Recurrent and Recursive Nets

Recurrent Neural Networks

RNNs are a class of neural networks that can represent temporal sequences, which makes them useful for NLP tasks because linguistic data such as sentences and paragraphs have sequential nature.

2.1. Definition

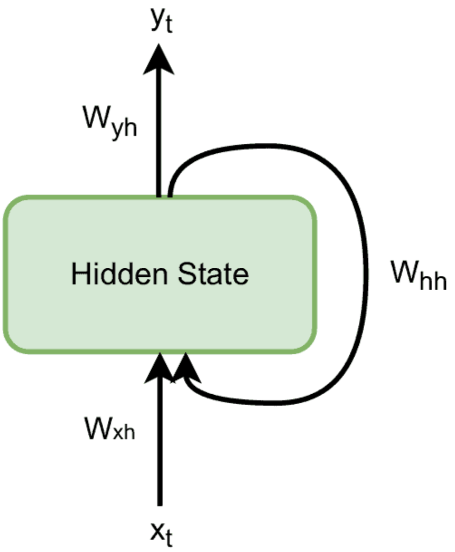
Let's imagine a sequence of an arbitrary length.

An RNN processes the sequence one element at a time, in the so-called time steps. At step , it produces the output and has the hidden state , which serves as the representation of the subsequence .

The RNN calculates the hidden state by combining and . It multiplies them with matrices and and applies the transformation to the sum:

Then, the output is the product of the matrix and :

Since the matrices are the same across the steps, the network has a simple diagram:



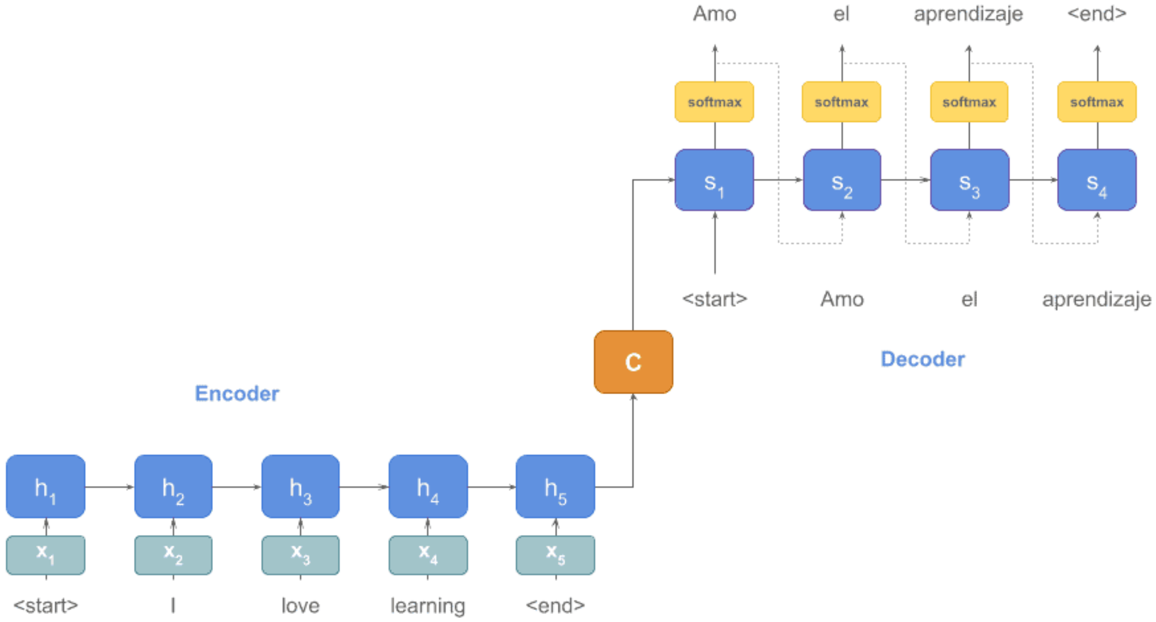
2.2. Training

Training the network consists of learning the transformation matrices. A widely used algorithm that does that well is the Gradient Descent algorithm.

Usually, we initialize the weights and the first hidden state randomly. For the function , there are several alternatives. For example, we can go with the function:

2.3. Example: Machine Translation

As an example, we can take a look at an encoder-decoder model for machine translation. In this scenario, the encoder receives a sentence in the original language, and the decoder produces the translation in the target language. Let's imagine that our original language is English and the target language is Spanish:

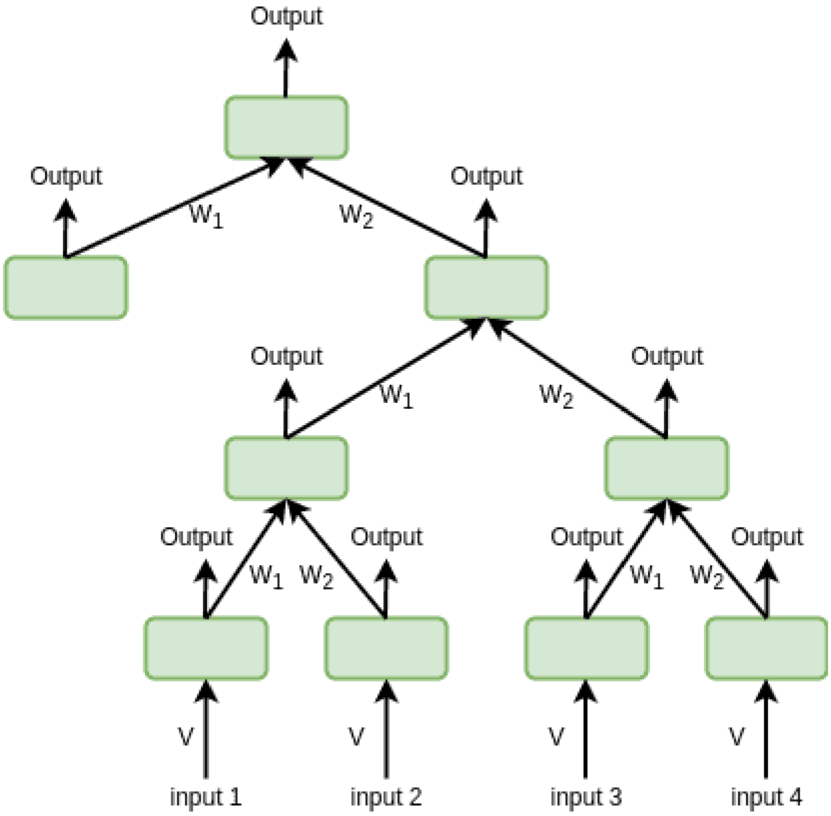


Recursive Neural Networks (RvNNs)

RvNNs generalize RNNs. Because of their tree structure, they can learn the hierarchical models as opposed to RNNs that can handle only sequential data. The number of children for each node in the tree is fixed so that it can perform recursive operations and use the same weights across the steps.

3.1. Definition

The tree structure of RvNNs means that to combine the child nodes and produce their parents, each child-parent connection has a weight matrix. Similar children share the same weights. In other words, considering a binary tree, all the right children share one weight matrix and all the left children share another weight matrix. In addition, we need an initial weight matrix ( ) to calculate the hidden state for each raw input:



Therefore, the number of weight matrices is equal to the number of children that a node can have. We can calculate the parent node's representation by summing the products of the weight matrices ( ) and the children's representations ( ) and applying the transformation :

where is the number of children.

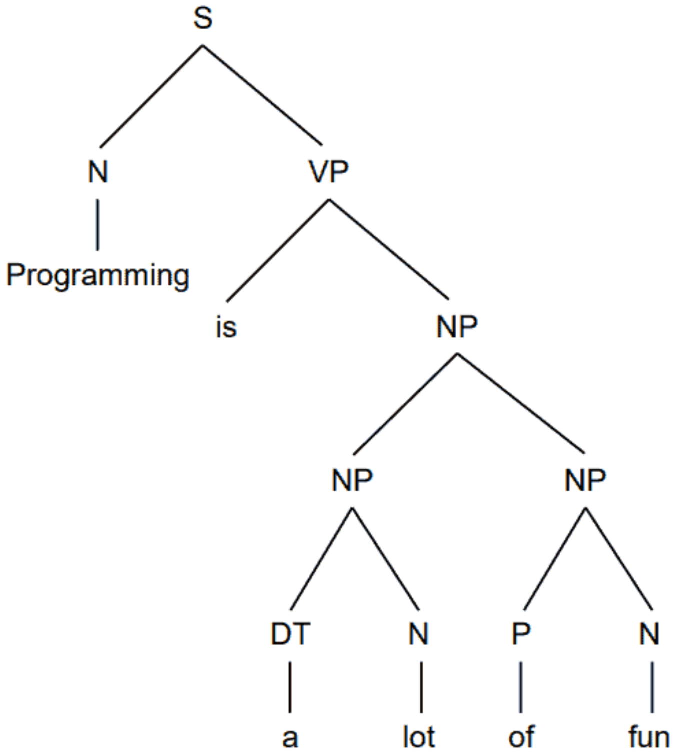
3.2. Training

Training RvNNs is similar to RNNs with the same optimization algorithm. There's a weight matrix for each child that the model needs to learn ( ). These weight matrices are shared across different recursions for the subsequent children at the same position.

3.3. Example: Syntactic Parsing

One of the main applications of RvNNs in NLP is the syntactic parsing of natural language sentences.

When parsing a sentence, we want to identify its smaller components such as noun or verb phrases, and organize them in a syntactic hierarchy:

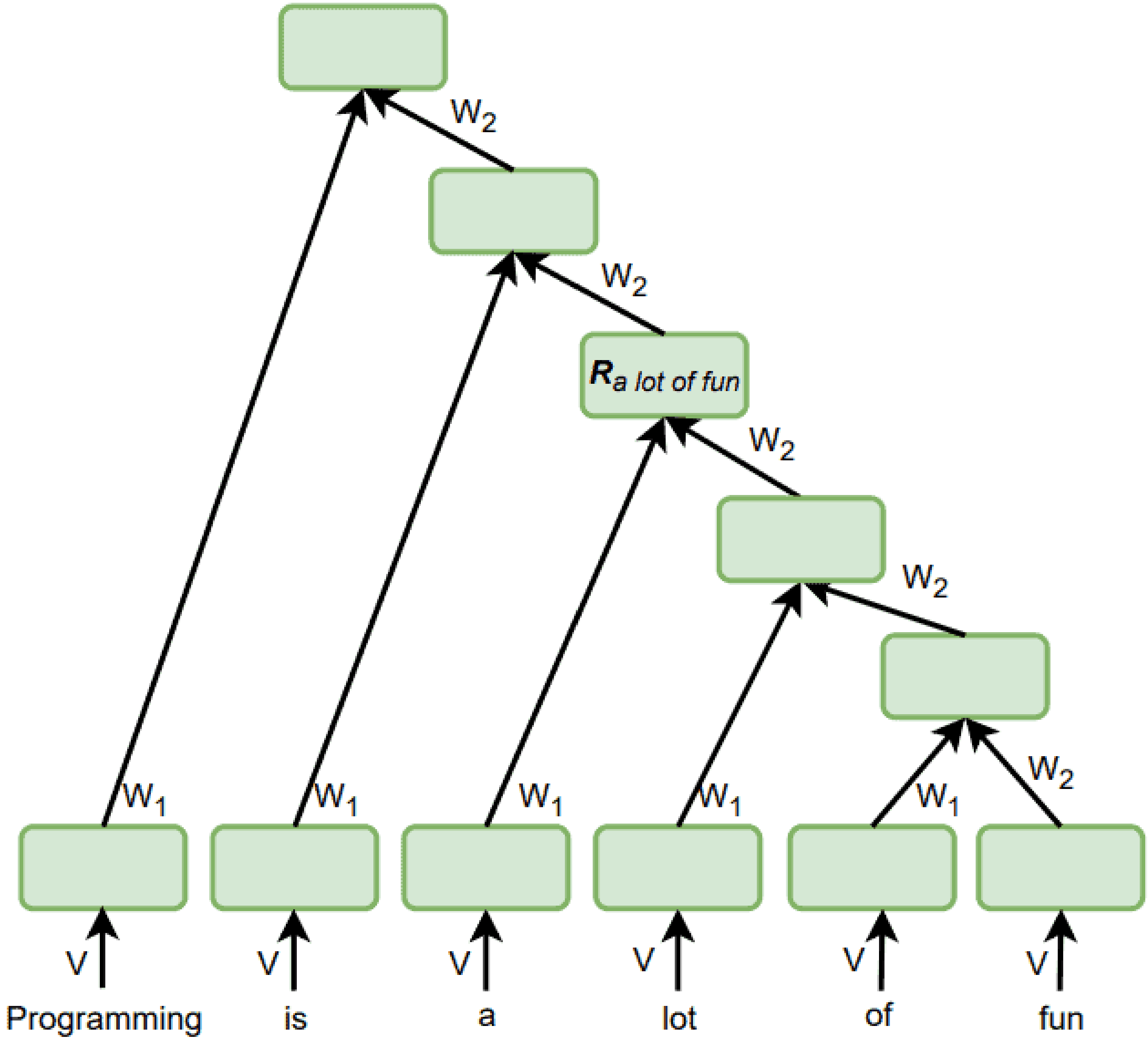


Since RNNs consider only sequential relations, they're less suitable for handling hierarchical data in contrast compared to RvNNs. Let's say that want to capture the representation of the phrase a lot of fun in this sentence:

Programming is a lot of fun.

What representation would an RNN give to this phrase? Since each state depends on the preceding words' representation, we can't represent a subsequence that doesn't start at the beginning of the sentence. So, when our RNN processes the word fun, the hidden state at that time step will represent the entire sentence.

In contrast, the hierarchical architecture of RvNNs can store the representation of the exact phrase:



It is in the hidden state of the node .