# How to overcome the limitations of RNNs?

Thus, reiterating the problems in RNNS more succinctly:

**Encoding bottleneck:**

* In RNNs, sequential information was fed in and processed time step by time step and this imposed an “encoding bottleneck” – that is: in RNNs, we were trying to encode a lot of content – for example, a very large body of text with many words into a single output (considering the example of sentiment classification as shown below) that may be at the very large time step.

A picture containing screenshot, diagram, design

Description automatically generated

**Figure: Sentiment classification - encoding a large text into a single output- using RNNs.**

Thus, it was not ensured that all the information was properly maintained and encoded and learnt by the network. As discussed, a lot of information was lost because of the problem relating to vanishing and exploding gradients.

**RNNs were slow to train:**

* Another limitation being that because the of time step by time step processing, RNNs could be quite slow to train and there was not really an easy way to parallelize the computation. Thus, the capacity of RNNs as well as LSTMs in terms of the “memory” was not long and they were unable to handle long sequences of data.

Because of the limitations of RNN, there was a lot of attention to move forward beyond the notion of the step-by-step recurrent processing to build even more powerful architectures to process sequential data. The idea was to process information continuously as continuous stream – to parallelize computation to speed up processing as well as to establish long memory to build rich understanding of sequential data, rich understanding of context.

**How do we overcome the limitations of RNNs?**

As pointed in the preceding paragraphs, RNNs use time step by time step processing of data – the notion of recurrence – to maintain the order information. We discussed the key limitations and reiterated those limitations in the section above.

Thus, what we want now is, that, rather than sequential time step by time step processing of data, we would like to process the information continuously as a continuous stream of information. Rather than being slow, we’d like to parallelize the computation to speed up the processing and importantly establish long memory that can build rich understanding of context, understanding of sequential data.

**Can we use feed forward networks?**

Thinking of resolving the problems of RNNs, a naïve approach will be to squash all the data/ all the time steps together to create a vector that effectively concatenated the time steps together, feed all the data to a feed forward network which through its densely connected layers does the learning and gives you the outputs.

A picture containing text, screenshot, line, rectangle

Description automatically generated

**Figure: Feed forward network for sequence modelling**

A picture containing pattern, art

Description automatically generated

With the above we’ve eliminated the need for recurrence, but we still have the following issues:

* **Scalability** – The network is not scalable because the dense feed forward will be immensely large defined by several connections.
* **No order** – Critically, we have lost the order information that was necessary for sequential processing of data.
* **No notion of memory** – There is no temporal dependence – we’re stuck in the ability trying to establish long term memory.

**How do we solve the problem?**

We need to extract the information from the input data more intelligently – the key idea being to identify and “attend” to what is important in a sequential data, and this is the notion of attention mechanism which is an extremely powerful concept in modern deep learning and natural language processing.

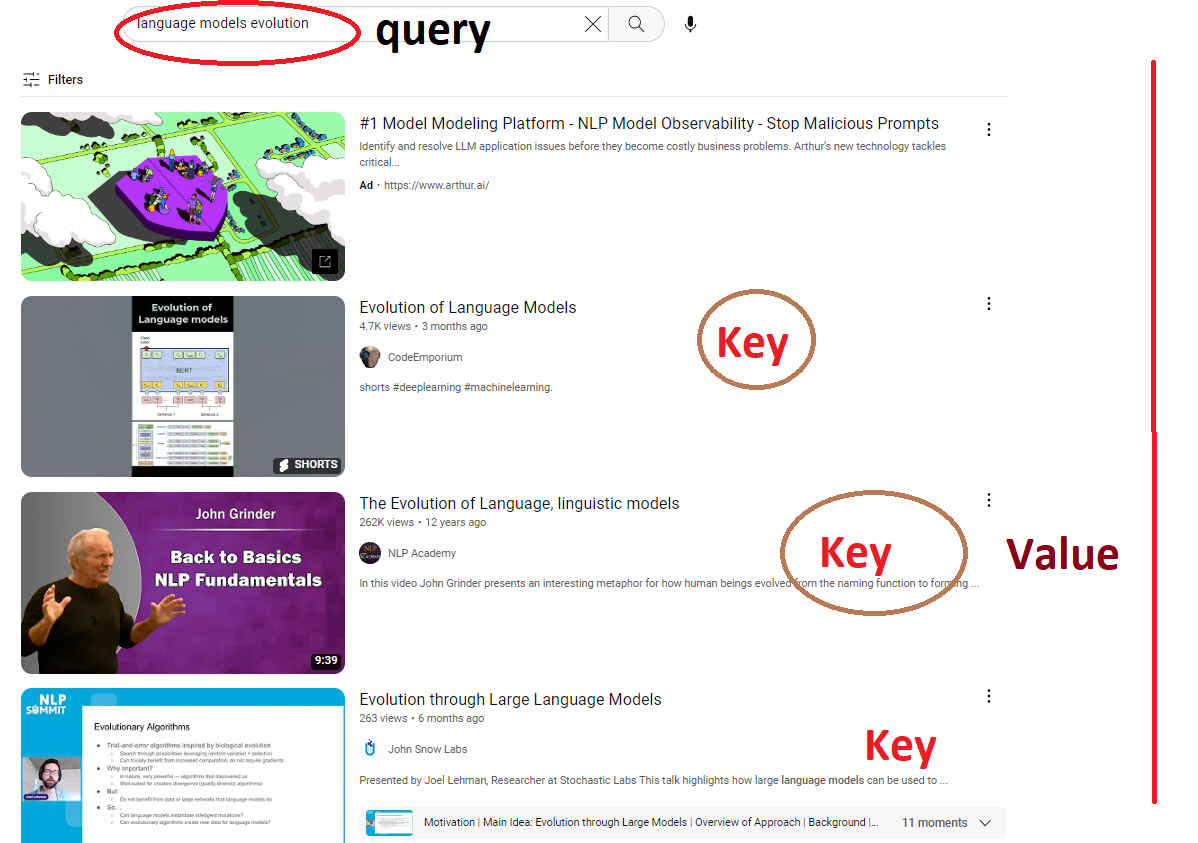
# An Intuitive Understanding of Attention

Attention is the fundamental mechanism of the Transformer architecture. As the name suggests, the goal of attention is to allow the model to focus on important parts of the input. This makes sense from a human perspective: when we look at an input (e.g., an image or a text), some parts are more important for our understanding than others.

We can relate certain parts of the input to each other and understand long-range context. These are all essential for our understanding and attention mechanisms allow Transformer models to learn in a similar way.

The intuition of attention can be best felt through search.

* For example, in order to search about “the evolution of language models” through the database of You-Tube videos, you will type in a query “evolution of language models”.
* In every video, that appears, you will have some key information related to the title.
* We need to find the overlap between the query and each of these titles – the “keys” in the database.
* We need to compute a similarity / relevance metric to find the relevance between query and each key that appears.
* Mathematically, the similarity metric we compute is a dot product or cosine similarity.
* We extract the relevant information wat we want to pay attention to. We get a Value. We extract the relevant video.



And this intuition of giving the query, trying to find the similarity and trying to extract the relevant information forms the basis of Attention which is the heart of the Transformers and heart of ChatGPT.

Now, with this mechanism and Intuition of Attention established, we are much ahead of Recurrent Neural Networks:

* We do not need Sequential processing of data.
* We know which part of the inputs we need to pay attention to and capture the contextual meaning.

Let’s go back to our text/language example, with this sentence the goal is to identify features in the input that are relevant to the semantic meaning of the sentence. [*This is important because when we’re dealing language and making a language model to predict the next word I ought to know the semantic meaning of the sentence*]

We perform the following operations:

1. Encode the positional information.
2. Extract query, key and value – exactly similar to search operation
3. Compute attention weights
4. Extract features with high attention

It should be noted that the query, key and values are arrays of numbers, and you can think of them as vectors in the space. The query value is a vector and key value is another vector. Mathematically, we can compare these 2 vectors by taking the dot product and scaling it – these captures how similar the vectors are.

A picture containing font, line, screenshot, text

Description automatically generated

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

A picture containing screenshot, text, diagram, square

Description automatically generated

**About Positional Embedding**

Let’s lift the mystery behind positional embedding. Transformers do not model the order of the input anywhere; it is important to encode the order of the input. This happened in positional encoding.

Transformer has to know what comes after the other word – before or after – and not do any permutations. This is where positional embeddings come in. They are kind of a “hint” for the transformer about the whereabouts of the word within the sequence.

We add the positional embedding to the input embedding – making it move a certain distance from where it currently is.

A picture containing line, diagram, screenshot, design

Description automatically generated

Positional embedding are the identifiers that are added to the original word embedding for the Transformers to know the order of the sequence.

The positional embedding should fulfil certain requirements:

1. Every position must have the same identifier irrespective of the sequence length.
2. It should be noted that the positional embeddings push the original input embedding, care should be taken to see that the input embedding do not get drifted away too far away so that the semantic meaning of the embedding is lost.
3. We could think of numbering the positional embedding as 1, 2, 3 but that violates the above rule!
4. We thus need to keep the value of the positional embedding bounded.
5. To keep the values of the input embedding bounded, we choose functions of sine and cosine, sines and cosines are bounded between -1 and +1 range from – infinity to + infinity
6. We take both sines and cosines because: had we taken only sines we might have got some repeated position values which is not permitted by 1)
7. Hence, we take both sines and cosines.

**Reinforcement Learning and ChatGPT:**

ChatGPT is a GPT model that is fine tuned to respond to a user’s request given a prompt. IT is then further fine-tuned by Reinforcement Learning. Reinforcement Learning most simplistically can be defined as a method of achieving some goal via rewards.

Let us simplistically understand reinforcement learning by the diagram below. We’ll also see how these concepts extend to ChatGPT.

**Agent** – As shown in the figure. The goal here is to make the agent reach the end state. In order for the agent to make certain moves we entice it with some rewards. The reward is a scalar value. When the goal is reached, we provide a higher reward and otherwise a lesser reward. We penalize the agent with lower rewards because we want it to achieve its goal as quickly as possible.

**State:** The state is the representation of the current step – wherever the agent is during its journey

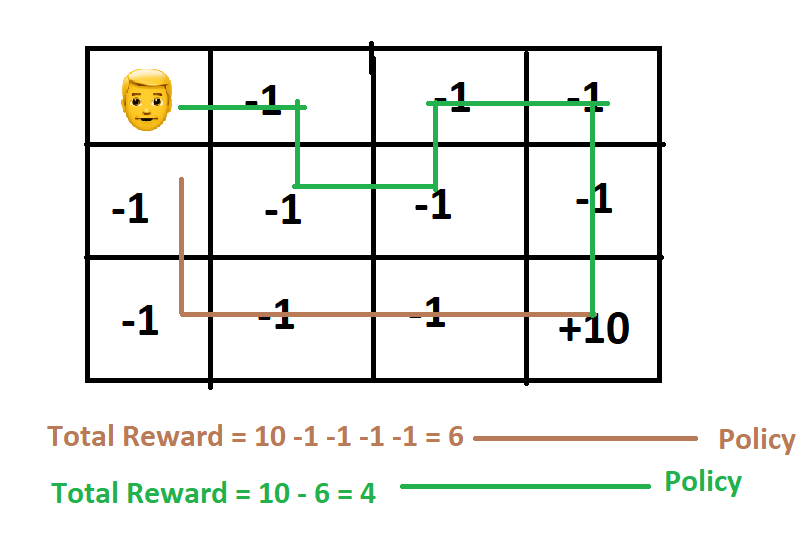
**Action:** Action is the action that is taken by the agent within the boundaries we have to get to its final goal

**Policy** is the sequence of action the agent takes in order to achieve its goal.

In relation to ChatGPT:

Agent is the model itself, the Reward depends upon the type of response ChatGPT generates, if the response is good, the Reward is high, if the response is poor reward is low.

Every single action taken by the agent is the time step. The time step occurs as word token are generated.



**Transformer Architecture:**

1. Transformer Architecture was introduced in 2017 which was a sequence-to-sequence architecture. IT would take as an input some sequence and output a sequence.
2. In the field of NLP this can be super useful. Because sentences are a sequence of words.
3. So, we started to use this for NLP problems like translation.
4. Transformer architecture has 2 parts.
5. It has Encoder and Decoder
6. Encoder is going to take all inputs simultaneously.
7. You will get some vectors for each of these words.
8. These are context aware.
9. These 4 vectors passed simultaneously to the Decoder.
10. Translation
11. My name is Ajay.
12. This overall architecture has some understanding of language.
13. The Encoder and Decoder part also have some understanding of language.
14. So we pick them and stack separately
15. Encoder – BERT
16. Deocoder – GPT
17. Focus more on GPT

Amini:

Let’s see how Reinforcement Learning fits into this whole paradigm of different topics. Two different types of Learnings

Supervised Learning – predict y given x

Unsupervised Learning: No notion of labels

Trying to uncover – underlying structure, hidden features

Example: Build a model that understands features of an apple, red aple is similar to black and white sketch of an apple, cluster them together

Reinforcement Learning: Data – state – action pairs

State – observations

Action – is the behaviour that the agent takes in those particular states

Agent: Build rewards