Understanding Attention In Deep Learning

How a little attention changed the AI game!

Ria Kulshrestha · Follow

Published in Towards Data Science · 9 min read · May 8, 2020

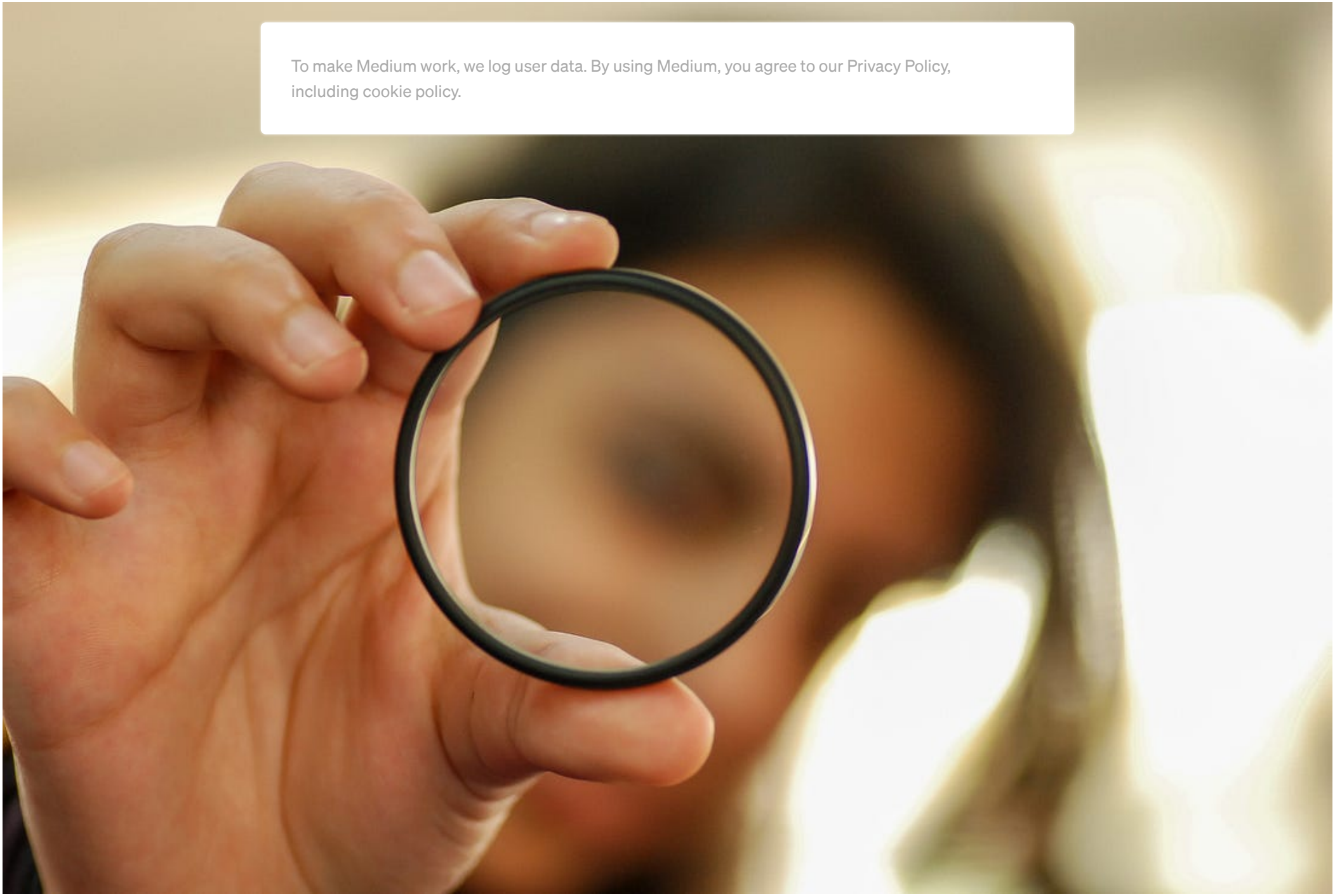


Photo by Haneen Krimly on Unsplash

In important assets — we as humans — have. How we choose to spend our time and on what we choose to focus on during that time determines the outcome for almost all our endeavors.

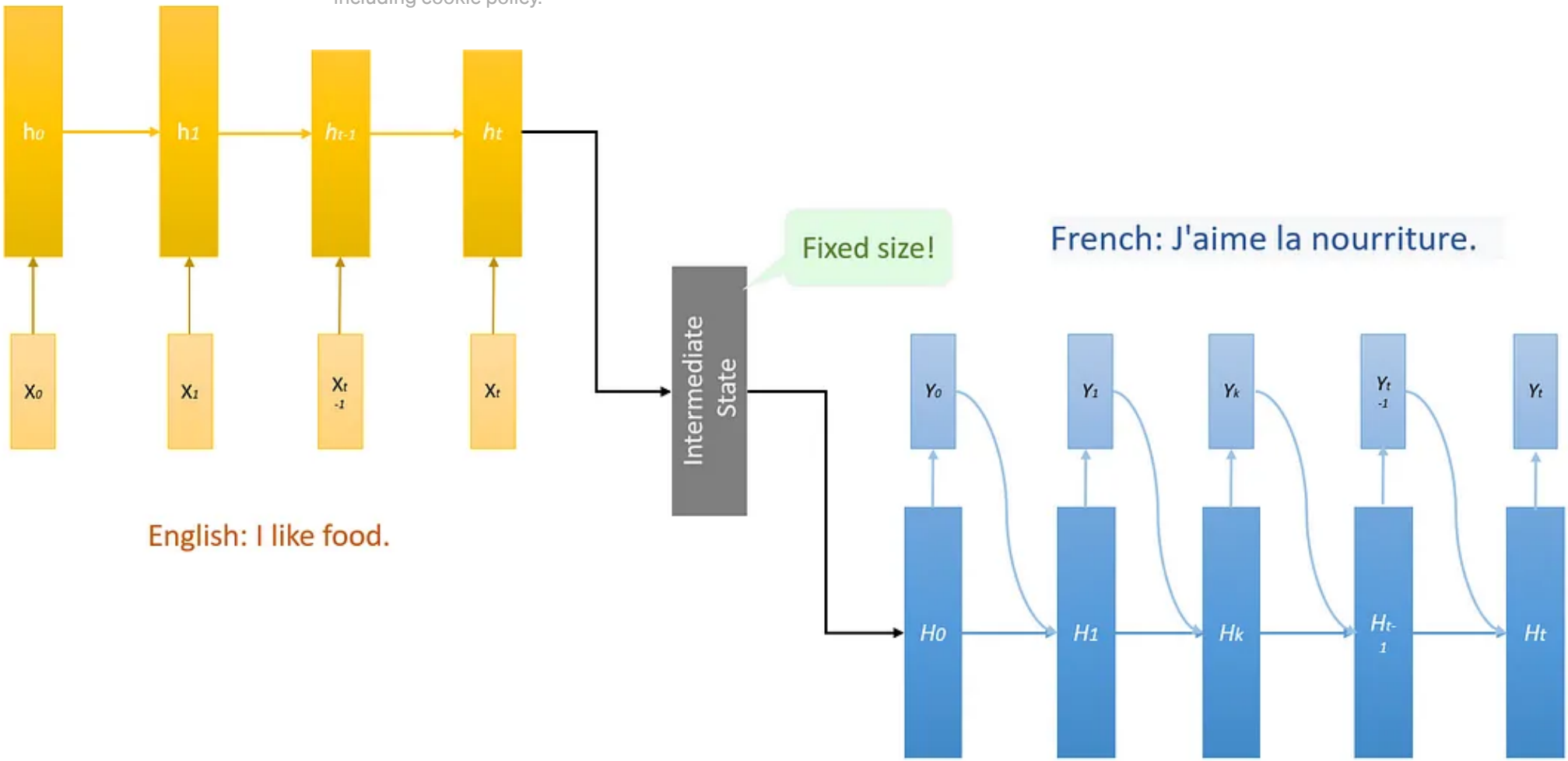
In this article, we will discuss how we gave an ML model the ability to focus and the impact it had on its performance.

Why models need to have attention

Let's go over a task — popularly solved in many NLP models — translation. A word to word translation doesn't work in most cases as most languages don't share a common sentence structure. A simple example:

```
English => French
red => rouge
dress => robe
“red dress” => “robe rouge”
Notice how red is before dress in English but rouge is after robe.
```

intermediary information and expressing it in the output language. The size of the vector used for this intermediary state i.e. the state capturing all the information from input sequence — before we start decoding the output sequence— is fixed.



How NLP models translate from input to output sequence. The input sequence and the encoder model are shown in yellow, while the output sequence and the decoder model are shown in blue. **Note the t in the encoder model has no correlation with the one in the decoder model, and is just used to represent the next time step in each of the models.**

critical to the quality of decoded output. Whether it is a translation or a QnA task, in which the input is a question and a paragraph and the model needs to predict the answer to that question based on the paragraph or, any other sequence to sequence modeling operation, the intermediary state continues to be the most crucial piece of the puzzle.

> *It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way — in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.*
>
> *- A Tale of Two Cities, Charles Dickens.*

Now I ask you to just memorize, not even translate, this sentence after going over it once, from left to right, and I limit the number of words that you can

write in your notes. Not so easy, is it?

How [totally made up? *Disclaimer: The numbers and use cases appearing in the following statement are fictitious, any sense they make is purely coincidental.*

> *This car is 2m in **height** — so lean on my friend, 12m in **width** — not very hug-able unfortunately, with a 8m **wheelbase** — whatever use that information is for, a **turning radius** of just 0.1m — making it easier to turn away from all your problems, a **boot-space** of 200 litres — for all the luggage you'll carry for a trip and never use, a **ground clearance** of 0.25m — in case you ever seek refuge under the car, a 6 **cylinder engine** with 5 **valves** — a spec you'll only ever use while showing off and lastly has a **dual overhead camshaft** because big cars comes with big words.*
> *-Yours truly :)*

Jokes aside, in this case not only you have to remember all the numbers I randomly threw in there, but also the features —shown in bold — it corresponds to. Mess it up at one place and you have got it all wrong.

As evident by now, in **cases with very long sentences as input, the intermediate state fails and is not sufficient to capture all the information.**

Often it will forget the first part by the time it completes processing the

wh

in

information available to the decoder in a fixed length vector and from being a potential bottleneck, we use the concept of ***attention***. With this new approach, the information can be spread throughout the sequence of annotations — encoder hidden states, which can be selectively retrieved by the decoder accordingly.

Attention mechanism in a model

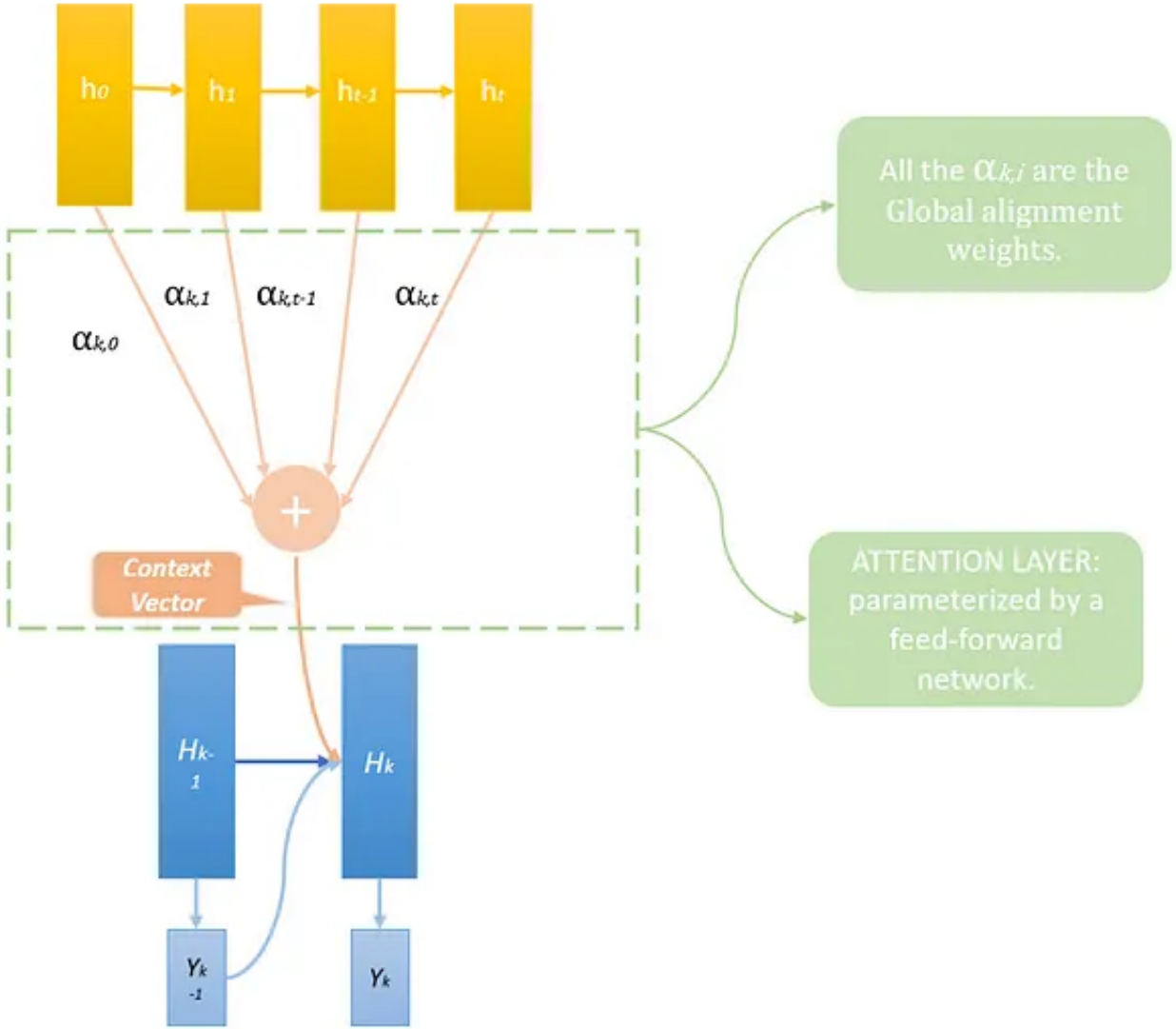
Attention mechanism tries to overcome the information bottleneck of the intermediary state by **allowing the decoder model to access all the hidden states**, rather than a single vector — aka intermediary state — build out of the encoder's last hidden state, while predicting each output.

$$H_k = f(H_{k-1}, Y_{k-1}, C_k)$$

Calculating the next hidden state for the decoder model.

The input to a cell in decoder now gets the following values:

2. The previous output of decoder model Y_{k-1}.
3. A context vector C_k— a weighted sum of **all** encoder hidden states(h_j*'s*) aka annotations. *(New addition)*

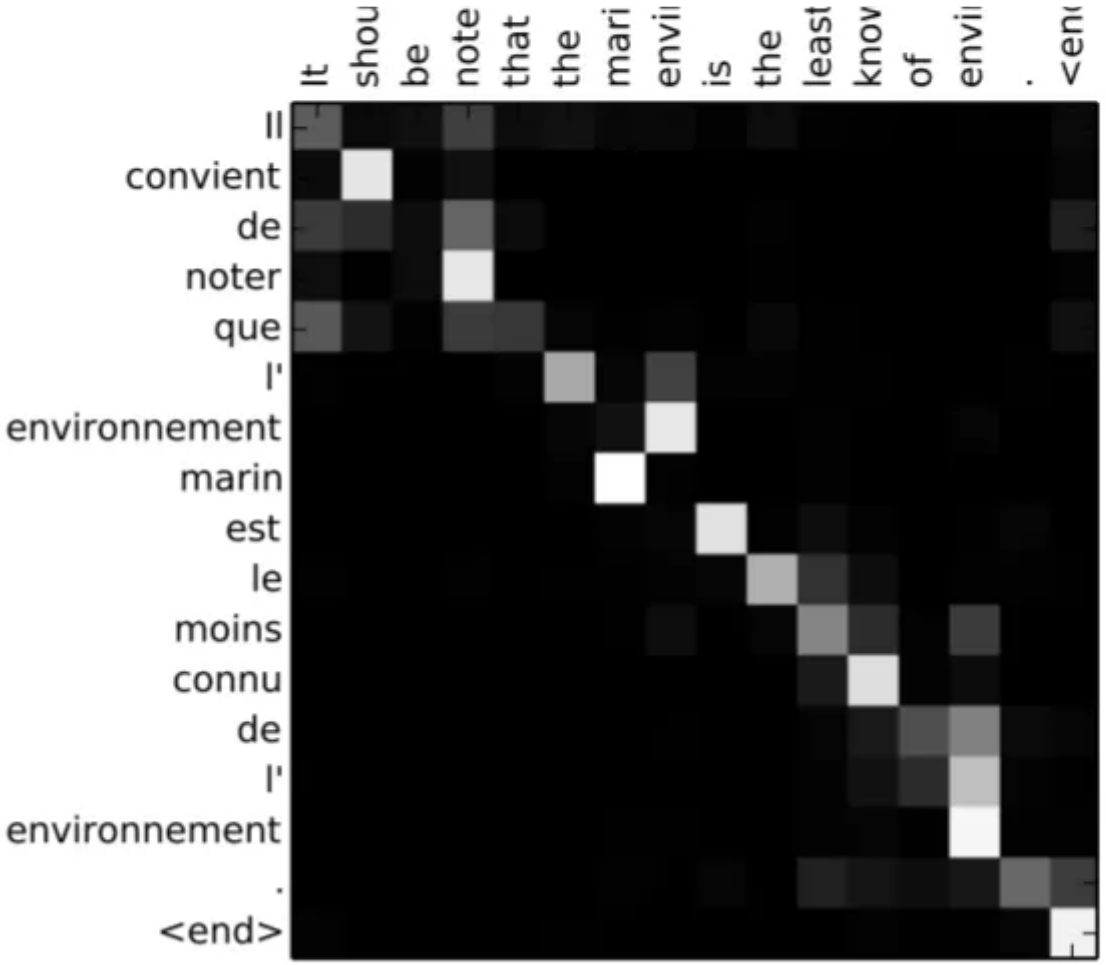


My take on the image in the original paper. I have not shown the intermediary state in this diagram to keep it simple. ht(the last encoder hidden state) will be the intermediary state and it will be fed, as in input in the first

step to the decoder model.

A

The global alignment weights are important because they tell us which annotations(s) to focus on for the next output. The weights will and should vary in each time steps of the decoder model. They are calculated by using a feed forward neural network.



"The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight αij of the annotation of the j-th source word for the i-th target word (see Eq. (6)), in grayscale (0: black, 1: white)" — The original paper.

A few observations worth noting:

- only for a few words at a time. No more than 3–4 words have high attention for a given output word.
- The attention doesn't linger for very long on a particular input word across time steps. That is, just because the weight was high in the previous step, doesn't imply it would be in the subsequent steps.
- Sometimes attention returns to an input word — look at the word "*que*" in output and how a part of its attention is on the first word "*It*".

The flow while predicting — during testing/validation of the model

The sequence of steps is as follows:

ENCODER MODEL:

- ***Step 1*:** Run the input sequence to get all the hidden states and calculate the intermediary state.

DECODER MODEL:

- *Step 3:* Calculate the context vector by multiplying the $\alpha_{k\ j}$ with h_j for j in range 0 to *t, where t*=steps in encoder model.

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

The formula for calculating context vector. For our step 3, i = k.

- *Step 4:* Take the previous hidden state of the decoder, H_{k-1},the context vector C_k, and the previous output Y_{k-1} to get the next hidden state of the decoder H_k. Predict Y_k from H_k.
- *Step 5:* Repeat 2, 3 and 4 till model predicts end token.

How to get the Global Alignment Weights

We know we use a feed forward neural network that outputs these global ali

im ıg

the next state H_k. This, in a way, allows the model to decide which parts of the input to attend to. The bigger the weight, the more attention it gets. So the next question is, what are its inputs, and how do we train it?

The Inputs

We input both decoder hidden state and the annotations in our neural network to predict a single value — e_{kj} as the authors of the paper liked to call it, the "**associated energy**"— signifying the importance of the annotations in next decoder step H_k. We repeat this process for all annotations. Once, we have associated energies corresponding to all annotations, we do a softmax to obtain the **global alignment weights** α_{kj}.

$$e_{ij} = f(H_{i-1}, h_j)$$

The neural network takes in the ith hidden state of the decoder and the jth hidden state if the encoder to predict each eij, the **associated energy**. For us i=k, as we are doing it for the kth time step in decoder model.

$\exp(e_{ij})$

Global Alignment Weights: Each weight is a Softmax on outputs(eij) from the neural network.

Step 2, mentioned before, can now be broken down as follows:

- ***Step 2.a:*** Calculate $e_{k\,j}$ where j = 0 to t, t = time steps in encoder model. The decoder hidden state will be the same for all j and will be H_{k-1}.

$$e_{kj} = f(H_{k-1}, h_j)$$

For j = 0 to t, i.e. for all hidden states in encoder.

- ***Step 2.b:*** Once we have all e_{kj}, do a Softmax to get $\alpha_{k\,j}$.

Training/Loss Calculation

The alignment model directly computes a soft alignment — considers all inputs, which allows the gradient of the loss function — calculated for final outputs of the entire sequence to sequence model — to be back-propagated

through. This gradient is used to train the alignment model as well as the whole

Let α_{ij} be a probability that the target word Y_i is translated from a source word X_j. Then, the *i*-th context vector C_i is the expected annotation over all the annotations with probabilities α_{ij}. When loss at each time step is back-propagated, we calculate the gradient for all three inputs — previous hidden state, the previous output and the context vector. *(If this is a little confusing for you, please read more about back-propagation here)* All the gradients flowing back in a time step shall be added together before we calculate the gradients for its inputs.

The gradient at each time step for C_k is used as the loss for the feed forward neural network we use to predict global alignment weights.

Impact on Performance

- The model with attention mechanism was able to outperform the conventional encoder–decoder model significantly, regardless of the sentence length and that it is much more robust to the length of a source sentence.

- It was also able to correctly align each target word with the relevant

- It enabled further development in the field by paving the road for Transformers and SOTA models like Google's BERT — which inspired RoBERTa by Facebook, AzureML-BERT by Microsoft and many more.

Conclusion

1. In our current approach with RNNs, the intermediary state act as a bottleneck for performance.
2. Attention can solve this problem by looking at each individual input state while predicting next output hidden state.
3. The mathematics for calculating attention.
4. Integration with RNNs based models.
5. Performance of RNN + attention models.

Follow Up Read:

Transformers	