source is a English phrase and the target is a French.

And you have a lot of these examples.

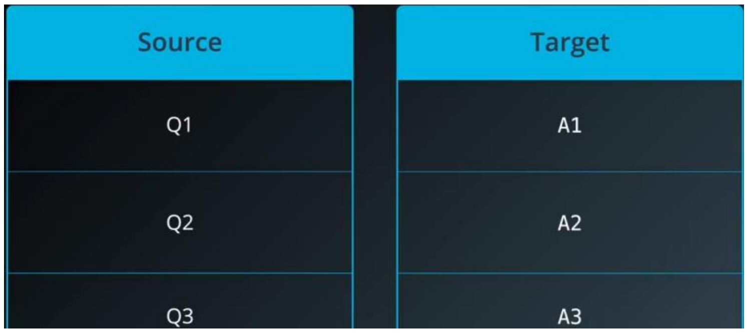
If you do that and you train it successfully, then your model is now in English-to-French translator.



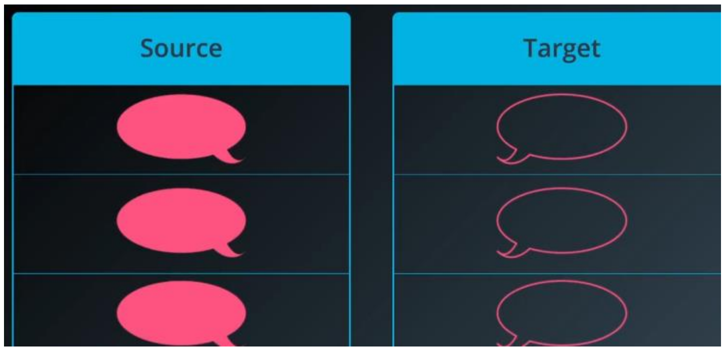
Train it on a dataset of news articles and their summaries and you have a summarization bot.


Source
Target
Article title in 2 lines, but not so big.
Article summarization in a few lines to be changed.

Train it on a dataset of questions and their answers and you have a question-answering model.



Train it on a lot of dialogue data and you have a chatbot.



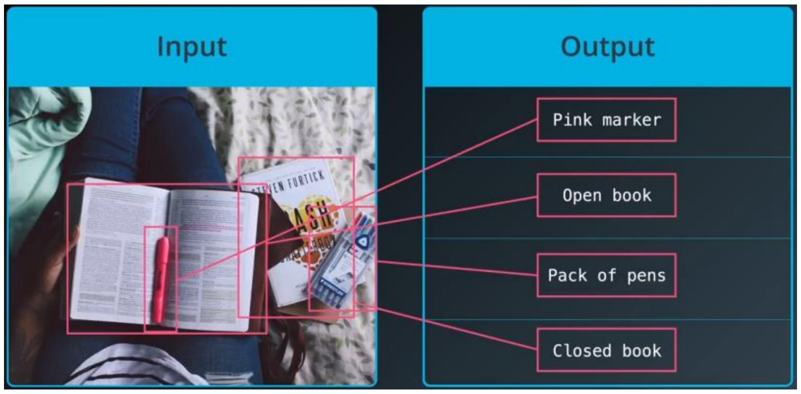
But the input don`t only have to be words, remember.

The RNNs are used along convolutional nets and image , captioning test, for example.

So it can be also be—the input sequence can also be audio.

As we saw, there are many possibilities for what you can do after you master sequence to sequence.

The challenge will be to find the right dataset to feed you model and guide it through the learning process.



Let`s look more closely how sequence to sequence to models work.

We will start with high level look and then go deeper and deeper.

Here are our two recurrent nets.

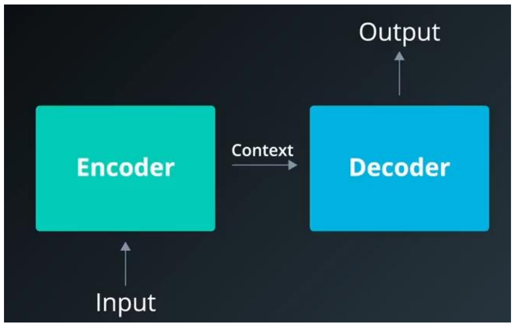
1- The one on the left is called the encode. (it read the input sequence then hands over what it has understood to the RNN)
2- And on the right which we call the decoder.

And the decoder generates the output sequence.

The quote unquote “understanding” that gets handed over a fixed size tenser that is called is state , or sometimes called the context.

So no matter how short or long the inputs and outputs are, the context remains the same size that was declared when we build the model in the beginning.

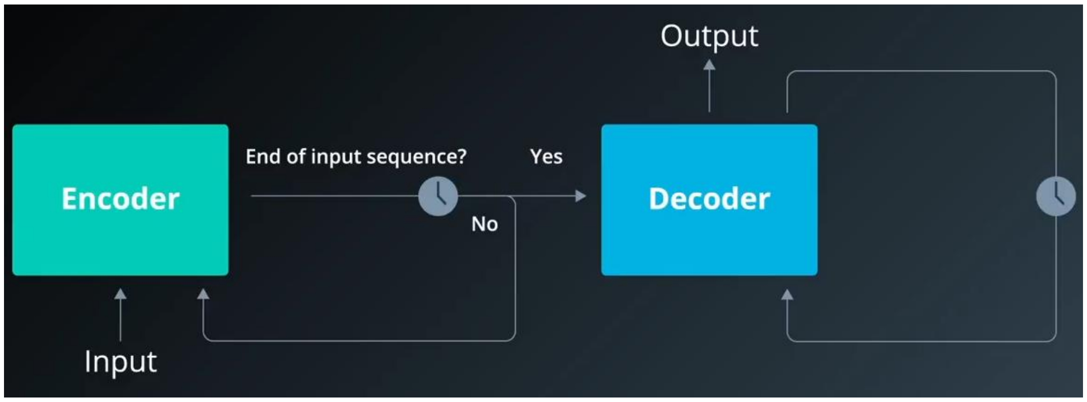
So at this high level inference process. It done by handing the input to encoder.



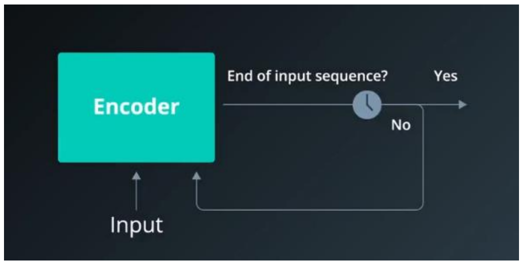
The encoder summarizes what it then understood into a context variable or state.

And it hands it over to the decoder, which then proceeds to generate the output sequence.

Now if we go level deeper we begin to see that since the encoder and decoder are both RNNs, they have loops naturally and that`s what allows to process these sequence of inputs and outputs.



Let`s take an example.

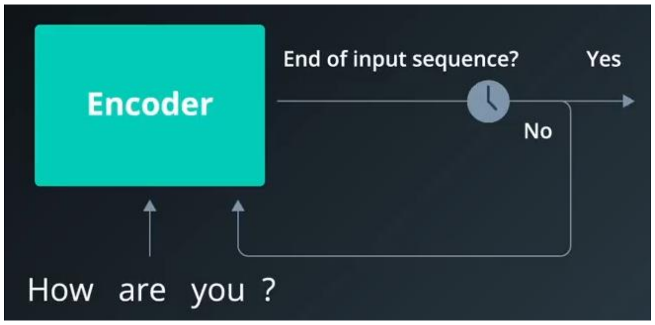


Say our model is jackpot, and we want to ask "how are you " question mark.

So first, we have to tokenize that input and break it down into four tokens and since It has four elements, it will take the RNN four timesteps to read in this entire sequence.

1- Each time it read it would read an input, and then do a transformation on its hidden state, then send that hidden state our to the next time step.

The clock symbol indicates that we are moving from one timesteps to the next.

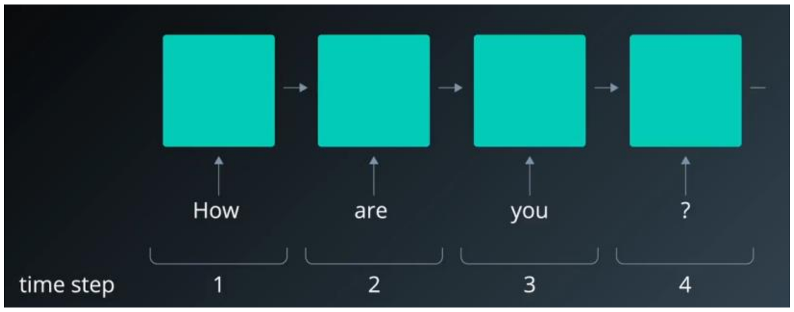


One useful way to represent the flow of data through an RNN is by unrolling quote unquote the RNN.

That is graphing it to show each timestep as a separate cell.

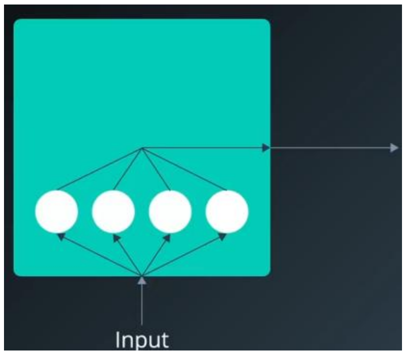
Even though, in practice it`s just the same cell. Only it`s processing a new input and it`s taking over the hidden state from the previous timestep.

"Encoder"



So, what a hidden state, you may ask. In the simplest scenario you can think of it as a number of hidden units inside the cell.

In practice it`s more likely to be the hidden state a long, short-term memory cell an LSTM.



So, the size of the network is usually another hyperparameter that we can be set to build model.

1- **The bigger the hidden states**
2- **The bigger the size**
3- **The more capacity of the model to learn**

And look at the pattern and try to understand them.

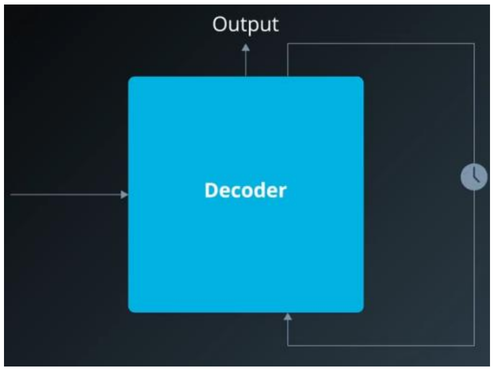
But the more resource intensive the model will be to train or deploy in terms of processing and memory demand.

So, it`s that trade-off that you usually face with models, in general.

A similar process happens on the decoder side, as well.

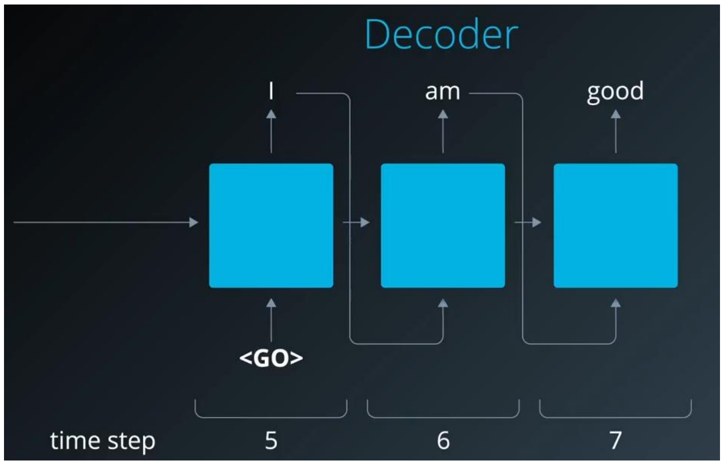
So, we begin by feeding it this data generated by the encoder.

And it generates the output elements by elements.



If we unroll the decoder just like we did earlier with the encoder, so we can see that we are actually feeding it back every element that it outputs.

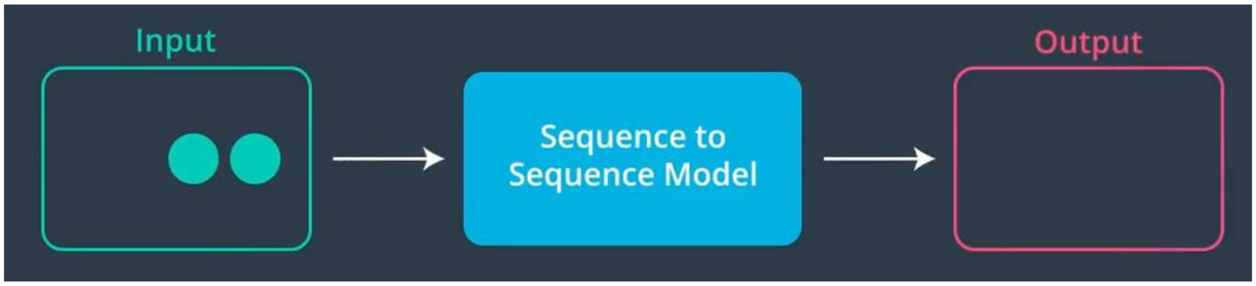
This allow it to be more coherent as each **timestep** sort of remembers what the previous timestep has committed to.



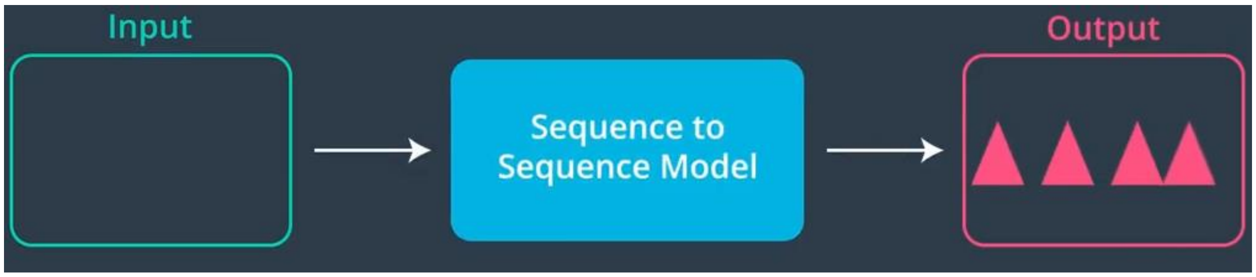
Sequence to sequence Recap

We will briefly recap how sequence to sequence model work.

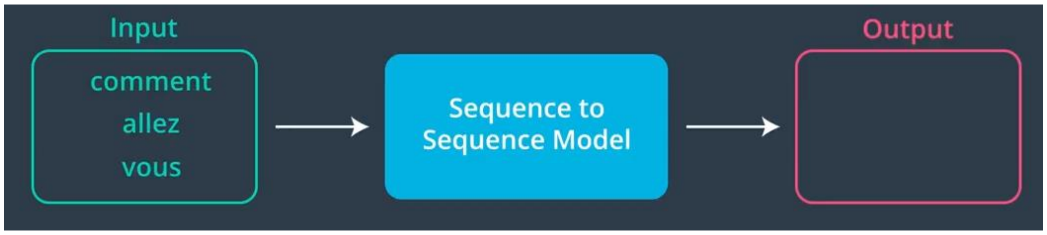
A sequence to sequence model takes in an input that is sequence of items, and then produce another sequence of items as an output.



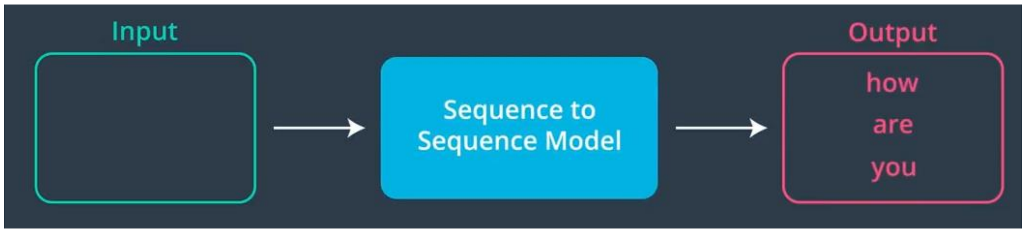
And then it produces another sequence of items as an output.



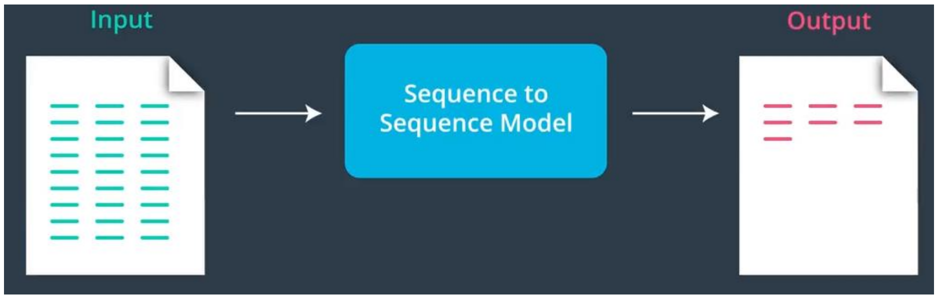
In a machine translation application the input sequence is a series of words in one language, and the output is the translation in another language.



Ant translates to another language.

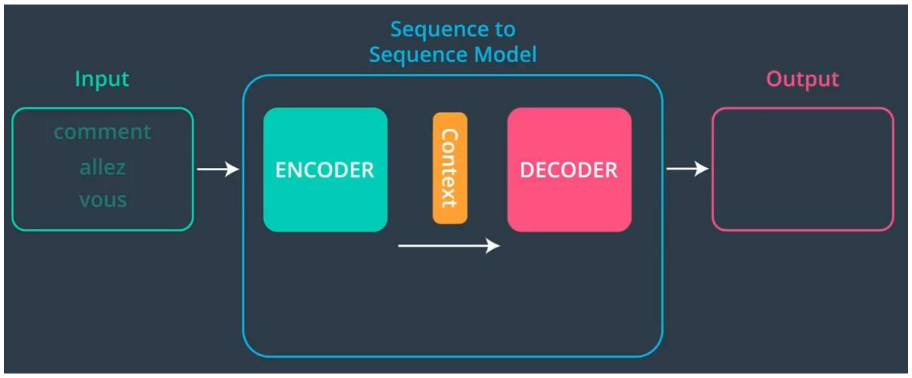


In text summarization, the input is a long sequence of words, and the output is a short one.

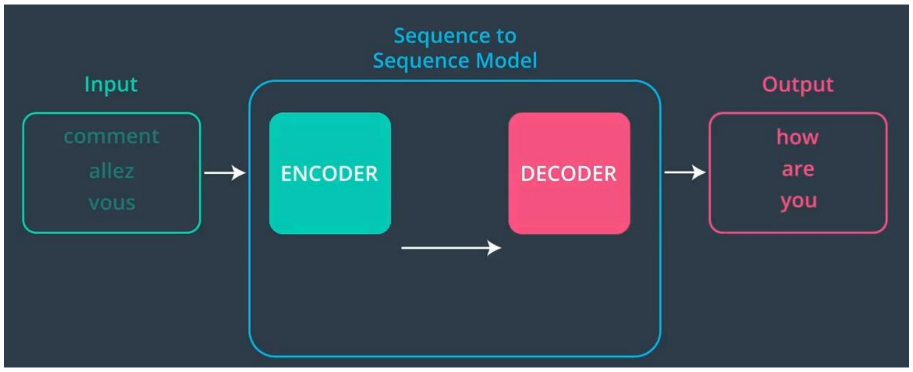


Sequence to sequence model

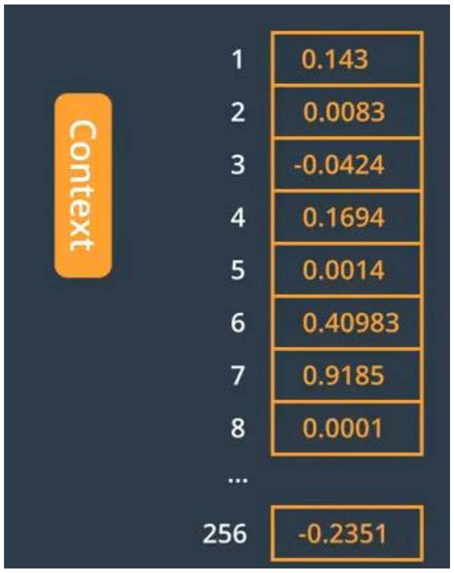
A sequence to sequence model usually consists of an encoder and a decoder, and it works by the encoder first processing all of the inputs turning the inputs into a single representation, typically a single vector. This is called the context vector, and it contain whatever information the encoder was able to capture from the input sequence.



This vector is then sent to the decoder which uses it to formulate an output sequence.



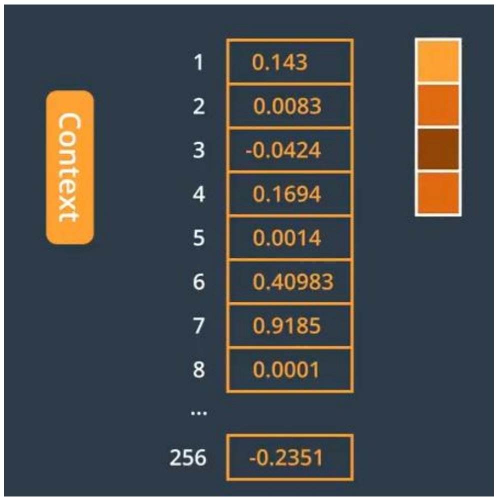
In machine translation scenario, the encoder and decoder are both recurrent neural network, typically LSTM cells in practice, and in this scenario, the context vector is a vector of numbers encoding the information that the encoder captured from the input sequence.



In real-world scenario, this vector can have a length of 256 or 512 or more.

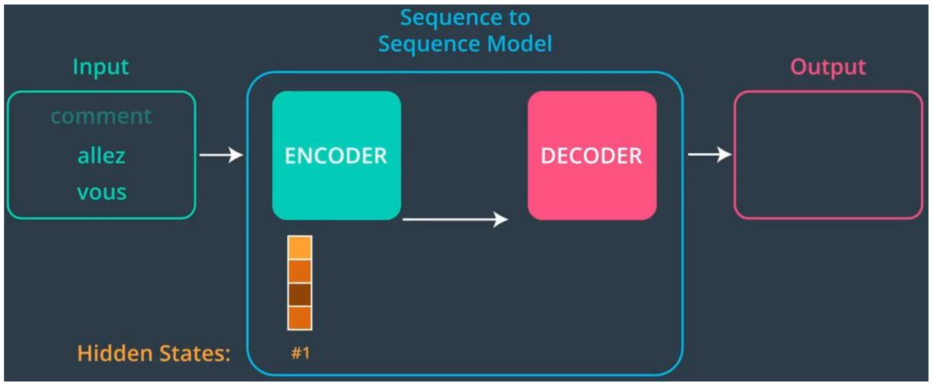
As visual representation, we will start showing the hidden states as this vector of length four.

Just think of the brightness of the cells corresponding to how high or low the value of the cell is.

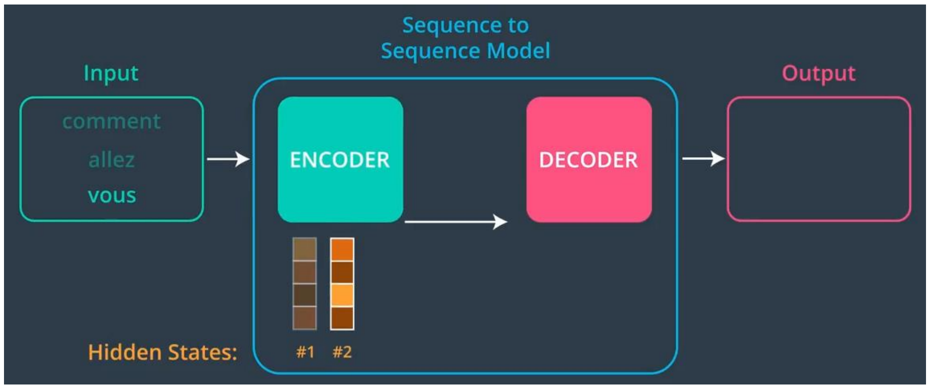


Let`s look our basic example again, but this time we will look at the hidden states of the encoder as they develop.

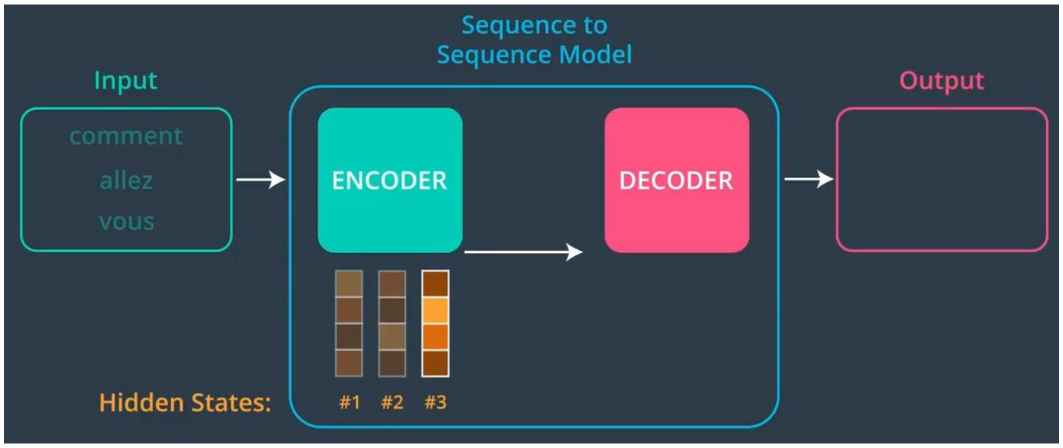
1- **The first step, we process the first word and generate a first hidden state.**



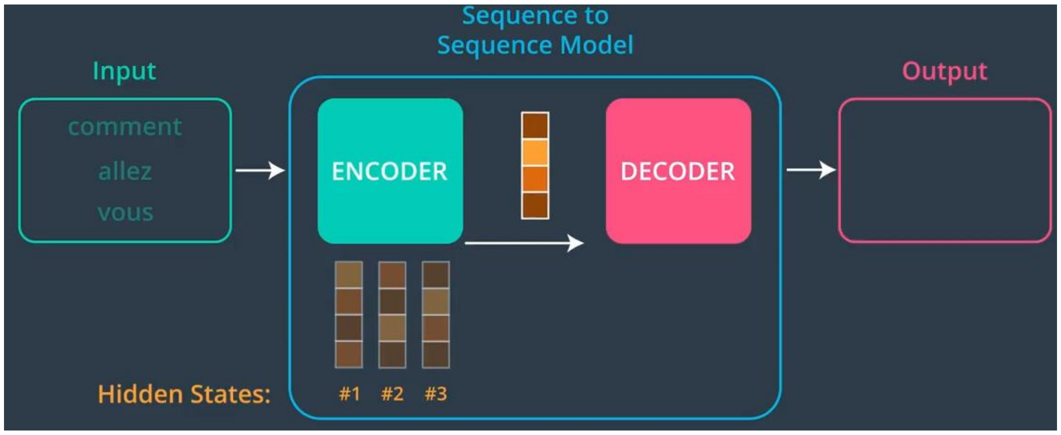
2- **Second step, we take the second word in the first hidden state as inputs to the RNN, and produce a second hidden state.**



3- In third step, we process last word and generate the last hidden last hidden state.



This is hidden state that would be the context vector will send to the decoder.



Now, this here is limitation of sequence to sequence models.

The encoding is confined to sending a single vector no matter how long or short the input sequence is.

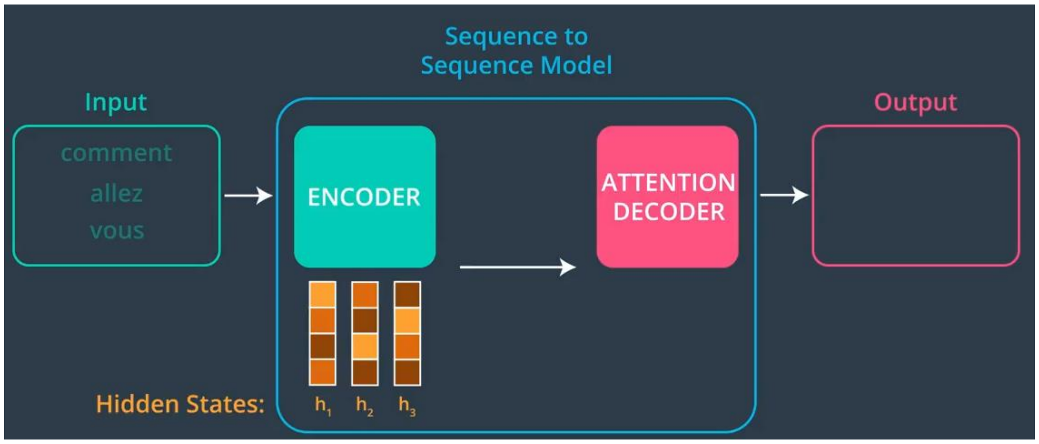
- Choosing a reasonable size or for this vector makes the model have problems with long input sequence.

Now, one can say, let`s just use a very large number of hidden units in the encoder, So the context is very large. But then model overfit with short sequence, and you make a performance hit as you increase the number of parameters.

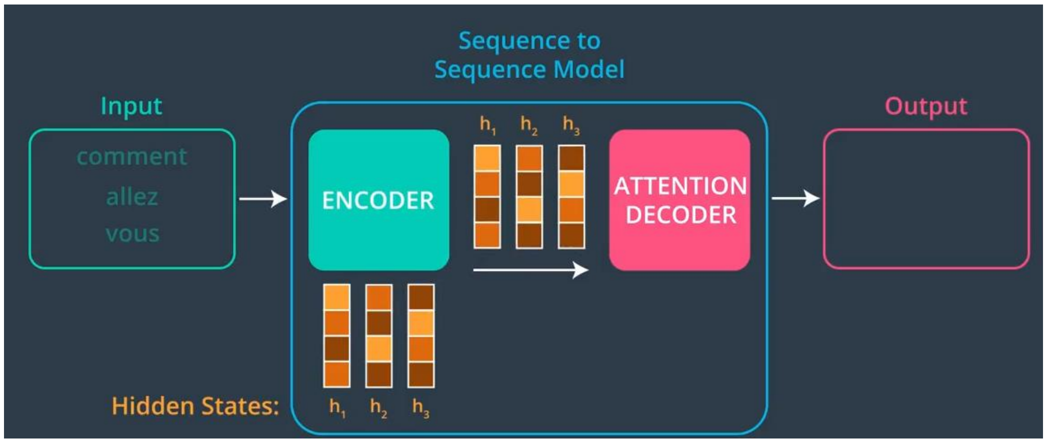
This is the problem that attention solves.

Attention overview encoding

1- **First, the encoder processes the input sequence just like the model without attention one word at a time, producing hidden state and using that hidden state and the next step.**



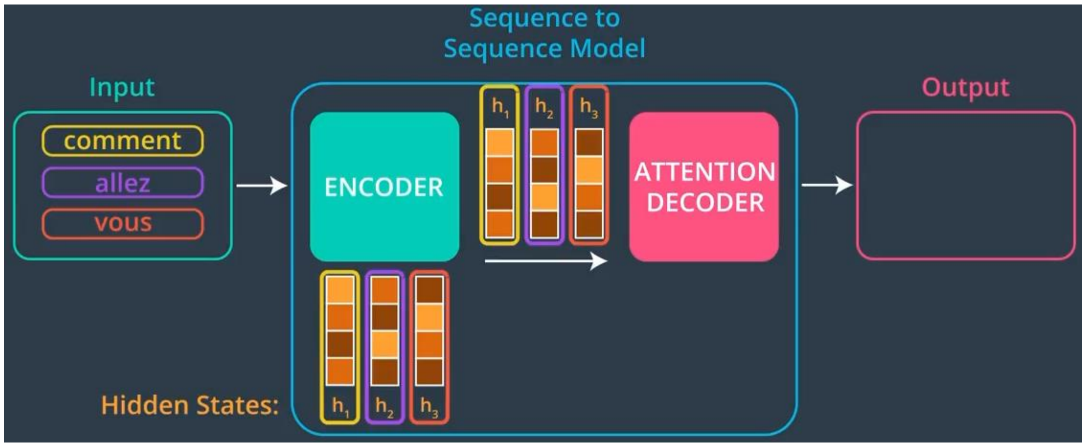
Next, the model passes a context vector to the decoder but unlike the context vector in the model without attention, this one in not just final hidden state it`s all of the hidden states.



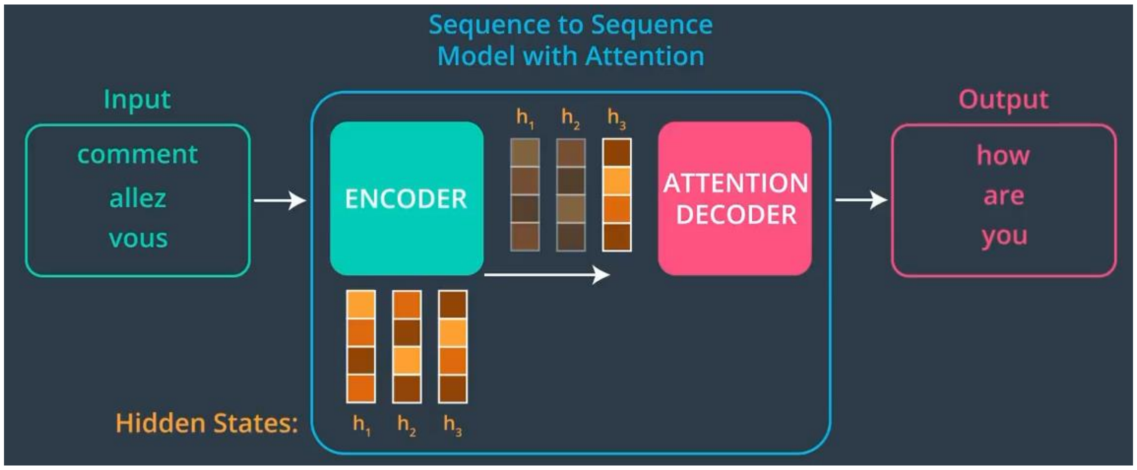
This is giving us the benefits of having the flexibility in the context size. So longer sequences can have longer context vectors that better capture the information form the input sequence.

One additional point that`s important for the intuition of attention, is that each hidden state is sort of associated the most with part of the input sequence that preceded how that word was generated.

1- **The first hidden state was outputted after processing the first word, so it captures the essence of the first word the most. So we focus in this vector on the word the most, we will focusing on the word the most, the same with the second hidden state with the second word with the third word, even though that last and third vector incorporates a little bit of everything that preceded it as well.**



At every time step , An attention decoder pays attention to the appropriate part of the input sequence using context factor.



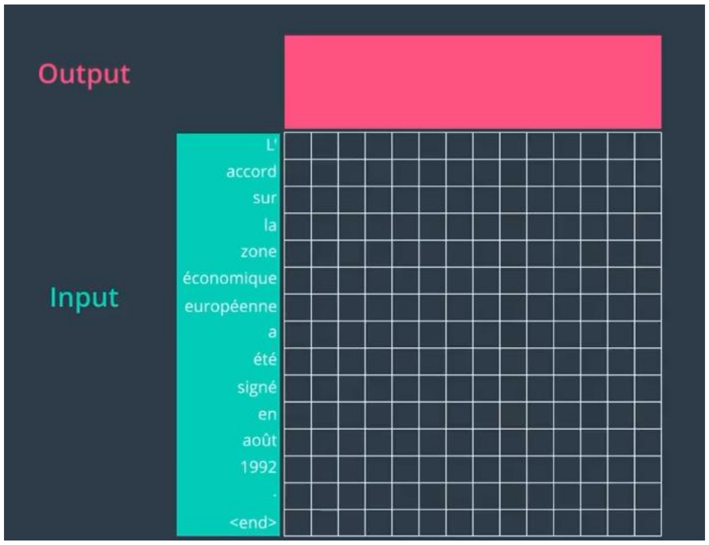
How to attention decoder know which of the parts of the input sequence to focus on at each step? At

1- **This process is learned during the training phase, and it`s not just stupidly going sequentially from the first and the second to the third.**

It can learn some sophisticated behavior.

Let`s look at this example of translating a French sentence to an English one.

So, let`s say we have this input senescence in French.

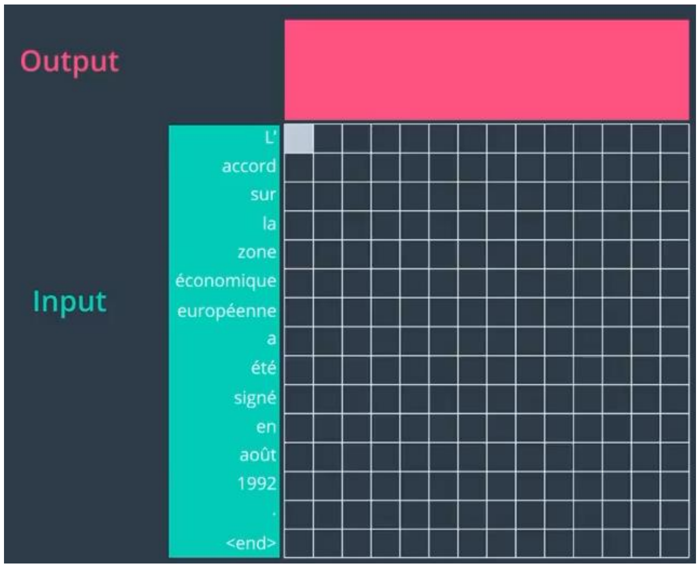


Let`s say we pass this to our encoder and now we are ready to look at each step in the encoding phase.

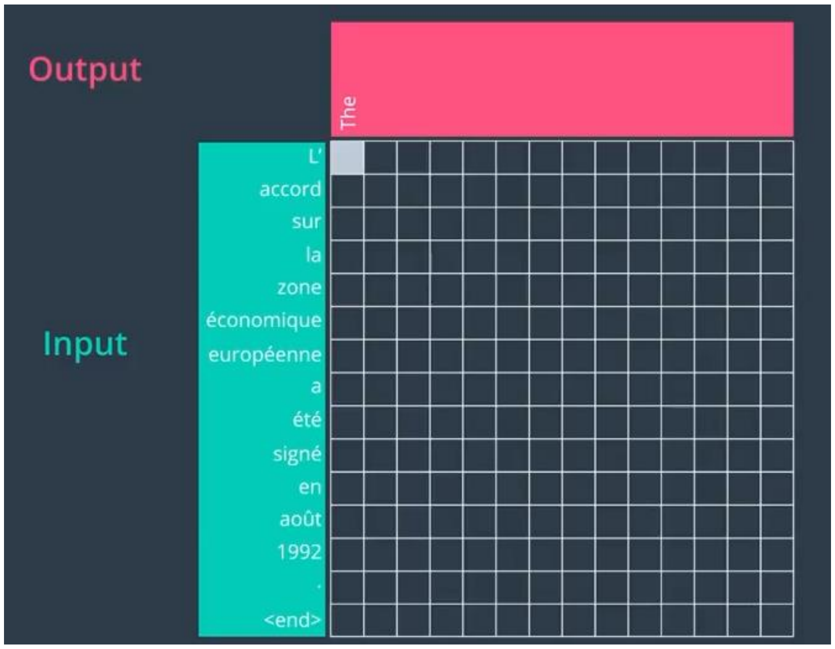
In the first step, the attention decoder would pay attention to the first part of the sentence.

This is trained model, right.

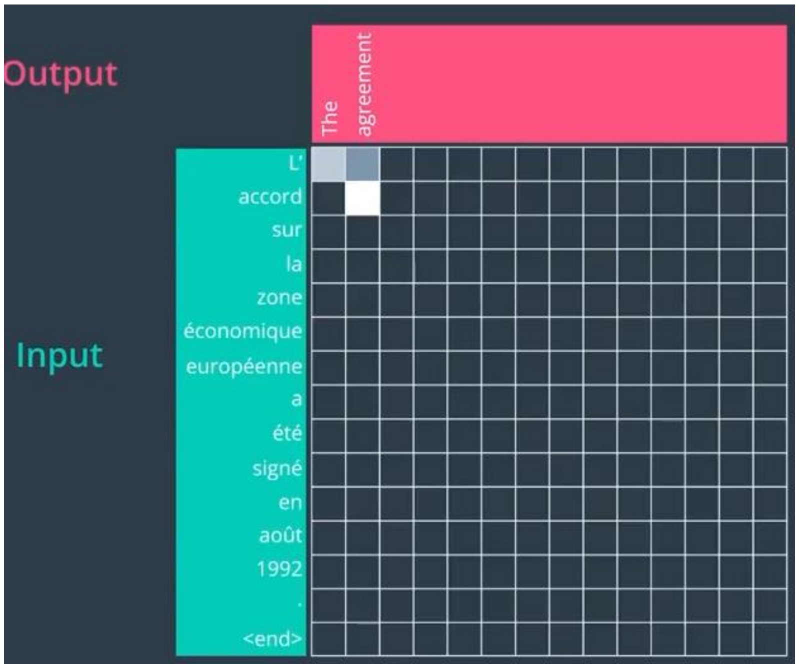
So, the more light the square is the more attention that he gave to that word in particular.



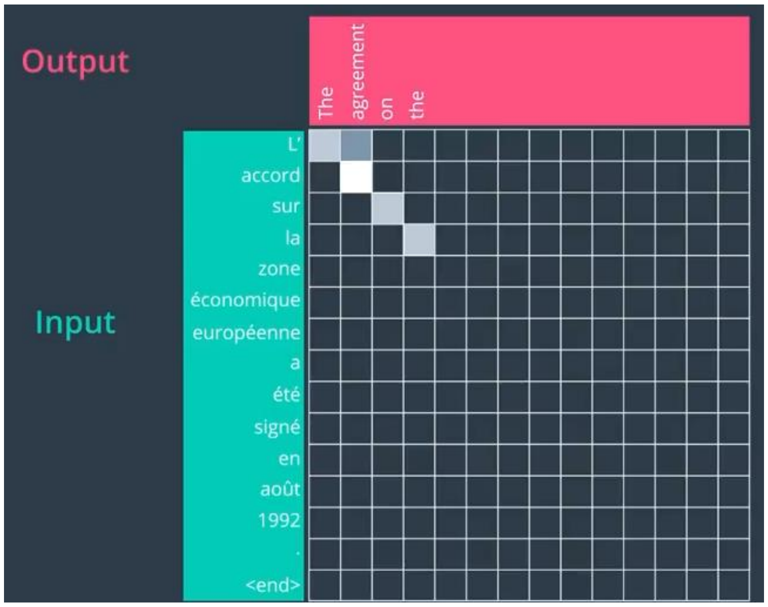
So, it pays attention to the first word and it outputs a first English word.



In the second step, it pays attention to the second word in the input sequence and translates that word as well.



It goes on sequentially for about four steps and it produces reasonable English translation so far.



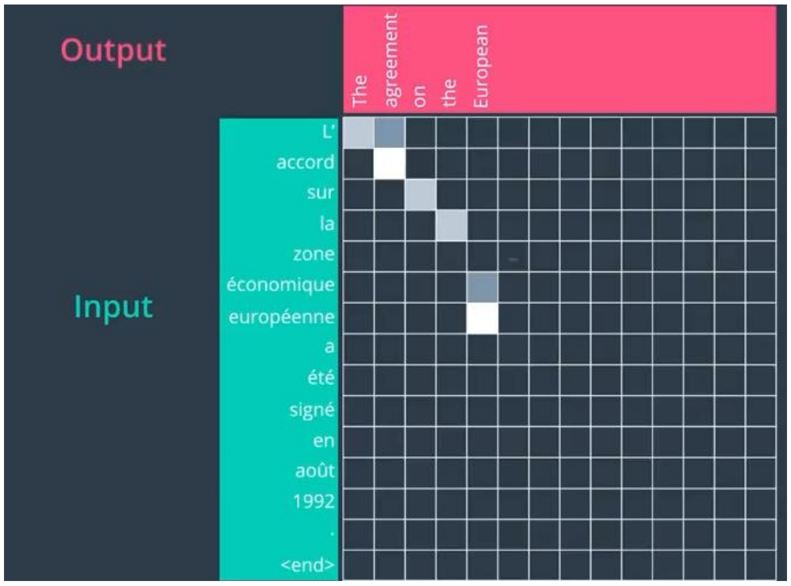
Then something different happens here in the fifth step.

So, when we are generating the fifth word of the output, the attention actually jumped two words to translate European.

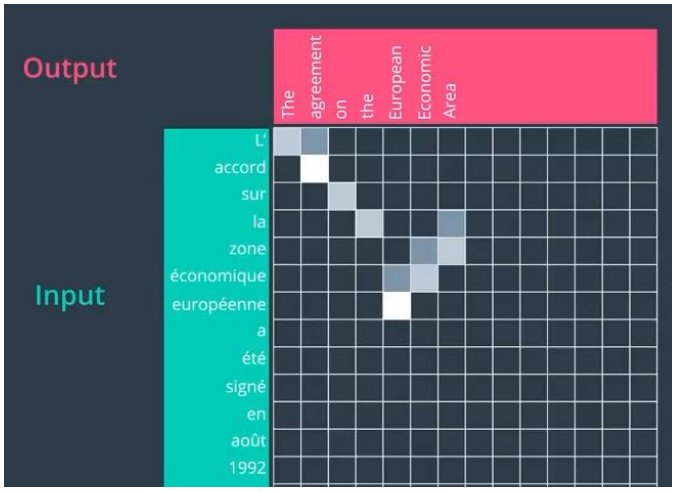
So, we have zone, economique, europeenne

So on the English side it`s not going to be in the same order.

So, europeene is translate as European.



The next step it focusses on the word before that, economique, economice and it focuses on zone and it outputs area.



This is a case where the order of these words in the French language does not follow how it would be ordered in the English language and the model was able to learn that just from a training data set.

The rest of the sentence goes on pretty much sequentially,

So, this is really cool example of how attention is able to make these model focus on the right parts at the right moment based on what dataset we have.

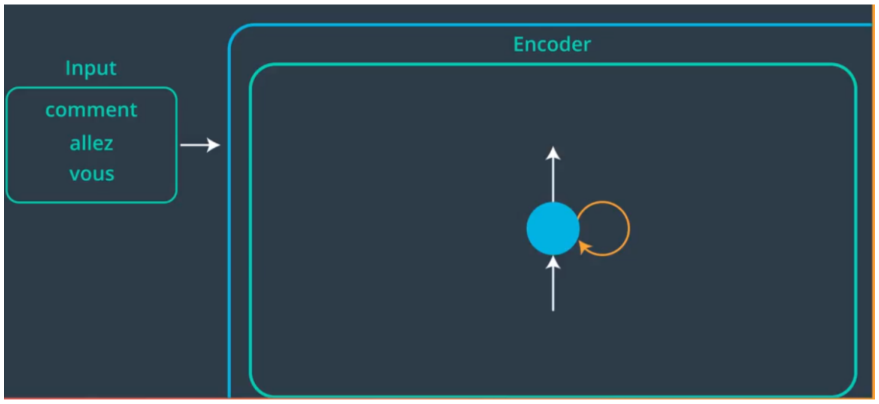
<u>Attention Encoder:</u>

<u>How to attention work in a sequence to sequence model, Let`s look into it in more details.</u>

We will use machine translation as the example as that`s the application the main papers on attention tackled. But whatever we do here, translate into other application as well.

- ***It`s important not that there is a small variety of attention algorithms.***

Let`s start from the encoder.



In this example, the encoder is an RNN. When creating an RNN, we have to declare the number of hidden units in the RNN cells.

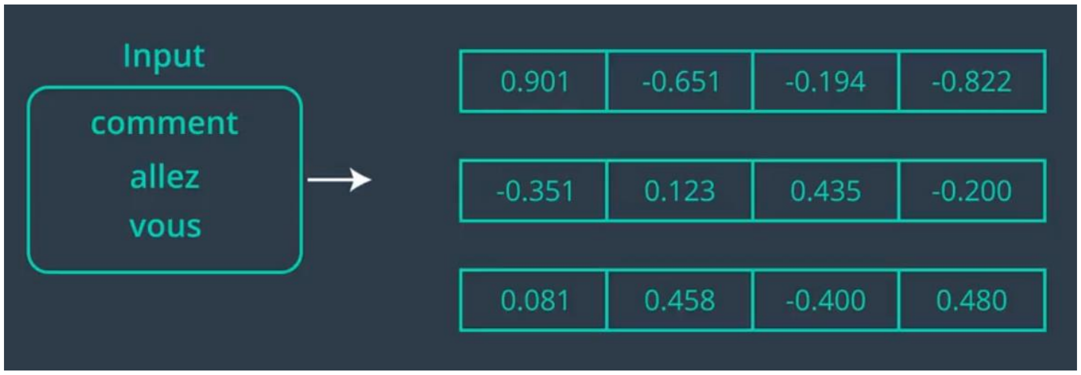
This applies whether we have a

1- ***Vanilla RNN cell***
2- ***LSTM cell***
3- ***GRU cell***

Before we start feeding our input sequence words to the Encoder, they have to pass through an embedding process which translates each word into a vector.

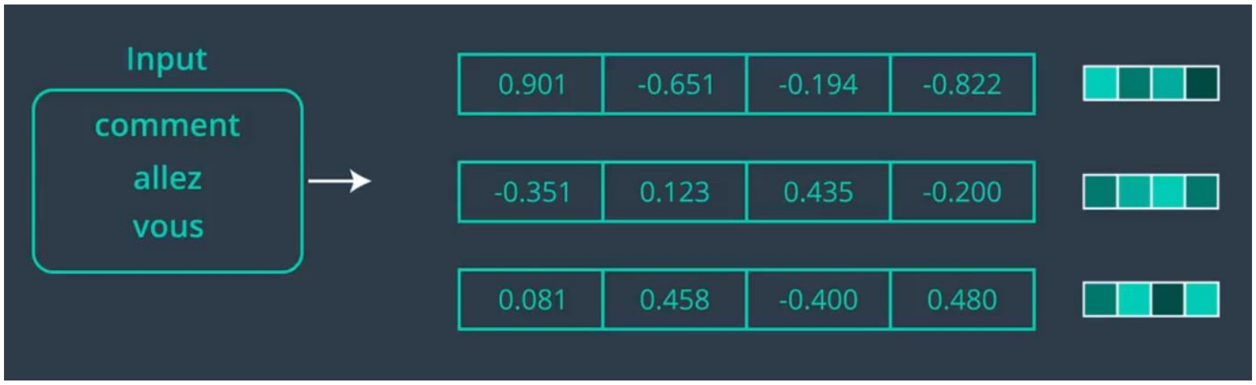
Here we can see the vector representing each of those words.

Now, this is a toy embedding of size four just for the purpose of easier visualization.



In real-world applications, a size like 200 or 300 is more appropriate.

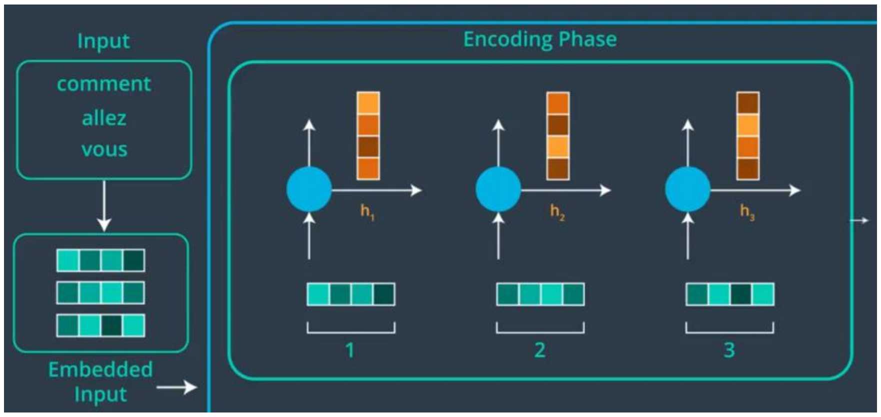
We will contain to use these Color-coded boxes to represent the vectors, just se we don`t have a lot of numbers plastered all over the screen.



*Now we have our words and their **embedding**, we are ready to feed that into our encoder.*

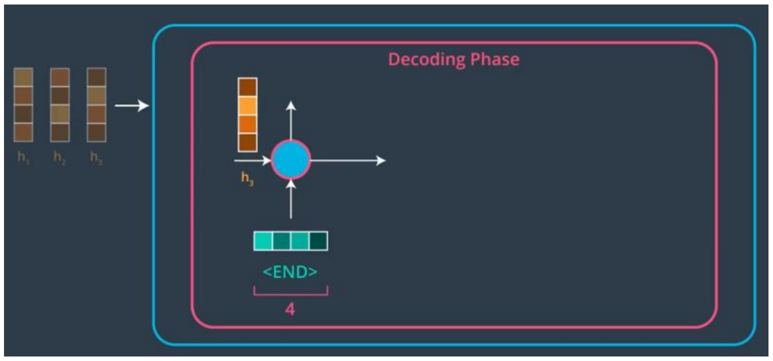
1- *Feeding the first word into the first time step of the RNN produces the first hidden state. (**This is what`s called an unrolled view of the RNN, where we can see the RNN at each time step.**)*
2- *We will hold into this state and the RNN would continue to process the next time step. (So, it would take word and pass it to RNN at the second time step, and then it would do that with the*

third word as well.) Now that we have processed the entire input sequence, we are ready to pass the hidden state to the attention decoder.



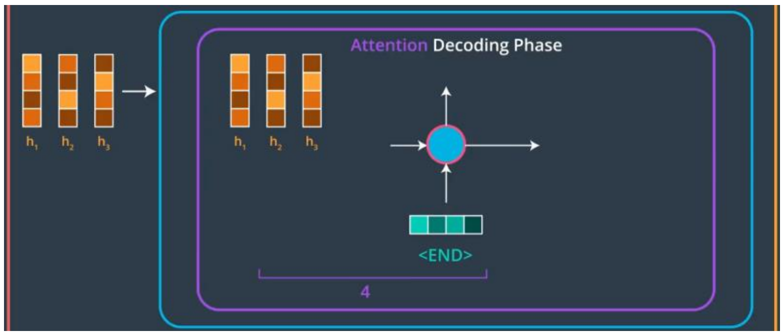
Attention decoder

In Model without attention, we would only feed the last context vector to the decoder RNN, in addition to the embedding of the end toke, and it will begin to generate an element of the output sequence at each time-step.



The case attention different in decoder.

An attention decoder has the ability to look at ***the inputted words****, and the decoders own hidden state, and then it would do the following.*



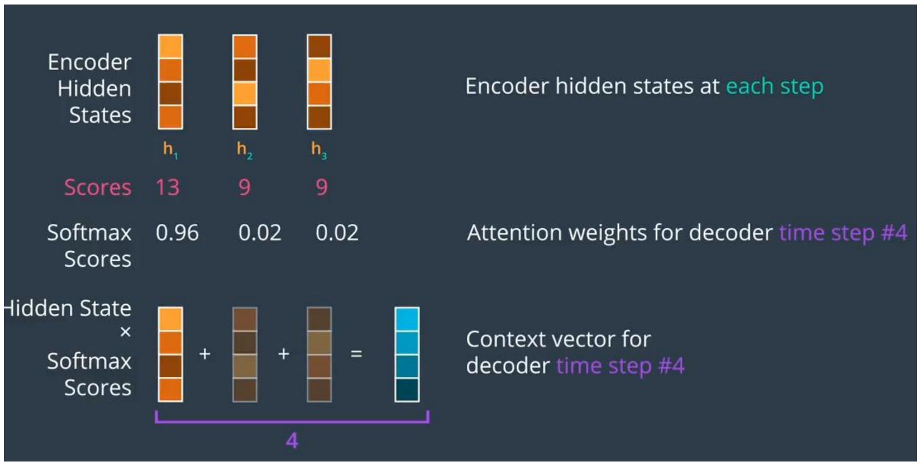
*It would use a <u>scoring function</u> to <u>score each hidden state</u> in the **context matrix**. We will talk later about the scoring function but after scoring each **context vector would** end up with a certain score and if we feed these scores into a <u>SoftMax function</u>, we end up with scores that are all positive, that are all between zero and one, and then all sum up to one.*

*These values are how much **<u>each vector</u>** will be expressed **<u>in the attention vector</u>** that decoder will look at before producing an output.*

***<u>Simply multiplying</u>** * each vector by its **<u>SoftMax score</u>** and then, summing up these vectors' producers an **<u>attention contexts vector</u>**, this is a basic weighted sum operation.*

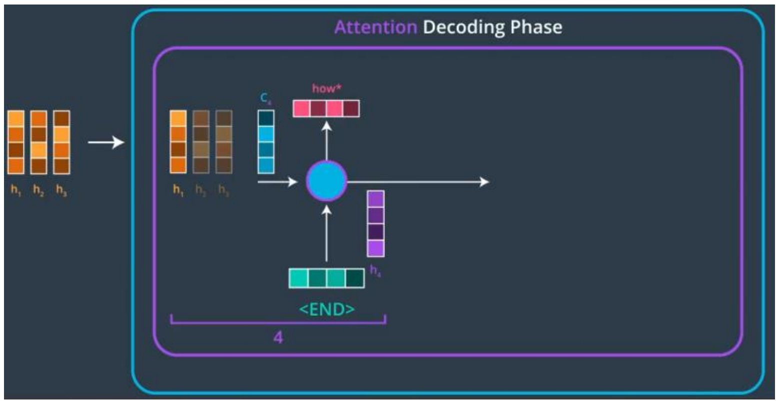
The context vector is ana important milestone in this process, but it`s not the end goal.

We will explain how the context vector merges with the decoders hidden state to create the real output of the decoder at the time-step.


Encoder Hidden States
h_1 h_2 h_3
Encoder hidden states at each step
Scores 13 9 9
Softmax Scores 0.96 0.02 0.02
Attention weights for decoder time step #4
Hidden State × Softmax Scores
+ + =
Context vector for decoder time step #4
4

The decoder has now looked at the input word and at the attention context vector, which focused its attention on the appropriate place in the input sequence.

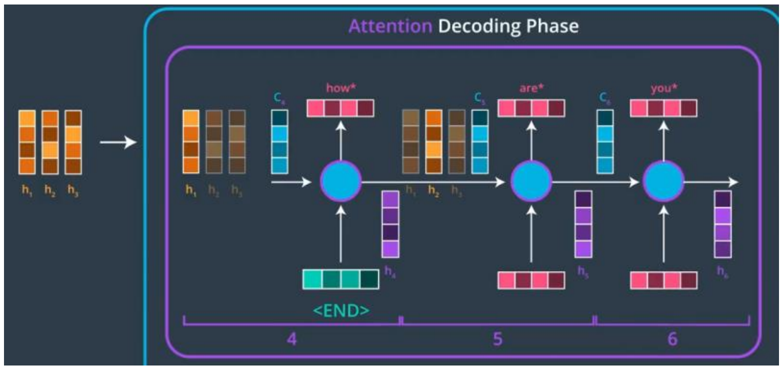
So, it produces a hidden state and it produces the first word in the output sequence.



Now, this is still an over-simplified look, that`s why we have the asterisks here.

There is still a step, whether we will talk about in later video, between the RNN and the final output.

In the next time-step, the RNN takes its previous output as an input, and it generates its own context vector for the time-step, as well as the hidden state from the previous time-step, and that produces new hidden state for the decoder. And a new word in the output sequence, and this goes on until we have completed our output sequence.



Additive and multiplication Attention

NEURAL_MACHINE_TRANSLATION_BY_JOINTLY_LEARNING_TO_ALIGN_AND_TRANSLATE

The scoring function in Bahdanau attention looks like this, where h of j is hidden state from the encoder, s of I minus one is the hidden state of the decoder in the previous time-step.

U of a , W of a , v of a , are all weight matrices that are learned during the training process.

Basically, this is a scoring function, which takes the hidden state of the encoder, hidden state of the decoder, and produces a single number for each decoder time-step.

Bahdanau Attention

$$e_{ij} = {v_a}^\top \tanh(W_a \boxed{s_{i-1}} + U_a \boxed{h_j})$$

If this look too complicated, don`t worry about it. We will get into more detail and a visual explanation as well.

The scores are then passed into SoftMax, this is SoftMax looks like, and then is our weighted sum operation, where we multiply each encoder hidden state, by it score, and then we sum them all up,

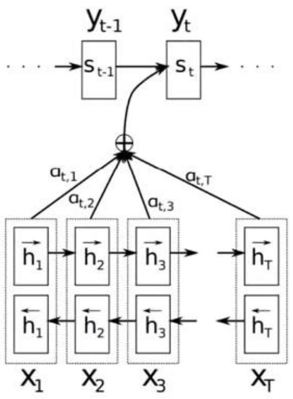
Bahdanau Attention

$$e_{ij} = {v_a}^\top \tanh(W_a s_{i-1} + U_a h_j)$$

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

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

Producing our attention context vector.



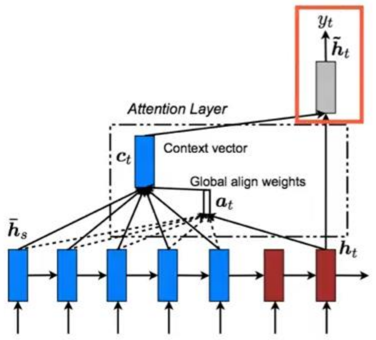
In their architecture, the encoder is a ***<u>bidirectional</u>*** *RNN, and they produce the encoder vector by concatenating the states of these two layers.*

Multiplicative attention or Luong attention,

"Effective Approaches to Attention-based Neural Machine Translation."

By adding a couple more scoring function.

Their architecture is also different and that they used only the hidden states from the top RNN layer in the encoder.



This is allowing the encoder and the decoder to both be stacks of RNNs, which will see later in the applications videos, which let to some of the premier models that are in production right now.

The scoring functions in multiplicative attention are three that we can choose from.

1- *The simplest one is the **dot scoring function**, **which is multiplying the hidden states of the encoder by the hidden state of the decoder**.*
2- *The second scoring function is called general, it builds on top of it and just **adds a weight matrix** between them, and **then multiplication** (dot scouring) the dot product is where multiplicative attention gets its name.*
3- *The third is very similar to Bahdanau attention, in that it adds up the hidden state of the **encoder** with the hidden state of the decoder, and so this addition is where additive attention gets its name, then multiplies it by a weight matrix, and applies a tanh activation and then multiplies it by another weight matrix.*

*So, this is a function that we give the hidden state of the decoder at this time-step and the hidden states **of the encoder at all the time steps**, and it will be produce a score for each one of them.*

We then do SoftMax just as we did before, and then that would produce c of t here , they are called the attention context vector, and so this is the step that would produce the final output of the decoder.

$$\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

$$\begin{aligned} \boldsymbol{a}_t(s) &= \operatorname{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \\ &= \frac{\exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \end{aligned}$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t])$$

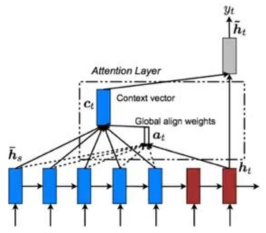
Again, if this doesn`t make a lot of sense right now, don`t worry about it, we will look at it more visually in the next video

This is one of the powerful things that attention does. It gives encoder the ability to look at parts of the input sequence, no matter how far back they were in the input sequence.

Luong Attention

English-German translations

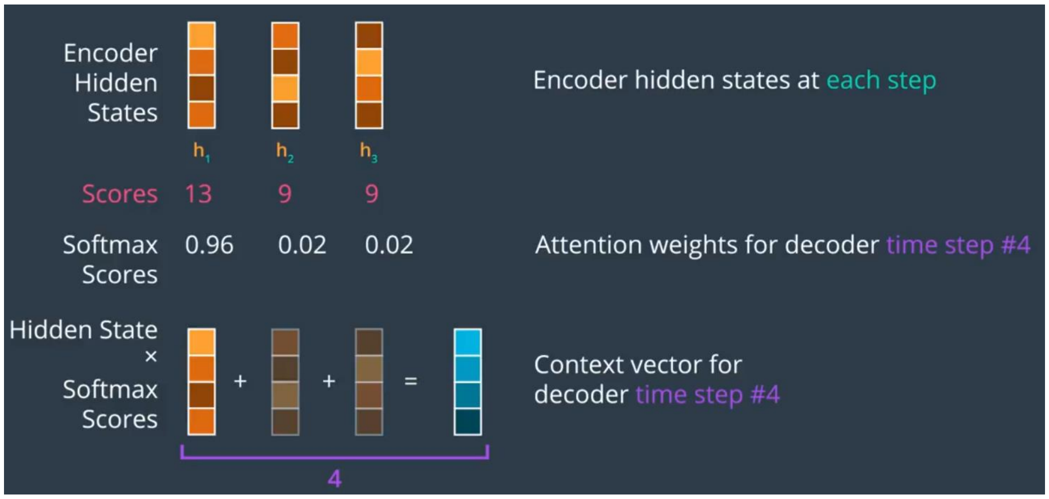
src	Orlando Bloom and Miranda Kerr still love each other
ref	Orlando Bloom und *Miranda Kerr* lieben sich noch immer
best	Orlando Bloom und *Miranda Kerr* lieben einander noch immer .
base	Orlando Bloom und **Lucas Miranda** lieben einander noch immer .



Multiplicative Attention

We look at how the key concept of attention is to calculate an ***attention weight vector****, which is used to amplify the signal from the most relevant parts of the inputs sequence and in the same time, drown out the irrelevant parts.*

We look at the scoring function that produce these attention weights.


Encoder Hidden States
h_1 h_2 h_3
Encoder hidden states at each step
Scores 13 9 9
Softmax Scores 0.96 0.02 0.02
Attention weights for decoder time step #4
Hidden State × Softmax Scores
+ + =
Context vector for decoder time step #4
4

An attention scoring function tends to be a function that takes in

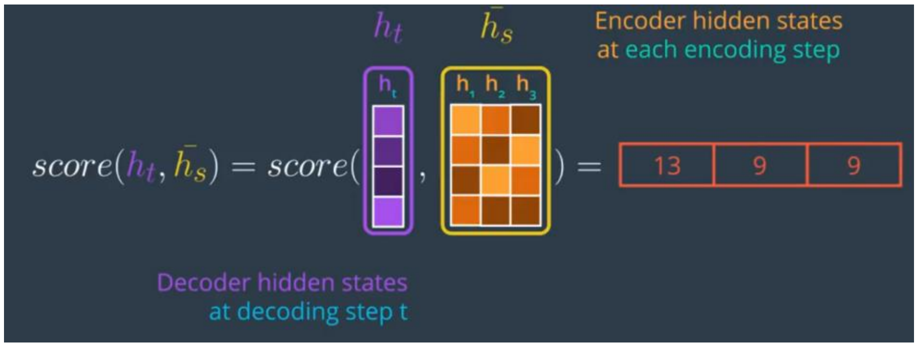
1- *Hidden state of the decoder*
2- ***Set of hidden state of the encoder***

Since this is something we will ***do at each time-step*** *on the* ***decoder side,*** *we only use the hidden state of the decoder at that timestep or* ***the previous timestep*** *in some scoring methods.*

Given these two inputs

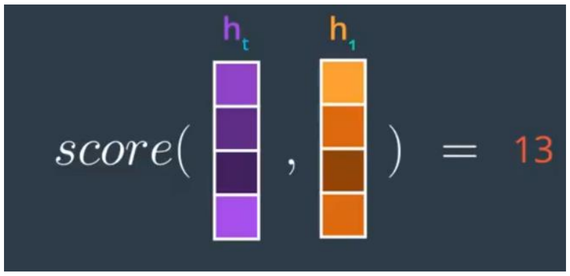
1- ***Vector***
2- ***Matrix***

It produces a vector that scores each of these columns .

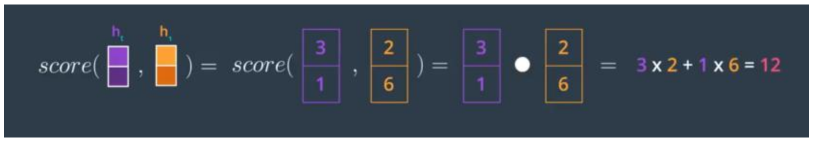

h_t $\bar{h_s}$
Encoder hidden states at each encoding step
h_t
h_1 h_2 h_3
$score(h_t, \bar{h_s}) = score(\ ,\ ) =$
13 9 9
Decoder hidden states at decoding step t

- ***Before looking at the matrix version, which calculate the scores for the all encoder hidden state in one step,***

Let`s simplify it by looking at how to score a single encoder hidden state.

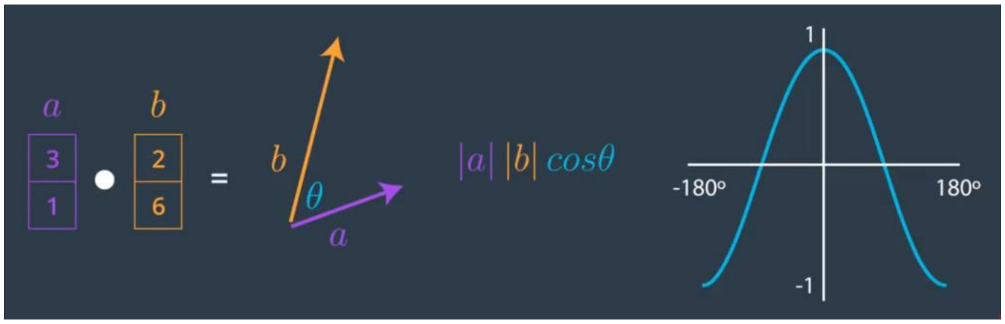


1- *The first scoring method and the simplest is to just calculate the dot product of the two input vectors.(The dot product of two vector produces a single number, so that`s good.*


score(h_t, h_1) = score(3 1, 2 6) = 3 1 ● 2 6 = 3 x 2 + 1 x 6 = 12

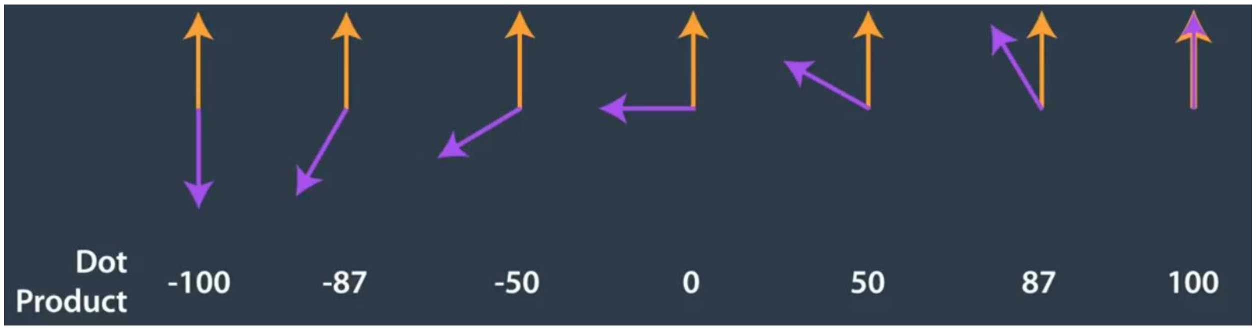
But the important thing is the significant of this number.

Geometrically, the dot product of two vectors is equals to multiplying the length of the two vectors by the cosine of the angle between them, and we know that cosine has this convenient property that it equals one if the angle is zero and it decreases, the wider the angle becomes.



<u>Intuition: Dot products as a similarity measure</u>

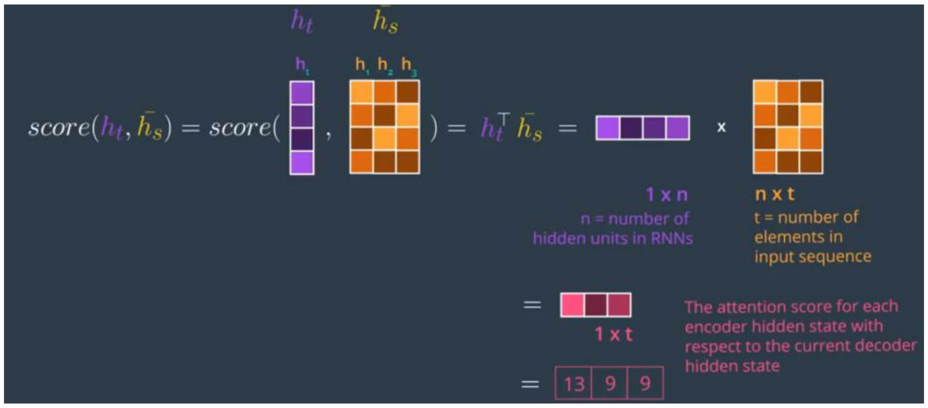
- *What this mean is that if we have two vectors with the same length, the smaller the angle between them, the larger the dot product becomes.*
- *This dot product is a similarity measure between vectors.*
- *The dot product produces a larger number the smaller the angle between the vectors are.*



*In pratice, however, we <u>want to speed</u> up the calcualtion by **scoring all the encoder hidden state** at once, which leads us to the formal mathmetical defincation of dot product attention.*

It is hidden state of the current timesetp transposed times the matrix of the encoder hidden state timesteps.

It will produce the vector of the scores with the simplicty of this method come thr drawback of assuming the encoder and decoder have the same embedding space.


$score(h_t, \bar{h}_s) = score(\ ,\ ) = h_t^\top \bar{h}_s = \times$
h_t
$\bar{h}_s$
h_t
$h_1\ h_2\ h_3$
1 x n
n = number of hidden units in RNNs
n x t
t = number of elements in input sequence
1 x t
The attention score for each encoder hidden state with respect to the current decoder hidden state
= 13 9 9

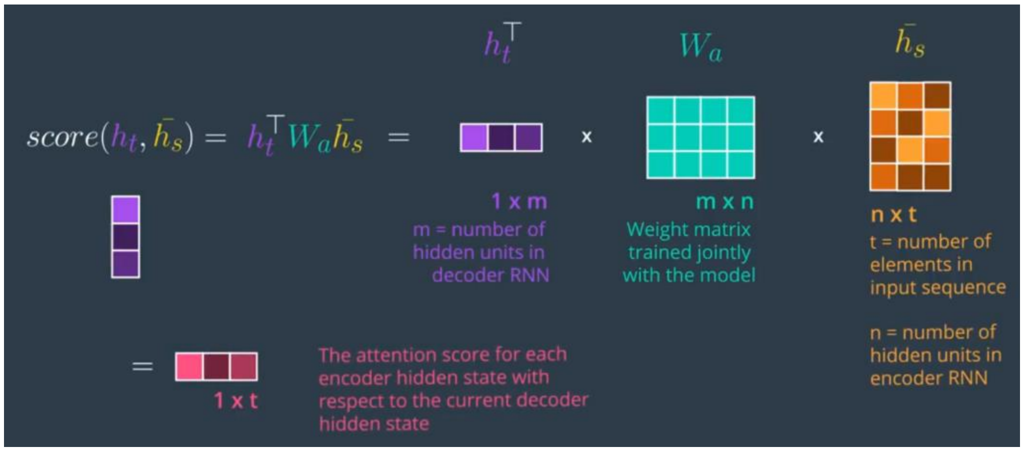
*So, while might work for text summerization for exmaple,where the <u>**encoder and decoder use the same language and the same embeding space**</u>.*

*For machine translateion, however ,you might find that each language tends to have <u>**its own embedding space**</u>.*

This is a case where we might want use the second scoring method , which is a sloght varaiton on the first.

*It simply introduces <u>**a weight matrix between the muliplication**</u> of the docoder hidden state and the encoder hidden state.*

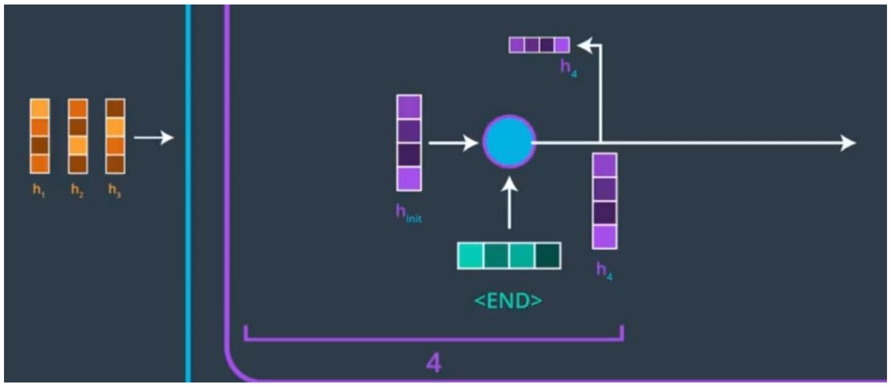
This weight matrix is a linear transofrmaiton that allows the inputs and outputs to use differnet embeddings and the result of this muliplication would be the weight vector.


$score(h_t, \bar{h}_s) = h_t^\top W_a \bar{h}_s$ =
$h_t^\top$
W_a
$\bar{h}_s$
1 x m
m = number of hidden units in decoder RNN
m x n
Weight matrix trained jointly with the model
n x t
t = number of elements in input sequence
n = number of hidden units in encoder RNN
1 x t
The attention score for each encoder hidden state with respect to the current decoder hidden state

Multiplicative attention decoding phase

Lets us now back at the animation and incorporate everything that we know about attention.

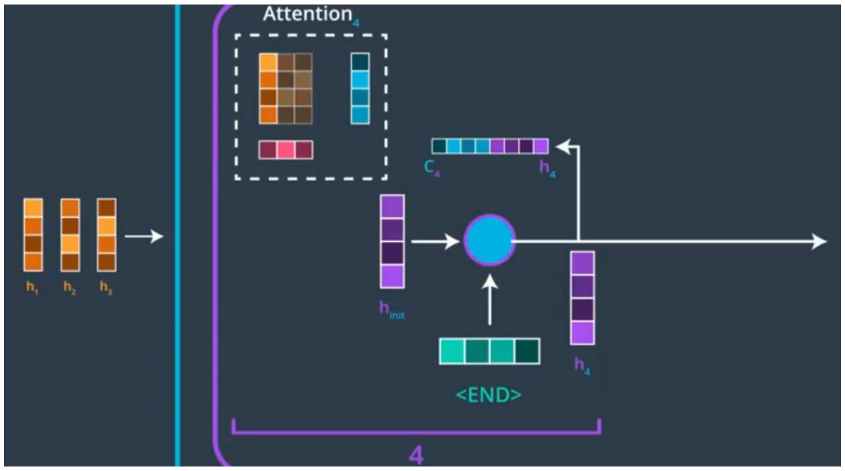
1- *The first time step in the attention decoder starts by talking an **initial hidden state** as well as the embedding for the end symbol. It does its calculation and generates the hidden state at the timestep and here, we are ignoring the actual outputs of the RNN, we are just using hidden states.*



2- *Then we do our attention step. We do that by taking in the matrix of the hidden states of the encoder. (we produce a scoring as we have mentioned.)*

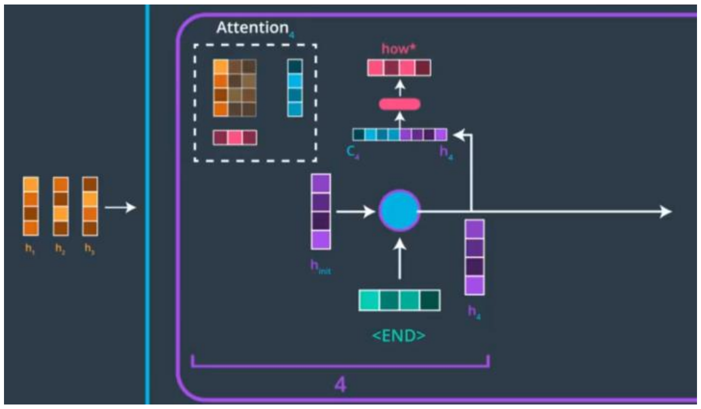
So, if we are doing multiplicative attention, we will use the dot product. In general, we produce the scores, we do a SoftMax,

3- *We multiply the SoftMax scores by **each corresponding** hidden state from the encoder, and we sum them up producing **our attention context vector** and then what we do next is this, we concatenate the attention context vector with the hidden-state of a decoder at the timestep so h4 So this would be c4 concatenated with h4, that`s what we will do here.*



So, we basically glued them together as one vector and then we pass them through a fully connected neural network which is basically multiplying by the weight matrix Wc and apply a tanh activation.

The output of this fully connected layer would be our first outputted word in the output sequence.



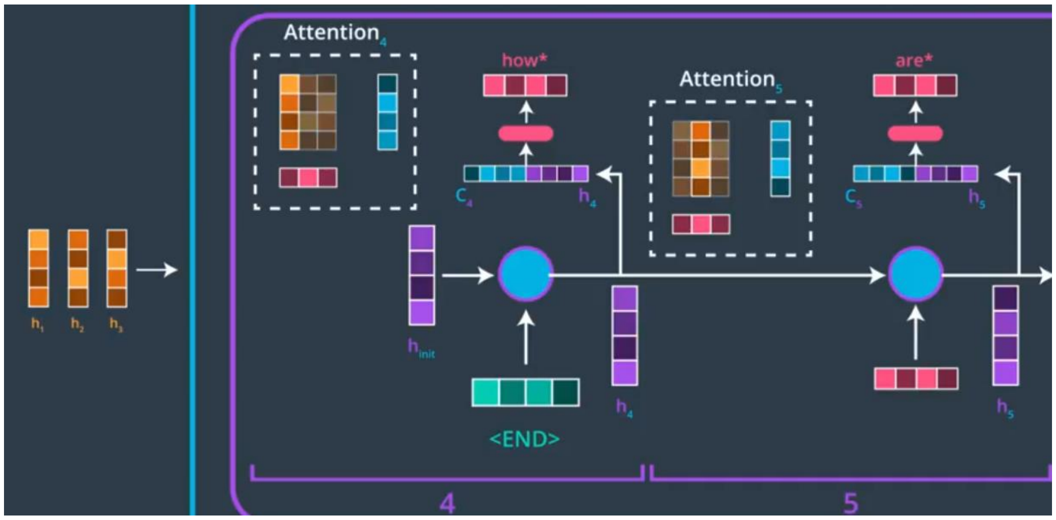
$$score(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & dot \\ h_t^\top W_a \bar{h}_s & general \\ v_a^\top \bar{h}_2\, tanh(W_a\ [h_t; \bar{h}_s]) & concat \end{cases}$$

$$a_t(s) = align(h_t, \bar{h}_s) = \frac{exp(score(h_t, \bar{h}_s))}{\sum_{s'} exp(score(h_t, \bar{h}_{s'}))}$$

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

$$\tilde{h}_t = tanh(W_c [c_t; h_t])$$

We can now proceed to the second step, passing the hidden state to it and talking the output from the first decoder timestep. We produce h5. We state our attention at this step as well , we score, we start our attention we score and we produce a weight vector, we do SoftMax , we multiply , we add them up producing c5 The attention context vector at step five, we glue it together with the hidden state we pass it through the same fully-connected network with tanh activation function producing the second word in our output and this goes on until we have completed outputting the output sequence.



<u>Attention scoring function | 3-Concat</u>

We looked at the third score method it is called concat , and the way to do it is to use a feedforward neural network.

<u>Take a simple example</u>

Let`s say we are scoring this encoder hidden-state, at fourth time-step at the decoder.

Again this is an oversimplified example scoring only one, while in practice we will actually do a matrix and do it all discord in one step.

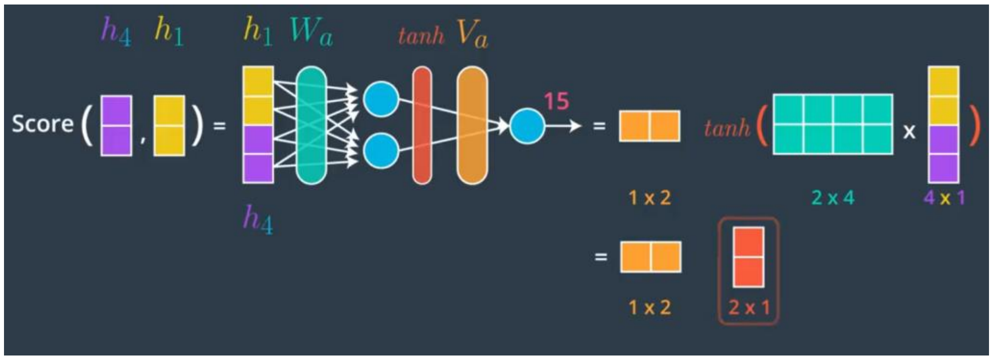
The concat scoring method, is commonly done by concatenating the two vectors, and making that the input to a feedforward neural network.

Let`s see how that works. So, we merge them, we concat them into one vector, and then we pass them through a neural network. This network has a single hidden layer, and the output this score.

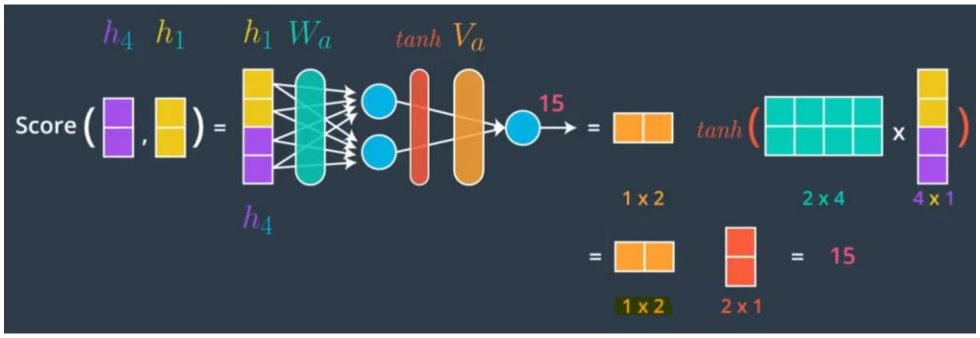
The parameter of this network, are learned during the training process.

Namely the Wa weight matrix, and the Va weights matrix. To look at the how the calculation is done.

This is our concatenated vector, we simply multiply it by Wa , and apply ten-H activation producing this two by one matrix.



We multiply that by the Va weights matrix, and we get the score for this encoder hidden state.



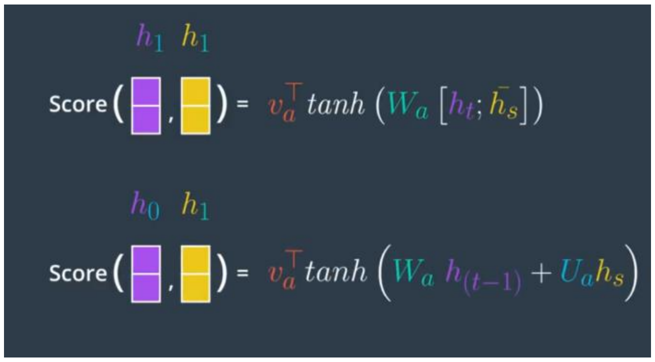
Formally, it is expressed like this, where H of T as we have mentioned is the hidden state at at the current time-step, and H of s is the collection of the set of encoder hidden states.

$$= v_a^\top \tanh\left(W_a\left[h_t;\boxed{\bar{h}_s}\right]\right)$$

This is the concatenation and then we multiply it by W-of-A, tanh activation and then multiply it by V of a transpose.

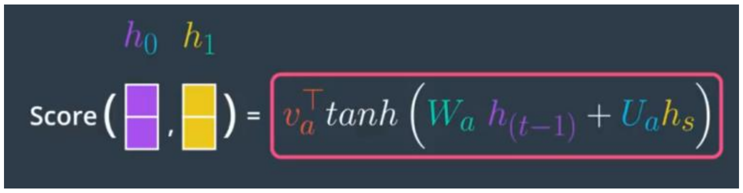
One thing to note is the difference.

So concat is very similar to the scoring method from Mark Daniel paper,


h_1 h_1
Score
$= v_a^\top tanh\left(W_a\left[h_t;\bar{h}_s\right]\right)$
h_0 h_1
Score
$= v_a^\top tanh\left(W_a h_{(t-1)} + U_a h_s\right)$

But this is one, this is concat method from the Lung paper, where is the only one weight matrix.

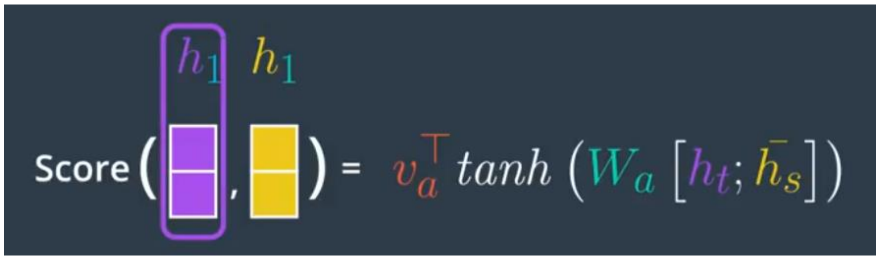
In the Mark Daniel paper there are two major differences that we can look at.



One of them is that the weights matrix is split into two, so we don`t have just of W of A, and we have U of a, and each is applied to the respective vector. The decoder hidden state in this case, and the encoder hidden state at this case.

Another thing to note is that the mark Daniel paper used the hidden state from the previous from the previous time-step at the decoder.

--While the loop paper it uses the one from the current time-step at the decoder.

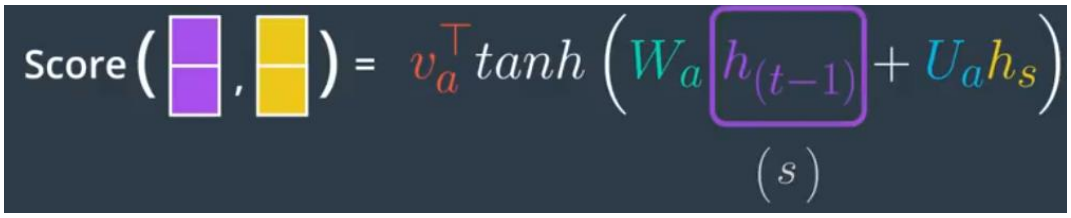


Let`s make a note on notation here, in case you are planning to read paper, here we have used the notation mainly from the Lung paper where we referred to the encoder and the decoder hidden state sa H So,

H of T for the decoder, and H of S for the encoder

This means so H is for hidden state, T is for Target, so that`s the target sequence that we are going to output so that`s associated with the decoder. S is for source,

In the baghdana paper, this is called S. So it is not H , it is called S. So now



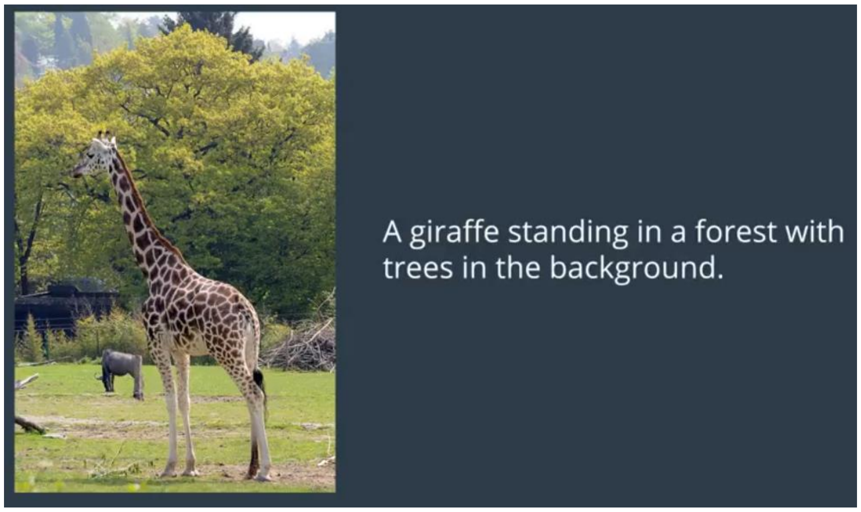
Computer Vision application and tasks that attention in-power

We will focus on image captioning and one of the key papers from 2016 titled, "Show, attend and Tell."

This paper presented a model that achieved the state of art performance in caption in caption generation in a number of datasets.

For example, when presented with an image like this, the generated caption was

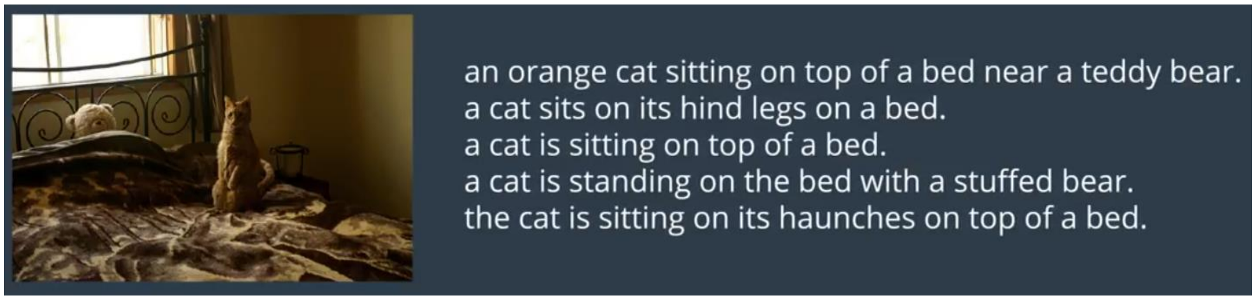
When presented with the image like this, the generated caption was


A giraffe standing in a forest with trees in the background.

Models like these are trained on a dataset like the Ms-Coco, which has a set of abl=out 200,000 images each with five captions written by people.

This is sourced through something like Amazon`s Mechanical Turk Service.

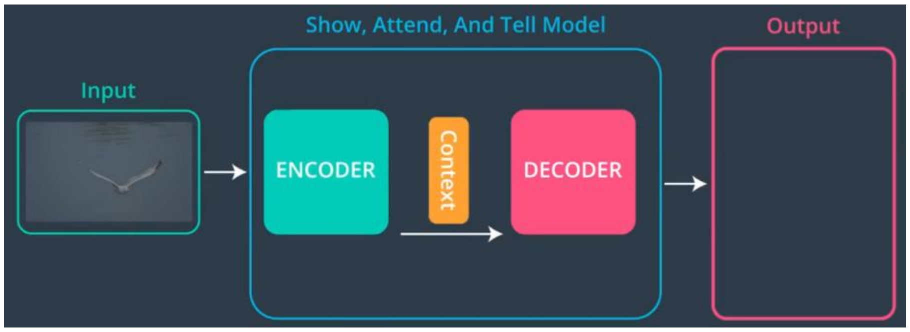
For example, the image from the dataset has these five captions as it`s labels and this the dataset that we used to train a model like this.



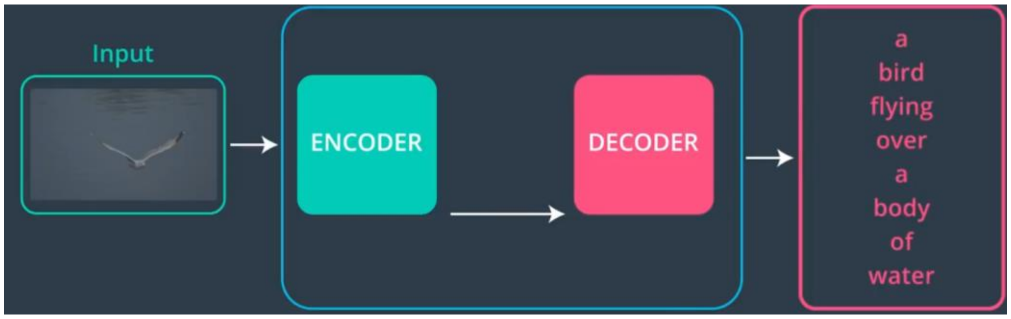
an orange cat sitting on top of a bed near a teddy bear.
a cat sits on its hind legs on a bed.
a cat is sitting on top of a bed.
a cat is standing on the bed with a stuffed bear.
the cat is sitting on its haunches on top of a bed.

If u look the model is very sillier to the sequence to sequence models we have looked at earlier in the lesson.

In this case, the model takes the image as an input to it`s encoder, the encoder generate a context,

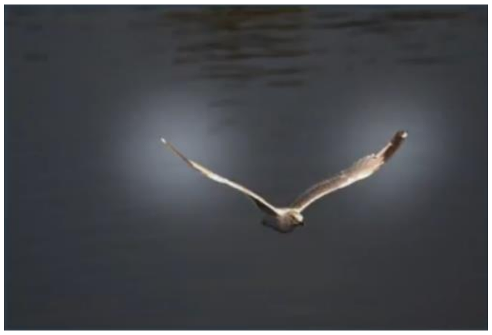


Passes it to the decoder. The decoders to output a caption.

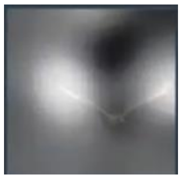


The model generates the <u>caption sequentially</u> and <u>uses attention</u> to focus on the appropriate place of the image as it generate each word of the caption.

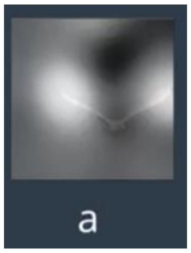
For example, when presented with this image, in the first step, the trained model focues on this region,



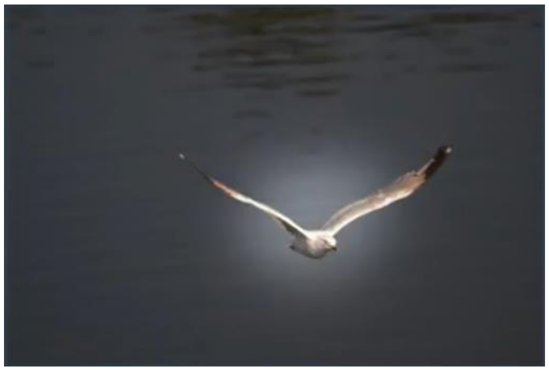
So, this is the thumbnail of this image. The white areas are where the model is paying the most attention right now. So, we can see that it`s mainly focused on the wings.



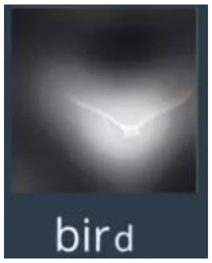
It then outputs the first element in the output sequence or the caption, which is the word "a."



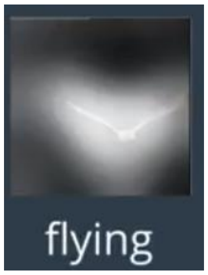
The next step at the decoder, it focus on this region So mainly the body of the bird as you see, and the output would be "bird"



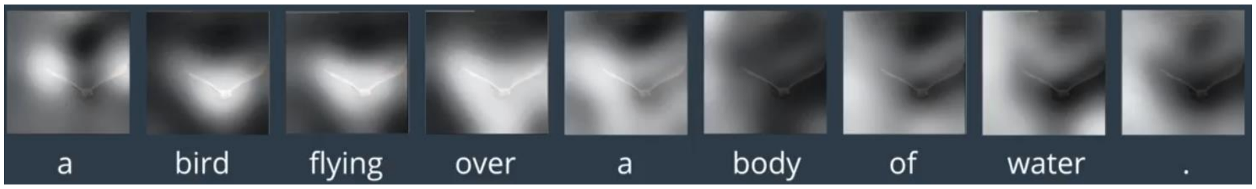
And then it expands it`s focus area to this area around the bird to try to figure.



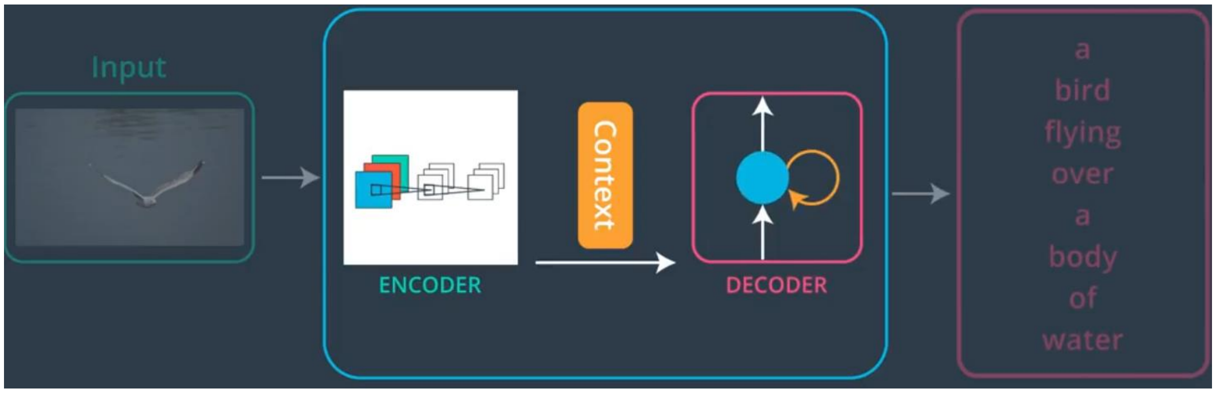
out what to describe next, and the output at this step is "flying".



These goes on. We can see how it`s attention. Right now is starting to radiate out of the bird. And focus on things behind it or around it. So, it`s generating " a bird flying over a body of " and then the focus here completely sort of ignores he bird, and trying to look at everywhere eels in the image, "water"



The first time I looked at something like this, image captioning specifically , it was mind-blowing to me, but now, we have an idea of how it works.

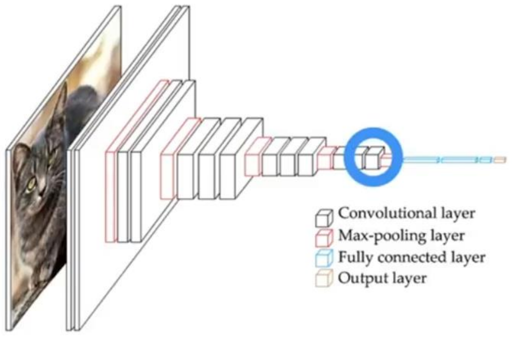


The model here is made of an encoder and a decoder as we are mentioned .

1- ***The encoder in this case is a convolutional neural network that produces a set of feature vectors, each of which corresponds to a part of the image or feature of the image.***

To be more exact, the paper used a VGG net Convolutional network trained on the image net.

The annotation was created from this feature map.



- ***This feature volume has dimension of 14 x 14 x 512 , meaning that it has 512 feature , each of them has the dimension of 14 x 14.***

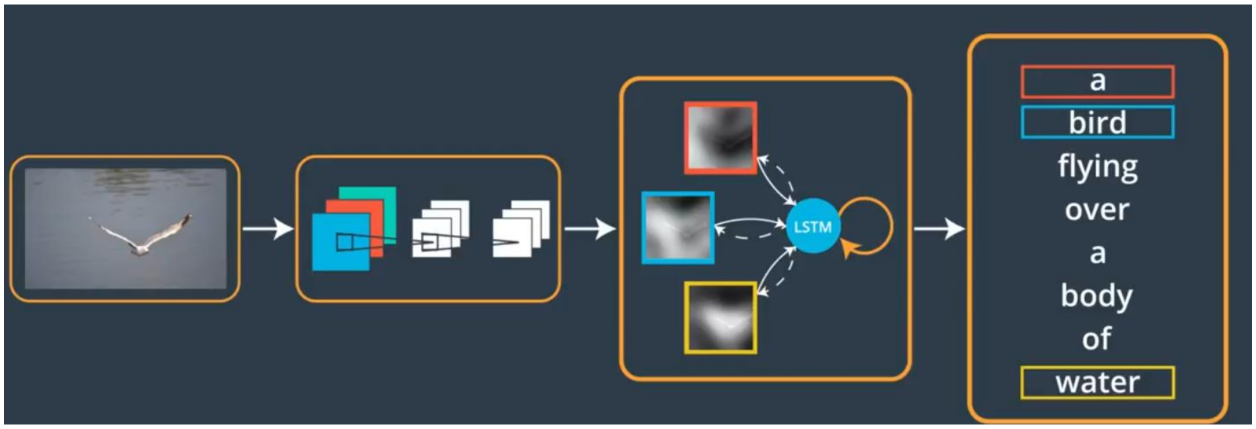
<u>To create our attention vector, we need to flatten each feature, turning it from 14 x 14 to 196 x 1 .</u>

<u>So, this is simply reshaping the matrix. After we reshape it to end up with a matrix of 196 x 512. So we have 512 feature, each one of them is a vector of 196 numbers. So, this is " context vector "</u>

We can proceed to use it just like we have used the <u>context vector</u> in the previous video, where we score each of these features and the them, we merge them to produce our attention context vector.

<u>The decoder is a recurrent neural network</u>, which use attention to focus on the appropriate annotation vector at each time-step.

We plug this into the attention process we have outlined before and that`s our image captioning model.



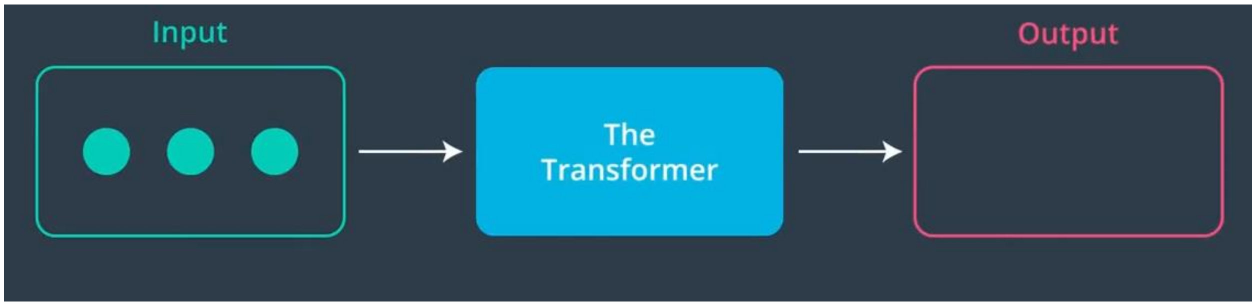
Other attention technique

While the two mechanism continue to be commonly used, there have been significant developments over the years.

This paper noted that the complexity of encoder-decoder with attention model can be simplified by adopting a new type of model that only uses attention no RNN.

They called this new model the Transformer. In two of their experiment on machine translation task,

The model proved superior in quality as well as requiring significantly less time to train.



The Transformer takes a sequence as an input and generate a sequence, just like the sequence-sequence models we have so far.

The difference here, however , that it does not take the inputs one by one, as in the case of an RNN.

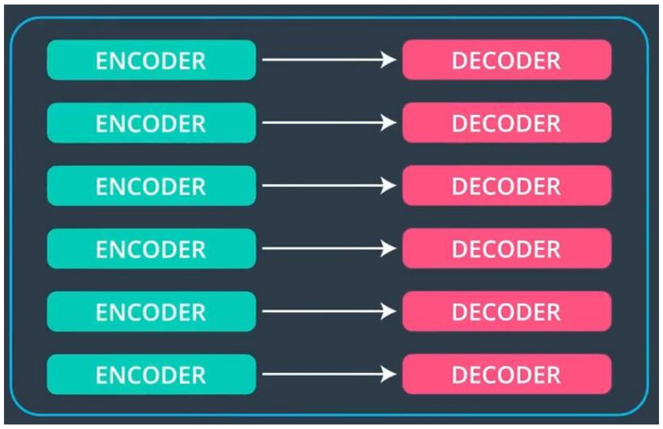
It can produce all of them together in parallel.

Perhaps each element is processed by a separate GPU if we want. It then produces the output one by one but also not using AN rnn.

The Transformer model also break-down into an encoder and a decoder. But instead of RNN, they used feed-forward neural networks and a concept called self-attention.

This is combination allows the encoder and decoder to work without RNN, which vastly improves performance since It allows parallelization of processing that was not possible with RNN.

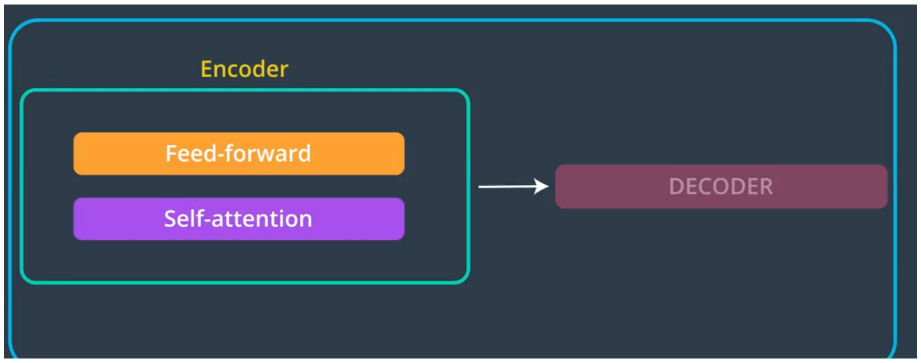
The Transformer contains a stack of identical encoders and decoders.



Six is the number the paper propose.

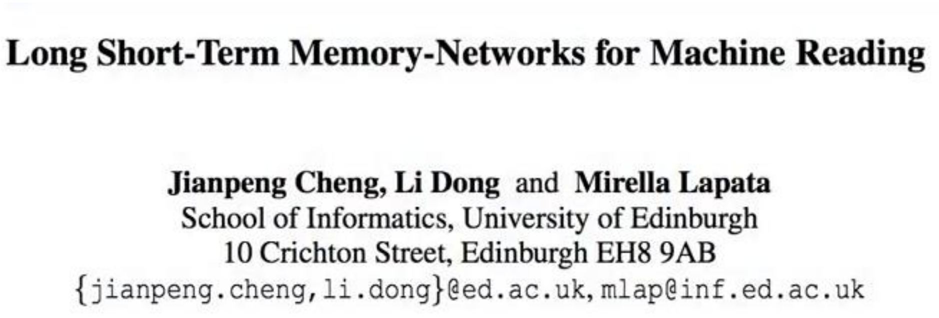
The Transformer

1- ***Each encoder layer contains two sublayers: <u>a multi-headed self-attention layer and a feed-forward layer</u>. This attention component is completely on the encoder side as opposed to being a decoder component like the previous attention mechanisms we have seen.***



This attention component helps the encoder comprehend its input by focusing on other parts of the <u>input sequence</u> that are relevant to each input element it processes.

<u>This idea is an attention of work previously done</u> on the concept of self-attention and how it can aid comprehension.



Long Short-Term Memory-Networks for Machine Reading

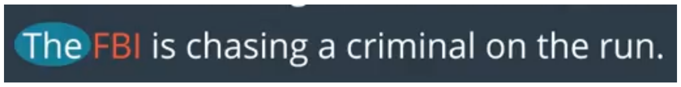
Jianpeng Cheng, Li Dong and **Mirella Lapata**
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB
{jianpeng.cheng,li.dong}@ed.ac.uk, mlap@inf.ed.ac.uk

In this paper, for example, this type of attention is used in <u>the context of machine reading</u>, where the experiments on this technique matched or outperformed the state-of-art at the time in tasks like language modeling, <u>sentiment analysis and natural language inference</u>.

They still used RNNs but they augmented it with the idea that later become self-attention.

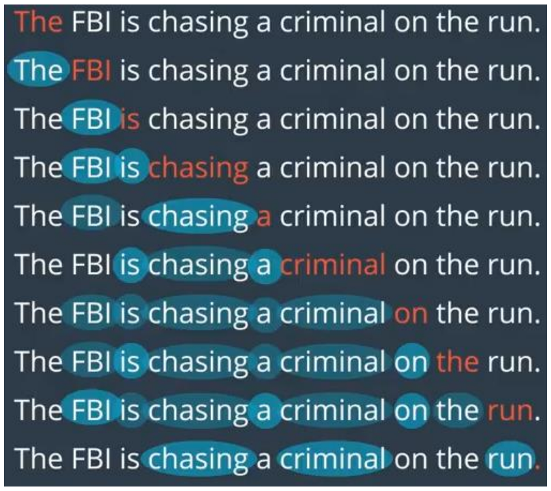
The example they used in this machine reading paper shows where the train model pays <u>attention as it reads each word.</u>

So, far example , when the model reads the sentence using an LSTM, it learn which other parts of the input to pay attention to as it processes each word of the input.



So, the red is where it`s reading and the blue is where it`s pay attention as it`s reading this word. At each step, it reads a word and it pays attention to the relevant previous words that would aid in comprehending that word.

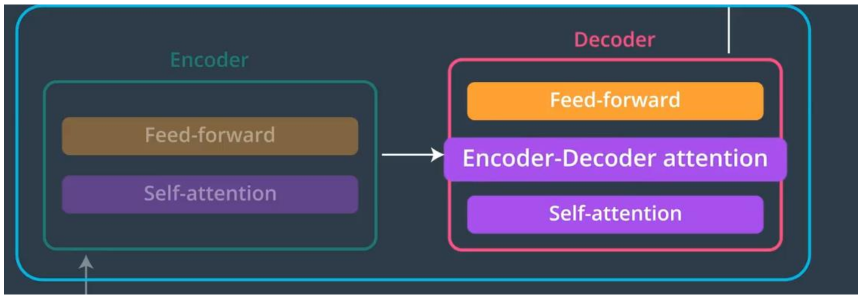
The structure of the transformer, however, allows the encoder to not only focus on previous words in the input sequence, but also on words that appeared later in the input sequence.



This, however is not the only attention component in the Transformer. The decoder contains two attention components.

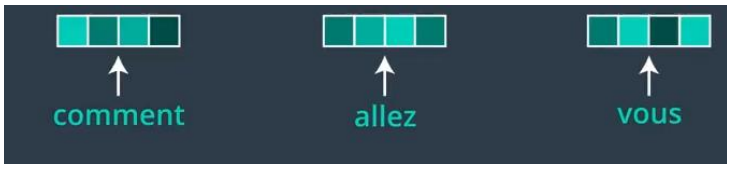
1- One that allows it to focus on the relevant part of the inputs and another that only pays attention to previous decoder outputs, and there you have it

A high-level view of the component of the Transformer. We can see how extensively this model uses attention. We can see three attention components here. They don`t work exactly the same way, but they all boil down pretty much to multiplicative attention.

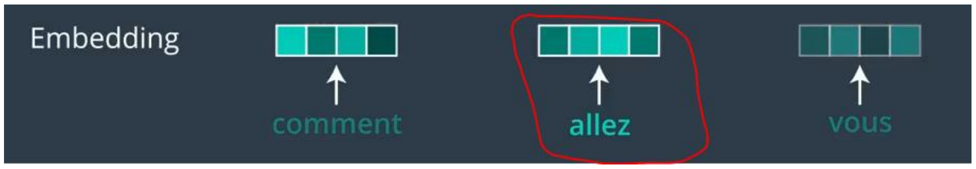


<u>The transformer and self-Attention</u>

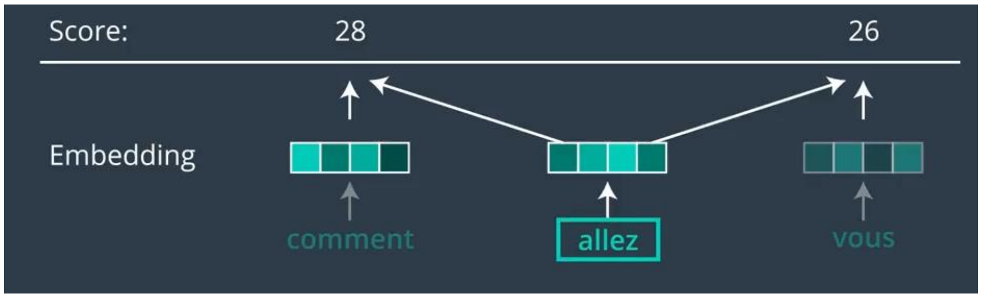
Let`s say we have these words that we want our encoder to read and create a representation of.



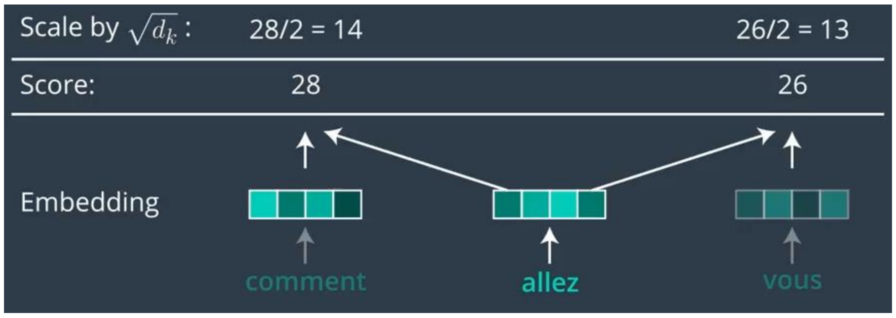
As always, we begin by embedding them into vectors. Since transformer gives us a lot of flexibility for parallelization, this example assumes we are looking at the process or GPU tasked with encoding the second word of the input sequence.



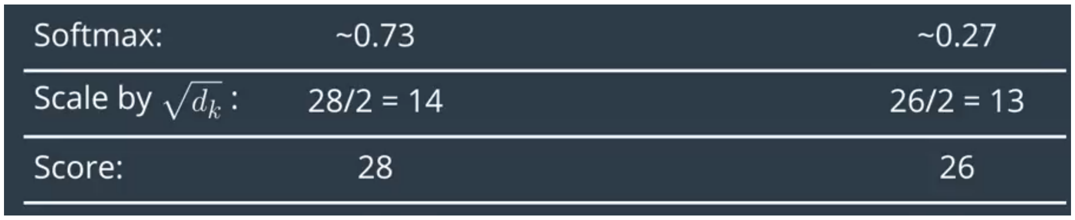
*First step, we compare them. So, we score the embedding against each other. So, we have a score here and a score here, comparing this word that we are currently **<u>reading</u>** or **<u>encoding</u>** with the other words in the input sequence.*



We scale the score; we basically divide by the two here. The dimension of the key, which we are using a toy dimension of four so that would be two.



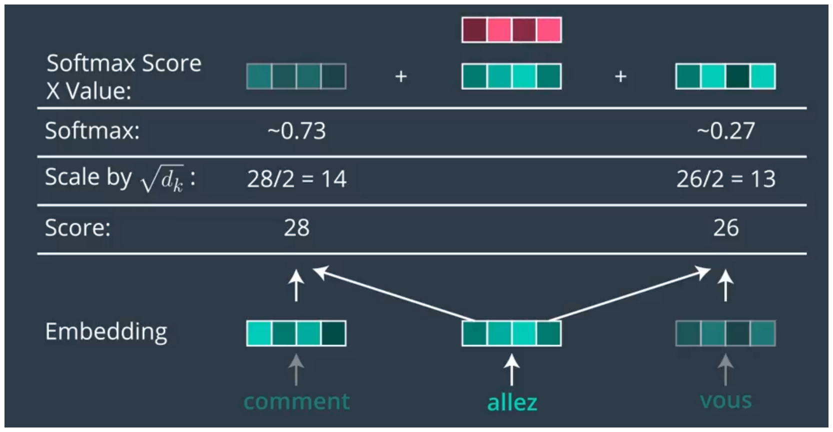
We do a SoftMax for these, and then,



Softmax:	~0.73	~0.27
Scale by $\sqrt{d_k}$:	28/2 = 14	26/2 = 13
Score:	28	26

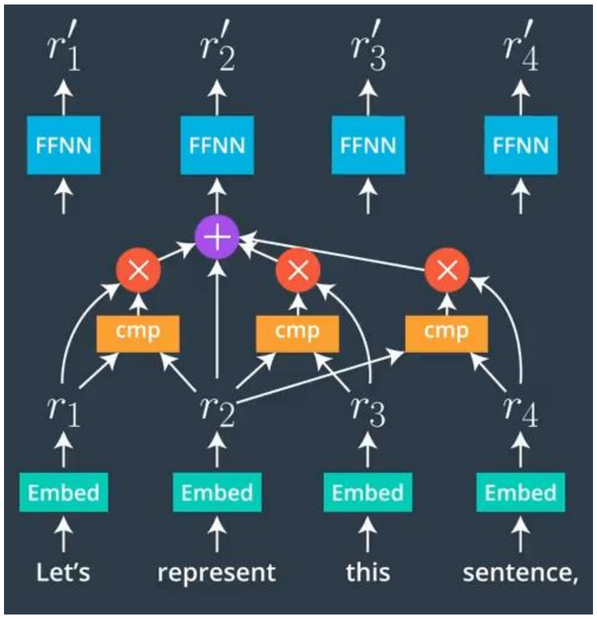
*We multiply the **<u>SoftMax score with the embedding</u>** to get the level of expression of each of these Vector.*

The Embedding of the current word, just [inaudible] as it is. We add them up, and that produces the self-attention context vector, if that`s something we `d like to call it.



The Transformer and Self-attention

This is imaging the authors of the paper showed whey they presented this paper first at the NIPS Conference. Then, we are looking at the second word.



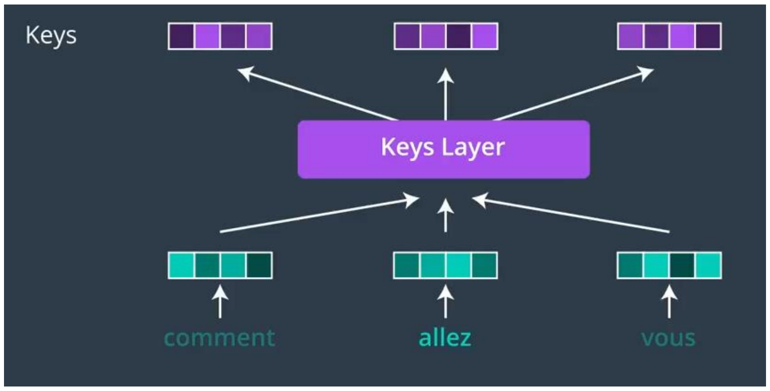
Then, we are looking at the second word. So,

1- *we have the words here, and then, then have their embedding*
2- *and then these are the vectors of the embeddings.*
3- *So, this word is cmp or scored against each of the other word in the input vector.*
4- *The score is then multiplying by the embedding of that relevant word, and then, all of them are added up. (We don`t score the current word. We scored the other words.)*
5- *After we add them up, we just pass this up to the feed-forward neural network.*

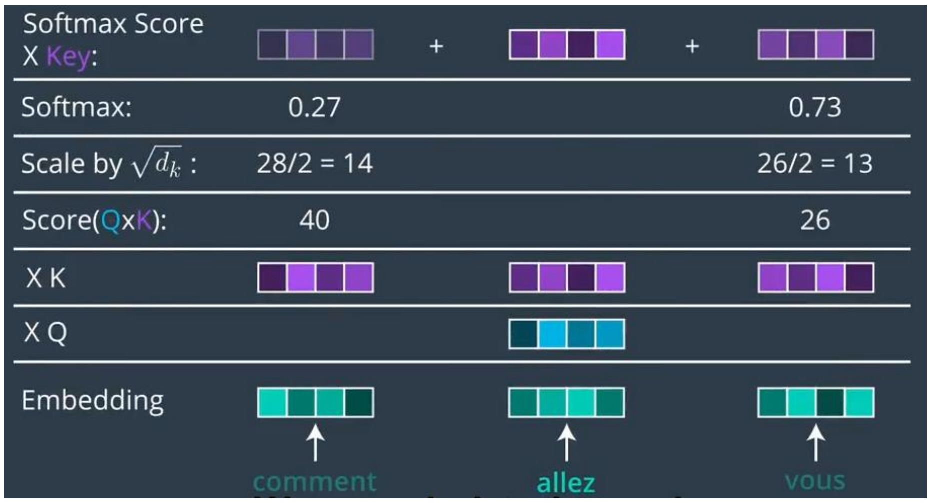
If you implement it like this, however, we could see that the model is mainly focusing on other similar words,

- *If we judge it only on the embedding of the word. So, there`s a little modification that we need to do here.*

We need to create queries out of each embedding. We do that by just multiplying by a ***query matrix*** *or just passing it through a* ***query feedforward neural network****. We also create keys. So, we have another separate matrix of keys.*

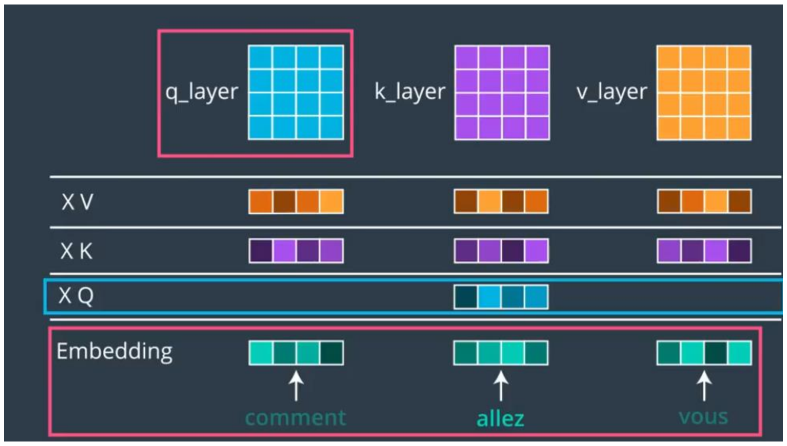


We can calculate that again.



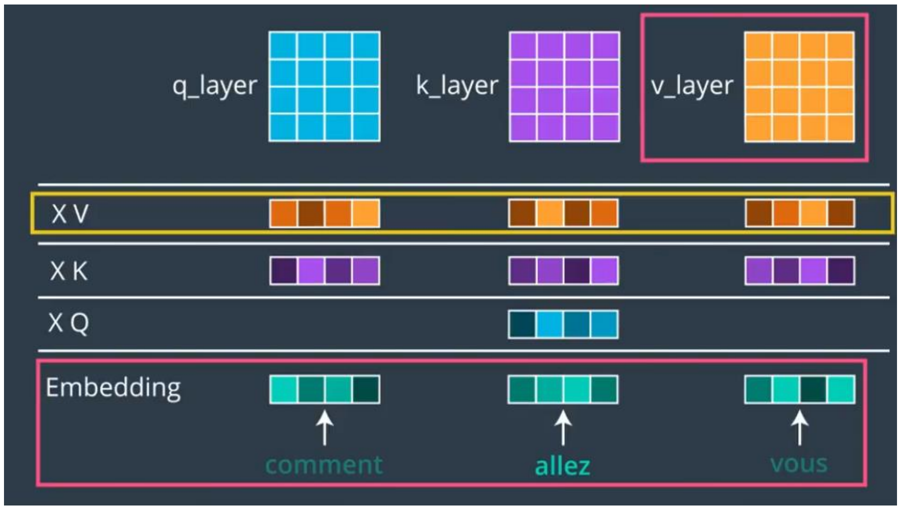
1- *So, we have our embedding*
2- *Create the queries.*
3- *We only processing the second word here. So, that where we created the query for.*
4- *Then we have our keys here.*
5- *The scoring is comparing the query versus the key. (So that`s where we get these number here 40 , and then 26.)*
6- *We scale SoftMax*
7- *We multiply the SoftMax score with the key. And that get the self-attention context vector after we all these together.*

This is an acceptable way of doing it, but there is a variation that we need to look at as well. So these are our embedding. We have our quires, which are made by multiplication the embedding by the Q matrix, which is learned from the training process.

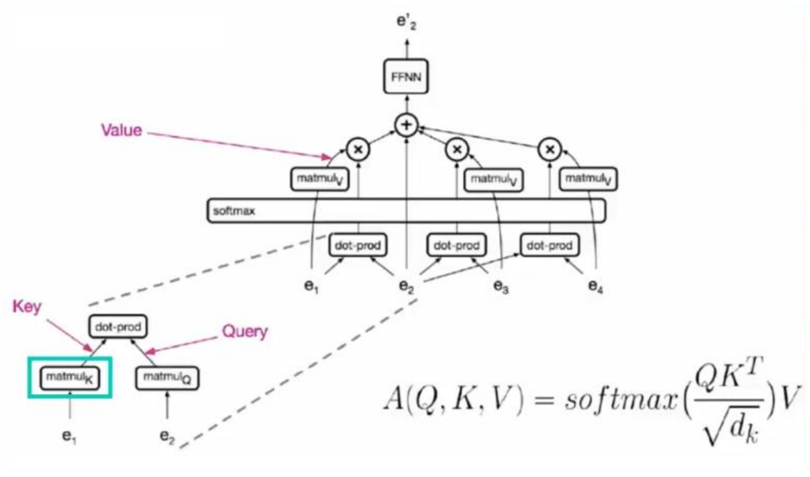


We have our keys, which are create by multiplying the embedding by the K matrix, and we have our values,

Which are produced the same way by multiplying by the V matrix, which is also learned is the training process.



This is graphic from the authors as well in their NIPS presentation, where they outline how to create the key, the query and the value. So, this is the embedding. We multiply it by V to get the value. We multiply it by Q to get the query. You multiply it by K to get the key.



So, the final of self-attention, as presented in this paper, is we have our embeddings.

1- *We have calculated our V, Q our values , key , and quires .*
2- *We score the quires against the keys,*
3- *And then the SoftMax score is multiplied by the values.*
4- *These we add up and pass to the feedforward neural network. This is a very high-level view at this model here and discussion of the self-attention concept.*