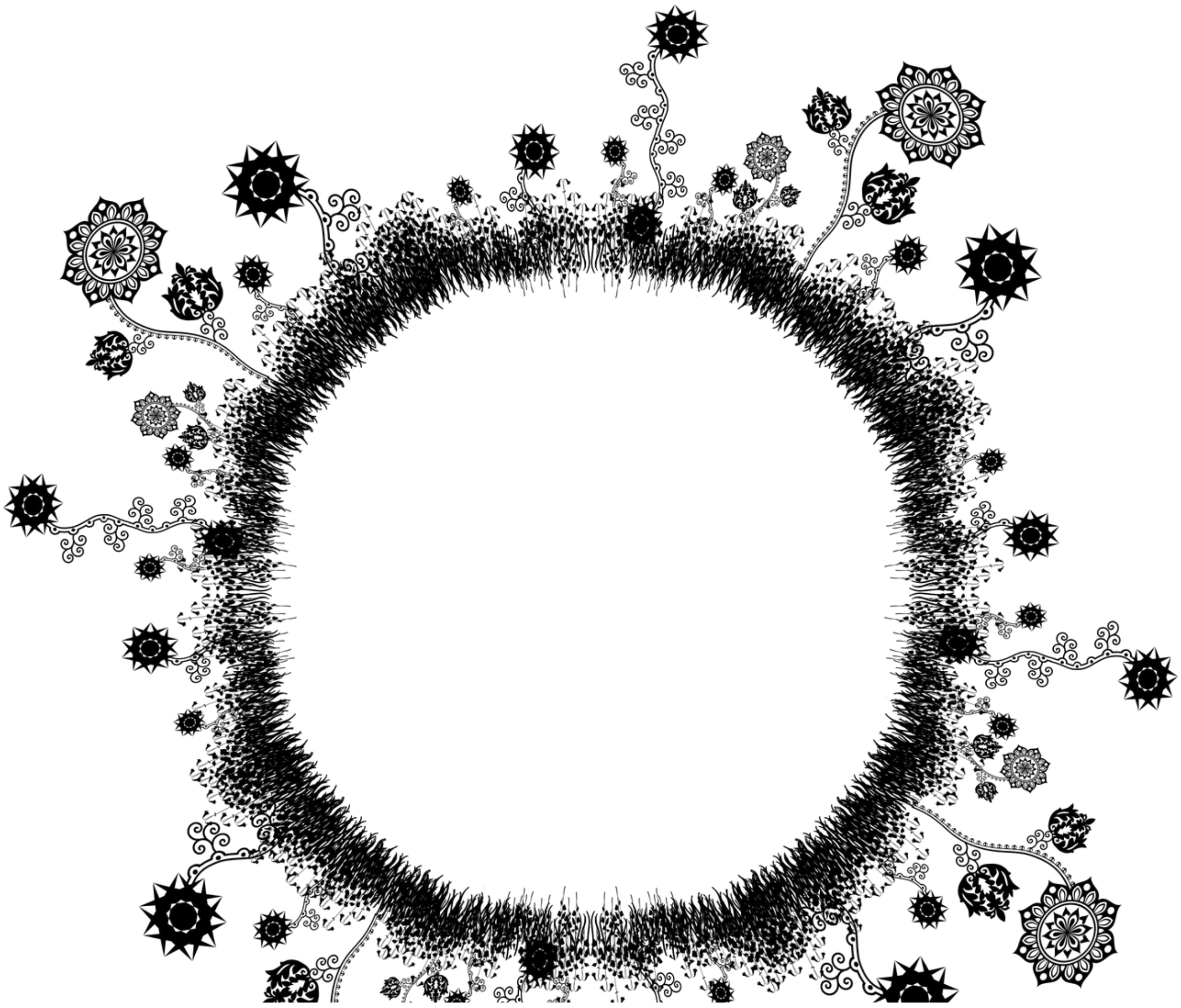
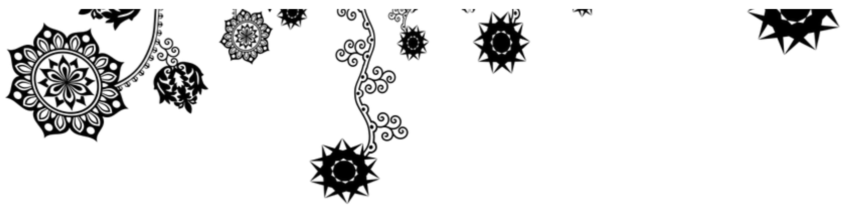
An introduction to Attention

The why and the what

Mahendran Venkatachalam Jun 29, 2019 · 5 min read ★





> *Our biggest source of confusion was the heptapods' "writing." It didn't appear to be writing at all; it looked more like a bunch of intricate graphic designs. The logograms weren't arranged in rows, or a spiral, or any linear fashion. Instead, Flapper or Raspberry would write a sentence by sticking together as many logograms as needed into a giant conglomeration.*

These lines ***from Ted Chiang's novella "Story of your life"*** perhaps give a good sense of what differentiates attention based architectures from the sequential nature of vanilla RNNs.

Let's take a quick look at vanilla RNNs and the encoder-decoder variation used in sequence to sequence tasks, understand what drawbacks these designs have and see how attention mechanisms address them.

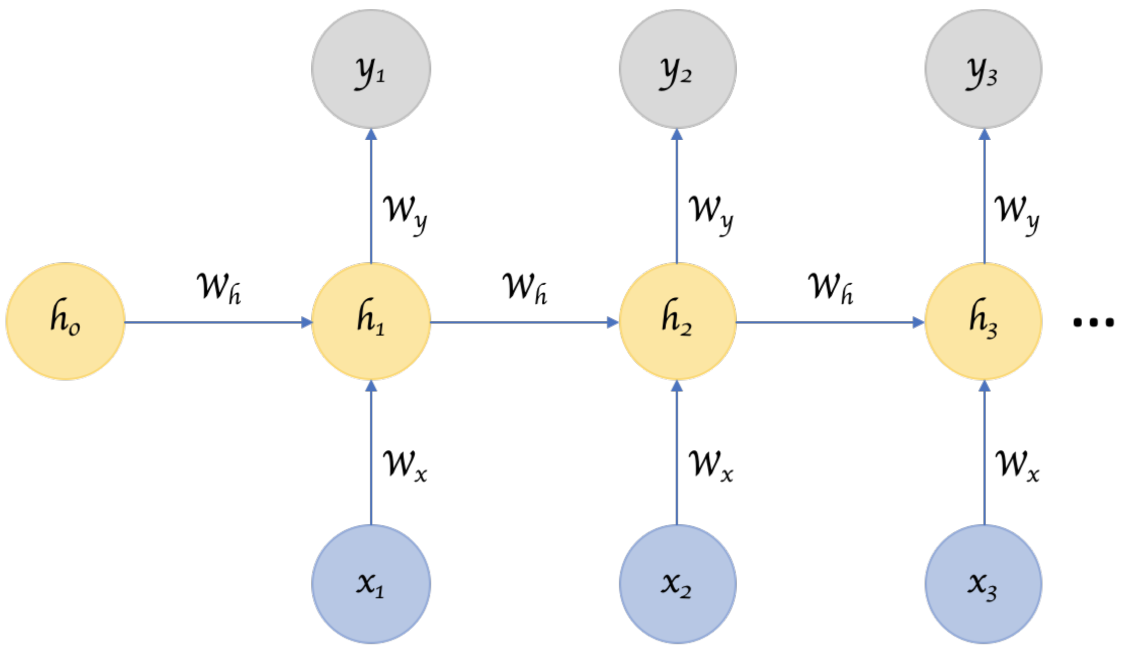


Fig 1: Vanilla RNN example

The basic premise of a vanilla RNN is to parse every item in an input series, one after the other, and keep updating it's "hidden state" vector every step of the way as shown in **Figure 1**. This hidden vector at the end of every step is understood to represent the context of all prior inputs. In other words, the last hidden state represents the context of the entire sequence.

In sequence to sequence translation tasks, this context-representing hidden-state-producing RNN is considered an Encoder and the final hidden state vector, referred to as "Context" in **Figure 2**, is fed into another sequence generating RNN called the Decoder.

But is this sequential nature of processing important or does it put us at a disadvantage? There are languages where word order doesn't strictly matter like Polish and Hungarian. Or even in English, where we can change word order depending on what we want to emphasize. Sometimes even in practical applications, say while processing a patient's history in diagnosis prediction models, the inter-event relationships are as, if not more, important than the actual event sequence itself.

Intuitively speaking, this strict order of processing is perhaps akin to flattening a 2-D image, i.e. converting from a matrix into a vector and using a vanilla feed-forward network to process it. It is much less efficient compared to CNN architectures that preserves the natural spatial relationship in the matrix representation (Of course, unlike in an image, we do not know if a correct order exists or what that order is. So that's that!).This strict sequential nature of processing is perhaps the ***first drawback***.

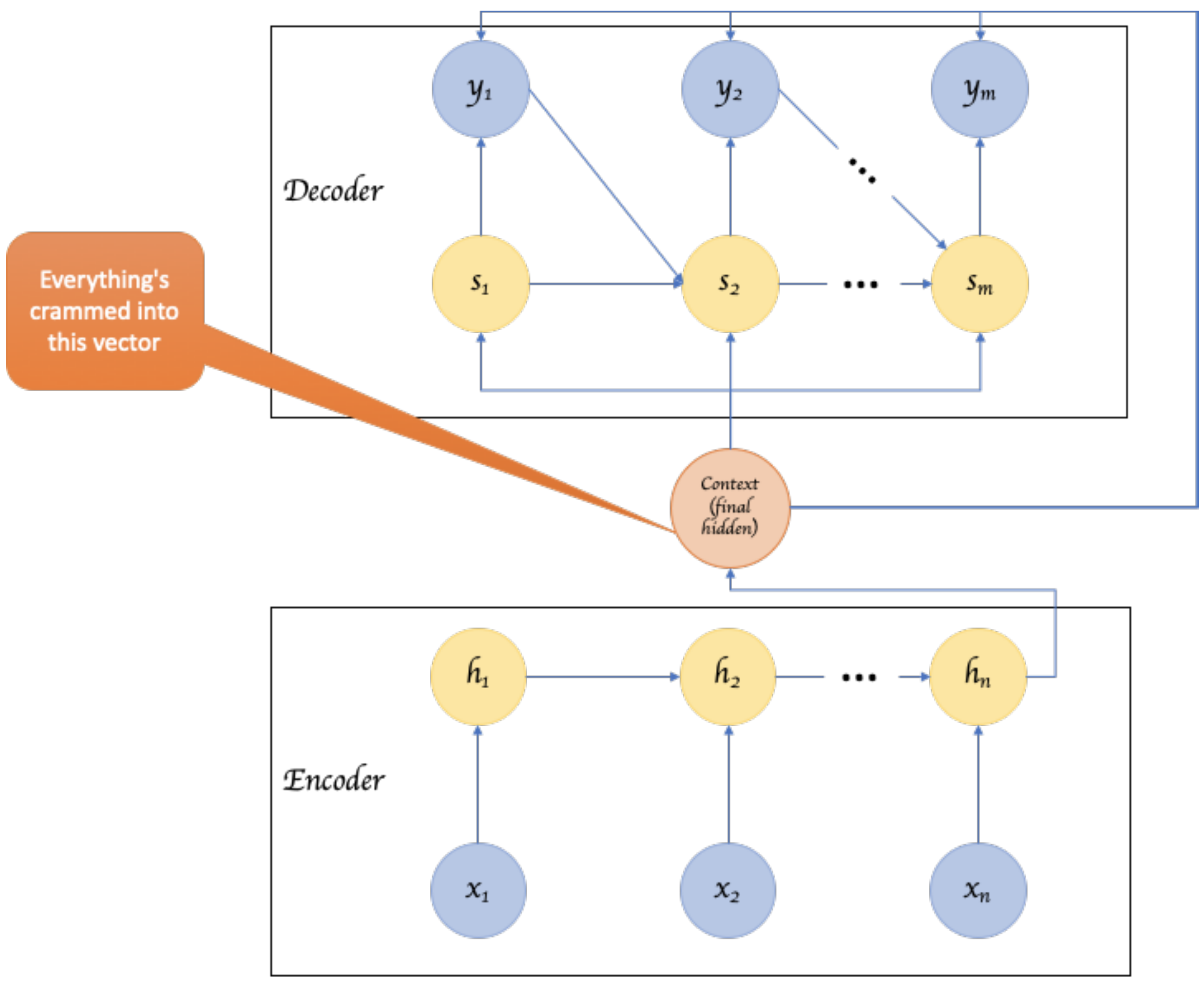


Fig 2: RNNs in Seq to Seq Encoder Decoder model

This is where LSTMs and GRUs helped in a big way by providing a way to carry only relevant information from one step to the next through various cell level innovations like forget gate, reset gate, update gate etc. Bidirectional RNNs provided a mechanism to look at not just the prior but also the subsequent inputs before generating an output at a time step. Such developments addressed the "strictly sequential" problem, but it didn't quite solve the next challenge.

The longer the input sequence length (i.e. sentence length in NLP) the more difficult it is for the hidden vector to capture the context (explanatory hypotheses suggested by Cho et al here, experimental demonstration of degrading performance can be found in the paper by Koehn and Knowles here). This drawback makes sense intuitively; the more updates are made to the same vector, the higher the chances are the earlier inputs and updates are lost (as demonstrated in Figure 3).

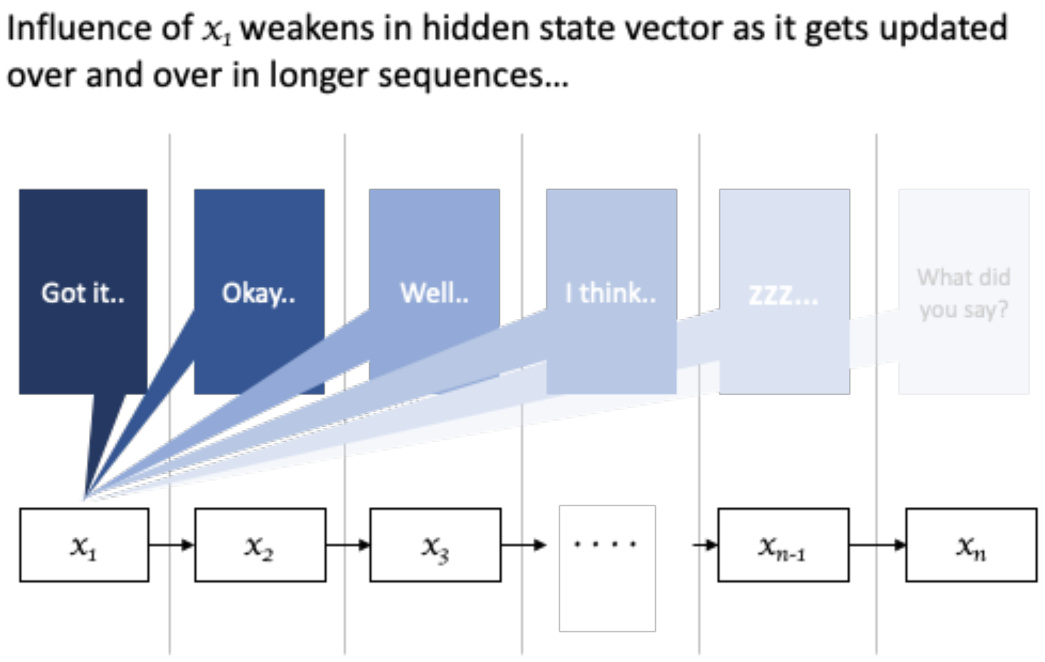
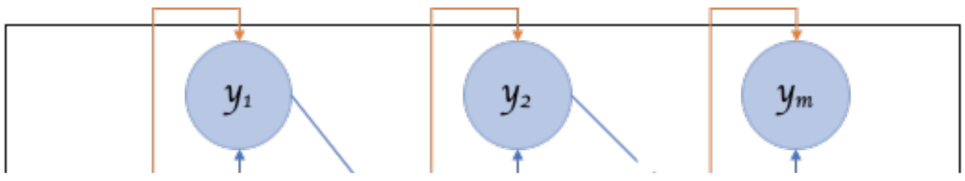

Influence of x_1 weakens in hidden state vector as it gets updated over and over in longer sequences...
Got it..
Okay..
Well..
I think..
zzz...
What did you say?
x_1
x_2
x_3
....
x_{n-1}
x_n

Fig 3: Context becomes weak with longer sentences

How could we solve this? Perhaps if we get rid of using just the last hidden state as a proxy for the entire sentence and instead build an architecture that consumes all hidden states, then we won't have to deal with the weakening context. Well, that is what "attention" mechanisms do. It was introduced in this paper by Bahdanau et al.



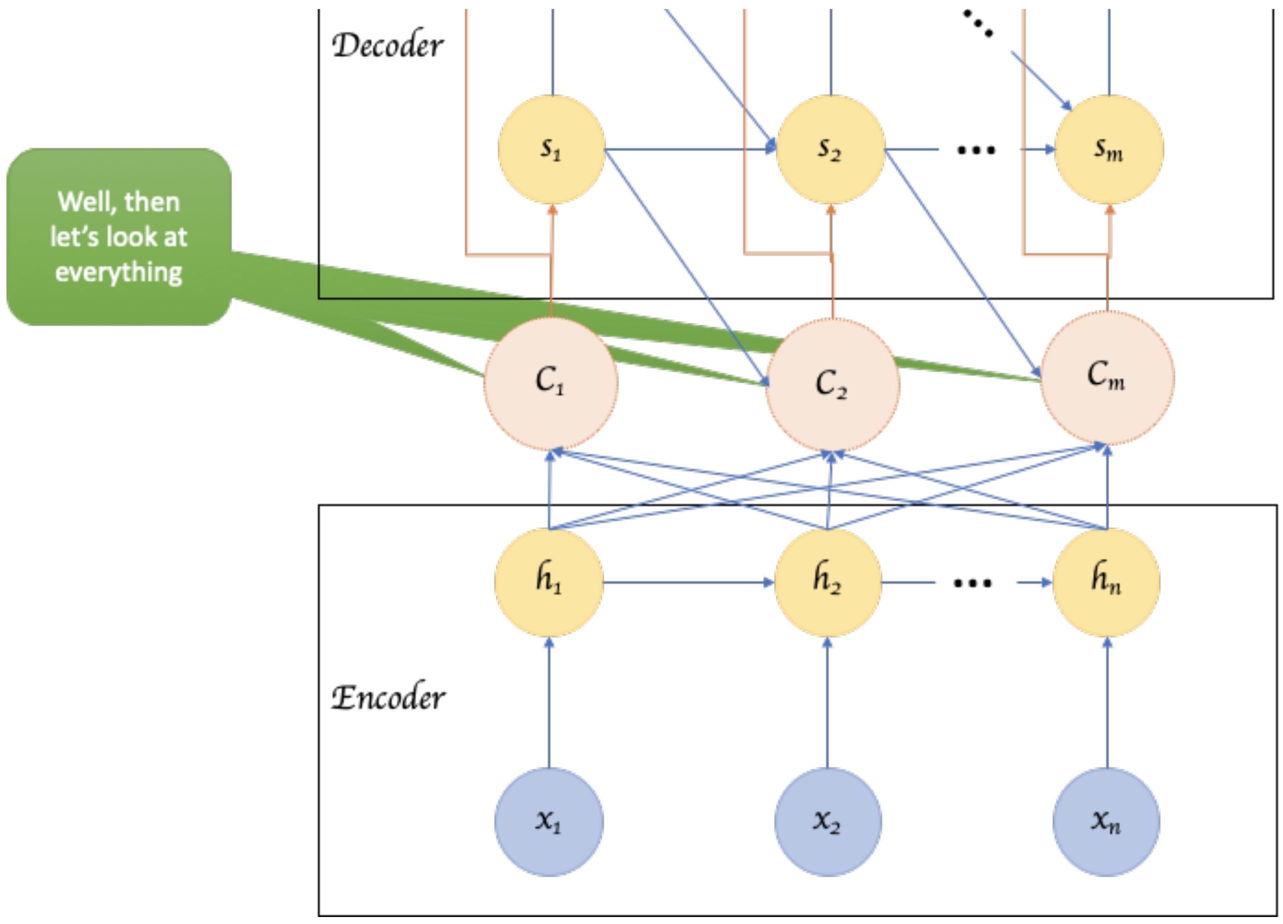
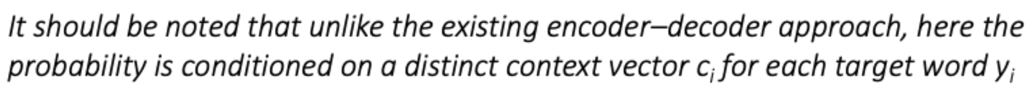


Fig 4: Using all hidden states, not just the last one

In the proposed model, each generated output word is not just a function of just the final hidden state but rather a function of **ALL** hidden states. And, it's not just a simple operation that combines all hidden state — if it was, then we are still giving the same context to every output step, so it has to be different! It is not a simple concatenation or dot product, but an "attention" operation that, for every decoder output step, produces a distinct vector representing all encoder hidden states but giving different weights to different encoder hidden state.

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

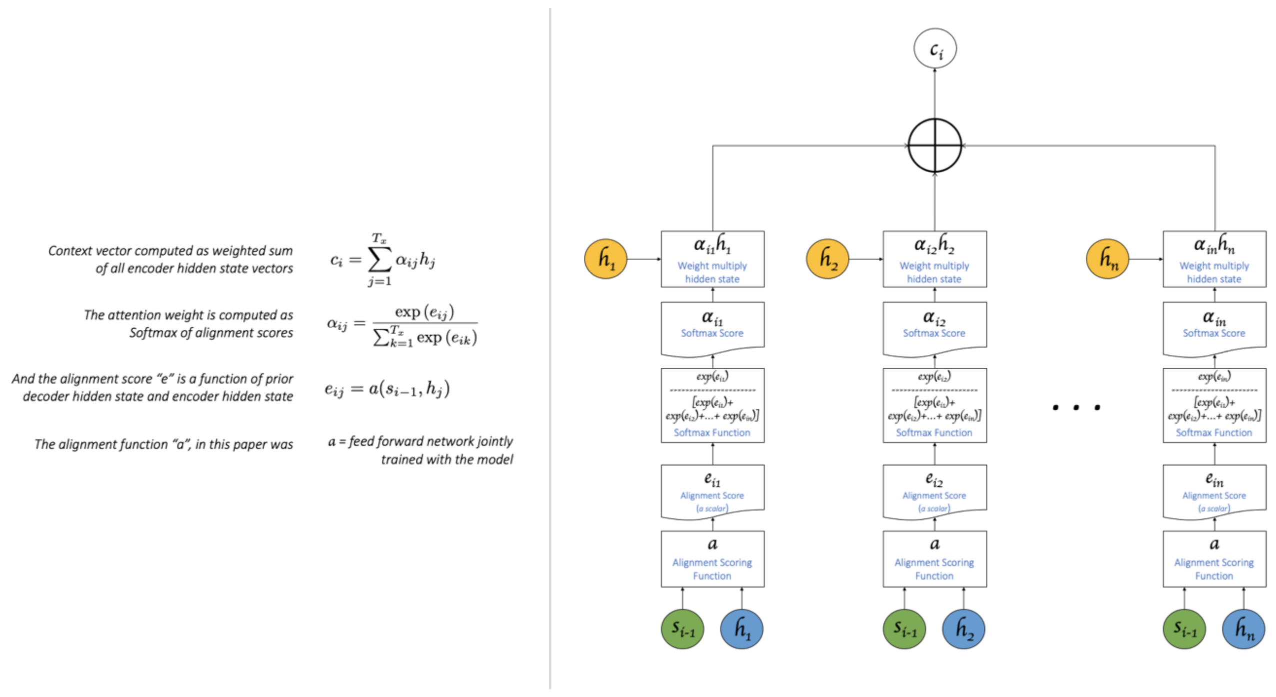
$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$



It should be noted that unlike the existing encoder–decoder approach, here the probability is conditioned on a distinct context vector c_i for each target word y_i

Fig 5: From paper by Bahdanau et al.

The distinct context vector for an output step is a sum-product of attention weights and all input hidden states. The attention weights for every single output will be different and therefore the sum of the weighted hidden vectors is distinct for each output step.



Context vector computed as weighted sum of all encoder hidden state vectors

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

The attention weight is computed as Softmax of alignment scores

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

And the alignment score "e" is a function of prior decoder hidden state and encoder hidden state

$$e_{ij} = a(s_{i-1}, h_j)$$

The alignment function "a", in this paper was *a = feed forward network jointly trained with the model*

c_i
h_1
$\alpha_{i1}h_1$ Weight multiply hidden state
α_{i1} Softmax Score
$exp(e_{i1})$ / $[exp(e_{i1})+ exp(e_{i2})+\dots+ exp(e_{in})]$ Softmax Function
e_{i1} Alignment Score (a scalar)
a Alignment Scoring Function
s_{i-1}
h_1
h_2
$\alpha_{i2}h_2$ Weight multiply hidden state
α_{i2} Softmax Score
$exp(e_{i2})$ / $[exp(e_{i1})+ exp(e_{i2})+\dots+ exp(e_{in})]$ Softmax Function
e_{i2} Alignment Score (a scalar)
a Alignment Scoring Function
s_{i-1}
h_2
. . .
h_n
$\alpha_{in}h_n$ Weight multiply hidden state
α_{in} Softmax Score
$exp(e_{in})$ / $[exp(e_{i1})+ exp(e_{i2})+\dots+ exp(e_{in})]$ Softmax Function
e_{in} Alignment Score (a scalar)
a Alignment Scoring Function
s_{i-1}
h_n

Fig 6: Based on paper by Bahdanau et al.

Keep in mind that the spirit of “attention” is more about the ability to attend to various inputs for every output step and is less about other aspects like alignment function used, nature of RNN involved etc. So, you might come across other variants to what is described above.

On that note, while this solution seems to have addressed the problem of single context vector, it has made the model really big. There are a lot of computations involved when you try to prepare a separate context vector for every output step.

In addition, there is yet ***another problem*** with computational complexity that wasn’t introduced by this solution, but existed even in the basic RNN. Given the sequential nature of the operations, if the input sequence is of length “n”, it requires “n” sequential operations to arrive at the final hidden state (i.e. calculate h1, h2 etc till hn). We cannot perform these operations in parallel as h1 is a prerequisite to calculate h2. This lack of parallelization within a sequence cannot be offset by adding more samples within a training batch either, as loading and optimizing weights for different samples increases memory needs which will limit the number of samples that can be used in a batch.

Solving some of these issues requires us to look at a few other variants of attention, and subsequently it will lead us into exploring the Transformer model. I intend to write about those in another post, hopefully sometime soon.

Machine Learning Attention Rnn Sequence To Sequence Deep Learning