Why Natural Language Processing is Such a Complex Task

First we will try to thoroughly explain the issue we're addressing before we explore the proposed solutions. It's important to recognize the complexity of a problem to truly understand it. When it comes to tasks involving data analysis, this complexity can present itself in various ways. We have factors like the number of dimensions, the size of the input, the presence of noise, outliers, and even intricate patterns lurking within the data. Speaking specifically about natural language, things get even more intricate. It encompasses both short-term and long-term dependencies between different elements, includes informal language like slang, deals with words or phrases that might have multiple meanings (like sarcasm), and navigates the realm of both implicit and explicit meaning. Additionally, language follows rules of syntax and grammar, meaning that the interpretation of a word can depend on its context, and continuity is not always guaranteed. Attempts to create structured solutions for language analysis, like the rule-based algorithm Cyc, unfortunately fell short due to these intricacies.

Let’s try to exemplify the complexity of language modelling tasks, on the challenging task of translation. To show you this, let's say we wish to translate the English phrase "The dog sleeps" into French: "Le chien dort". It seems we can translate every word in the sentence apart from the others. The only guideline we need to follow is that the English definite article "the" translates into the French "Le" before the word it modifies. Thus, we might believe that keeping a dictionary that has each English word's French counterpart, along with a couple of easy grammatical rules, should be enough for accurate translation.

Think about the English phrase "I am looking forward to it". If we try to translate it word by word using our handy dictionary and rules, we'd get "Je suis regardant en avant à cela". But the right way to say it would be: “Je l'attends avec impatience”. This shows a key challenge with this method: we're translating each word without thinking about how it fits in the whole sentence or looking out for those quirky idiomatic expressions. So when we're translating, it's super important to see how all the words in a sentence work together, and to watch out for those unique idiomatic expressions that don't translate directly.

How about we give our dictionary-based solution a little boost? We could store not just individual words and their translations, but also common word pairs. This might make things a bit more accurate, but it's kind of like putting a band-aid on a bigger problem. Before we know it, we'll be adding phrases of three, four, or even more words to our dictionary. But hey, that's the tricky part of rule-based algorithms: they can be really tough to use when you're trying to get all the complexities and little quirks of language just right!

The RNN architecture:

The Recurrent Neural Network (RNN), which dates back to the late 1980s, was the first architecture that sought to model the relationships between input words. The fundamental principle of RNNs is to analyse the current input in conjunction with information extracted from other words in the sentence. In processing the current word of the sentence, this model produces two distinct outputs. The first output corresponds to the network's result tailored to the particular task being performed, such as translation, text summarization, or text generation. The second output represents the context accumulated thus far, encapsulating all the information the network has processed, albeit in a lossy form.

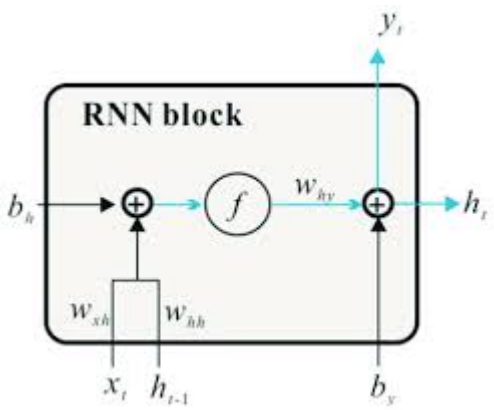

Figure 1 - RNN block

The Recurrent Neural Network (RNN) is essentially engineered to deal with sequential data, a characteristic intrinsic to language. In its most basic structure, an RNN consists of a sequence of computational units, often referred to as 'blocks' for simplicity. Each block functions at a particular time step t, considering both the current input at that point in the sequence (whether a sentence or a piece of text) and the previous context from the (t−1)-th time step. The block then produces an output that corresponds to these inputs.

However, this design brings up a natural problem because sentences can have different lengths. Since sentences might have various numbers of words, keeping an endless chain of blocks, with each block corresponding to a specific word in the sentence, is not practical. Also, this kind of structure would lead to an uneven training process. The earlier blocks would get much more training than the later ones. This happens because sentences that are shorter than the maximum number of blocks have to be filled with zeros, leaving the later blocks with less opportunity to learn.

To solve these problems, a repeated operation of the network is suggested. This approach uses just one block, and as a result, a common set of weights. During each repetition, the network takes in a new input and the context obtained from the output of the previous repetition.

The operation unfolds over time, as shown in Figure 2. In simpler terms, the network gets an input X_i and a previous internal state, H_{i-1} and then produces an output Y_i. This new internal state, Hi, is used again in the next step. This loop helps handle sentences of various lengths and ensures even learning throughout the sentence.

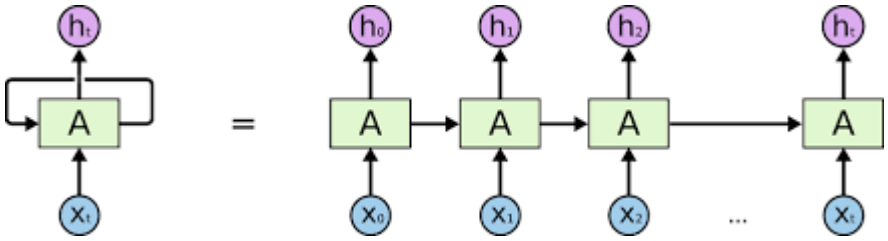

Figure 2 - RNN network (left) and unfolded RNN network (right)

Word embedding:

Using a Recurrent Neural Network (RNN) for language analysis starts with turning words or subwords into a mathematical form the model can work with. This is done through a method called word or subword embedding. In this step, each word is converted into a k-dimensional vector (or several vectors in case of a subword embedding). This creates a vector space where similar words are close to each other, while words with different meanings are far apart.

There are various methods for handling this encoding task. The most common method uses a neural network, like the word2vec or GLoVE algorithms. However, these schemes face a key challenge: words that have multiple meanings are represented the same way. This similarity complicates the learning process and hinders accurate understanding of a word's meaning.

The Transformer architecture innovatively integrated word embedding as a fundamental component, thereby overcoming the aforementioned limitations. This unique setup allows the model to generate word embeddings influenced by both nearby and distant words within the given context. Essentially, this capability enables the model to acquire contextualised embeddings, elevating its understanding of word semantics beyond mere proximity to adjacent words. This significant advancement

notably amplifies the model's semantic comprehension, thereby improving its performance in tasks related to language analysis.

Why did RNN fail to analyze natural language?

Upon initial examination, Recurrent Neural Networks (RNNs) appear to present a promising solution for modeling the intricate dependencies between different parts of input inherent in the problem at hand. However, their actual performance falls short of expectations for a multitude of reasons.

The first issue stems from the network's sequential architecture, which often results in the 'vanishing gradient' problem. In language parsing tasks, the output for an input sequence depends on previous terms in the sequence. Therefore, the network's weights are updated only after processing the whole sequence. In such iterative networks, calculating the gradient for these weights also involves a sequential process, but in the reverse order.

To measure how much the weights affect changes in the error, we have to multiply the partial derivatives of each internal state i with respect to its preceding state i−1 in a sequential manner. Because the same set of weights is used for this calculation, the gradient can either shrink or grow exponentially over the course of training. This behaviour can be illustrated by looking at the eigenvalues of the weight matrix. If the eigenvalues are smaller than one, the gradient diminishes with each weight update; if they are larger than one, the gradient grows exponentially due to the weight matrix amplifying it with each iteration. This phenomenon is commonly referred to as the vanishing/exploding gradient problem and has been specifically addressed in the context of Recurrent Neural Networks (RNNs).

The second issue stems from the interplay between the dimensions of the internal states, their ability to encapsulate information, and the sequence length that the network is required to analyze. The internal state effectively "compresses" all information up to the current moment. When the sequence surpasses a certain length, it introduces long-range dependencies among its elements. This makes the internal state a bottleneck in the network's operations, causing it to "forget" details that are distant from the current input.

The last challenge becomes evident during the network's deployment after training, commonly referred to as the inference stage. Due to the sequential nature of input processing, the network fails to fully utilize its own computational capabilities, as well as those of the machine it's running on. This results in slow inference times, or high

latency, particularly across various Natural Language Processing (NLP) tasks. This significant issue is addressed by the Transformer architecture, which we will explore in greater detail later on.

Key Points for Post Summary:

1. Traditional rule-based algorithms struggle in NLP tasks because they are unable to effectively model the complex relationships found in natural language. The alternative is to use networks that are capable of learning these relationships.

2. This network operates as a single processing unit that not only receives new input but also its previous internal state, which it uses to learn a compressed representation of all received inputs up to the current moment.

3. However, the network faces two major challenges: first, the issue of vanishing gradients during training, and second, the inability to capture long-range dependencies in the data.

4. Furthermore, when the network is deployed for inference, its performance is hampered by the sequential nature of data processing. This leads to inefficient use of computational resources and results in slow updates of the internal state.