CIS 522 - University Of Pennsylvania, Spring 2020

Lecture 11: Introduction to Natural Language Processing

20 February 2020

Lecturer: Konrad Kording *Scribe: Chetan, Jinaqiao, Nidhi*

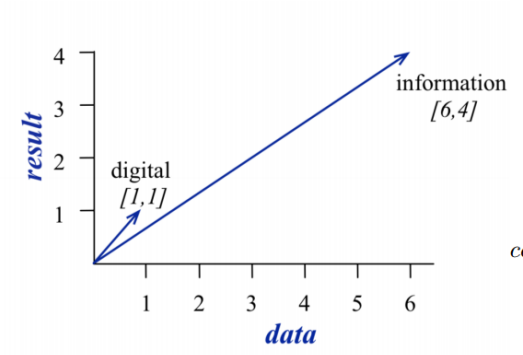
1 Word Vectors

The first step to discuss natural language is to represent it in computers. We first talk about word vectors, or sometimes we call it, word embeddings. In very simplistic terms, Word embeddings are the texts converted into numbers and there may be different numerical representations of the same text.

Before the deep learning era, we tended to treat each word as an individual symbol, totally independent of others;E.g. tw words: "hotel, motel", in one-hot encoding

$$Hotel = [00000000000000010000] \tag{1}$$
$$Motel = [00000000100000000000] \tag{2}$$
$$\tag{3}$$

The different types of word embeddings can be broadly classified into two categories- 1) Frequency based Embedding and 2) Prediction based Embedding. Let us try to understand each of these methods in detail.



$$cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

$$cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{V} u_i v_i}{\sqrt{\sum_{i=1}^{V} u_i^2}\sqrt{\sum_{i=1}^{V} v_i^2}}$$

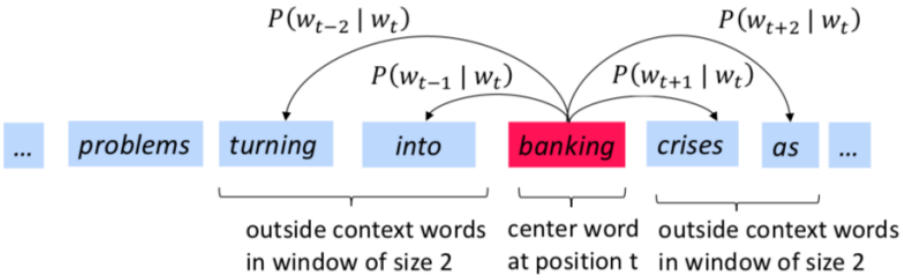
(a) Distance between two word vectors

The first type is to use frequency vector (count Vector) for learning an embedding. Consider a Corpus C of D documents $\{d_1, d_2 \ldots . . d_D\}$ and N unique tokens extracted out of the corpus C. The N tokens will form our dictionary and the size of the Count Vector matrix M will be given by $D \times N$. Each row in the matrix M contains the frequency of tokens in document $D(i)$.

The big idea is like this: Similar words tend to occur together and share similar context words for example: in a sentence 'Apple is a fruit. Mango is a fruit'. Here, *Apple* and mango tend to have a similar context i.e fruit. We mainly care about the frequency of co-occurrence.For a given corpus, the co-occurrence of a pair of words say w_1 and w_2 is the number of times they have appeared together in a Context Window. Here is another important concept of Context Window, which is specified by a number and the direction. So what does a context window of 2 (around) means? Let us see an example in Figure 2b.

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

(a) Co-Occurrence Frequency



(b) Context Window

The idea of word vectors, or embedding, is to represent a word using an d-dimensional vector that can be used to calculate the distance with another word.

The objective function for training embedding on such matrices is to predict any word by the context of a fixed-size window. Here we use the skip-gram, meaning that each word has a skip-connection with all words in its context. Then the likelihood can be defined as

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m<j<m, j\neq 0} Pr(w_{t+j}|w_t;\theta). \tag{4}$$

The core problem then is about how to define the conditional probability

$$Pr(w_{t+j}|w_t;\theta) = \frac{\exp(u_{w_t} \cdot v_{w_{t+j}})}{\sum_{k\in V} \exp(u_{w_t} \cdot v_{w_k})} \tag{5}$$

where

$$u_i, v_i \in \mathbb{R}^d, \tag{6}$$

are embedding vectors. We calculate the gradients of negative log-likelihood with respect to the parameters: in this case, the embedding vectors of each word.

$$\mathcal{J}(\theta) =_{t=1}^{T} \sum_{-m<j<m, j\neq 0} \log Pr(w_{t+j}|w_t;\theta). \tag{7}$$

here θ is all the parameters u_i, v_i. Then we simply run SGD

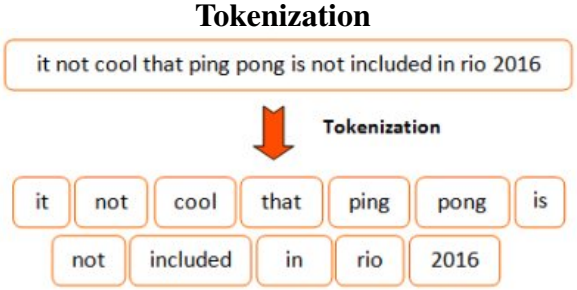
$$\theta = \theta - \eta \nabla \mathcal{J}(\theta). \tag{8}$$

2 Text Processing

Text processing is an important step before converting the sentences to their respective vectors. Text processing involves splitting of sentences into their respective words or removal of some commonly occurring words which are not required to understand the context of the sentence.

2.1 Tokenization

Tokenization involves splitting of text into it's corresponding words. Tokenization also removes commonly occurring characters such as punctuations which give no information about the context of the words.

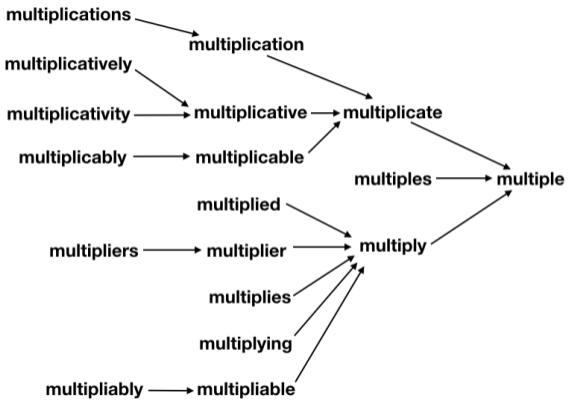


Tokenizers are language models which are trained on huge corpus of data and can mark the end or beginning of a sentence (sentence tokenizing). Word tokenizer is build from regex functions which match any digit/ character/ period and ignores white spaces.

2.2 Lemmatization

A sentence might use lot of words with similar meaning, It might be unnecessary to store all forms of the the same word in the vocabulary and hence bringing the words back to their root form might be advantageous in terms of space. Lemmatization involves the removal of inflectional endings to return the base or dictionary form of the word.

Lemmatization



3 Handling Variable Length Vectors

After converting the words/ snetence to their respective vectors it is often the case that we get floating points or vectors as input. The input is not always if fixed shape or size and will keep changing based on the type and representation of the sentence as input. It is important to make the input to a fixed size representation so as to allow batch processing of the data.

Question : Is there any advantage (other than speed) in processing input data in batches(Stochastic Learning) as compared to processing a single data point (Online Learning) at each time step in terms of optimizing a deep learning model?

Different Input Vector Sequences

```
[                                   [
[1    ,2    ,3    ],                [1.1  ,2.3  ,3.8  ],
[1    ,1    ],                      [1.6  ,1.9  ],
[2    ,5    ],                      [2.8  ,5.3  ],
[6    ,7    ,8    ,1    ]           [6.6  ,7.5  ,8.7  ,1.1  ]
]                                   ]
Integer input                       Floating point input

[
[[1.1  ,2.3  ,3.8  ],[1.6  ,1.9  ,2.6],[2.8  ,5.3  ,6.9]],
[[6.6  ,7.5  ,8.7  ],[1.1  ,7.2, ,2.3]]
]
Vector inputs input
```

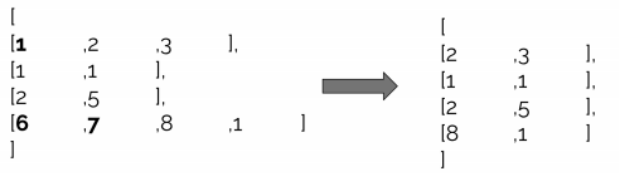
3.1 Truncation

The input vectors are truncated up to a certain length to ensure that all the input vectors are of similar size. Truncation generally causes loss of information due to removal of some values from the input vector sequence.

3.1.1 Pre-Sequence Truncation

The input vectors are truncated from the beginning to the length of the smallest length input vector.

Pre-Sequence Truncation



3.1.2 Post-Sequence Truncation

The input vectors are truncated from the end to the length of the smallest length input vector.

Post-Sequence Truncation

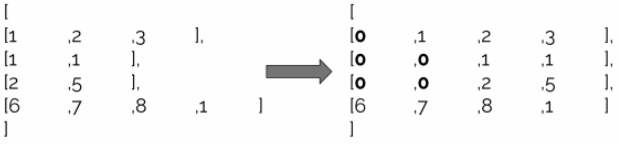
3.2 Padding

The input sequences are padded with zeros upto the length of the largest input sequence vector. Padding adds noise(zeros) to the input data and in some cases corrupt the data.

3.2.1 Pre-Sequence Padding

The input vectors are padded from the beginning to the length of the largest length input vector.

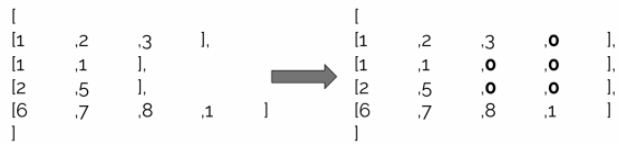
Pre-Sequence Padding



3.2.2 Post-Sequence Padding

The input vectors are padded from the end to the length of the largest length input vector.

Post-Sequence Padding

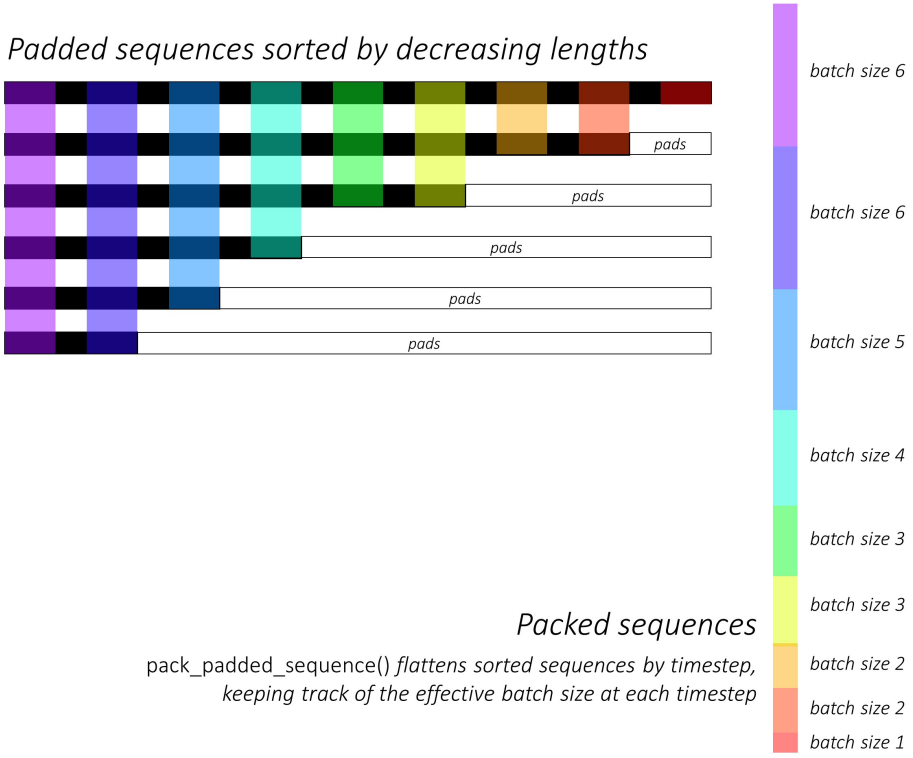


Question : In comparison between truncation and padding, which is a better pre-processing technique and why?

3.2.3 Pack Padded Sequence

Padding the input sequences causes the model to perform additional computations on the padded sequences. To eliminate the wasteful computations, one should efficiently hide the zeros while maintaining the integrity of the fixed vector batch size. This can be made possible by storing the input sequence as tuple of two lists. One containing the elements of the sequence interleaved by time steps and other containing the batch size at each step.

Packed Sequence Visualization



Pytorch Sample Code for Pack Padded Sequence

```
import torch
import torch.nn.utils.rnn as rnn

input = [torch.tensor([1,2,3]), torch.tensor([3,4])]
padded_input = rnn.pad_sequence(input, batch_first=True)

>>> tensor([[1,  2,  3],[3,  4,  0]])

rnn.pack_padded_sequence(padded_input, batch_first=True, lengths=[3,2])

PackedSequence(data=tensor([1, 3, 2, 4, 3]), batch_sizes=tensor([2, 2, 1]))
```

The two input vectors are represented as a list of tensors with sizes 3 and 2 respectively. The input vectors are padded to the length of 3 and then packed based on the time step and batch size at each time step.

3.3 Bag of Words

Input vectors are usually viewed as a sequence encoding of the words in a sentence. The words in a sentence can be split according to some statistic to create a meaningful representation of the words which gives bag of words. This gives a bag of words, which simply counts how many times each word occurs in a sentence.

$$\text{the cat sat on the mat} \rightarrow \{\text{cat,mat,on,sat, the, the}\}$$

Bag of Words is computationally easy to work with. Serves as crude way to capture what the sentence is about.
By maintaining a global set of the words and their corresponding word count in a particular sentence. The sentences can be converted into a list of fixed length frequency based vectors.

Bag of Words Representation

« The dog started to run after the cat, but the cat jumped over the fence. »
« The cat's food was eaten by the dog in a few seconds. »
« The cat attacked the bird the other day. »

↓

```
[ « dog », « start », « run », « cat », « cat », « jump », « fence » ]
[ « cat », « food », « eat », « dog », « second » ]
[« cat », « attack », « bird », « day » ]
```

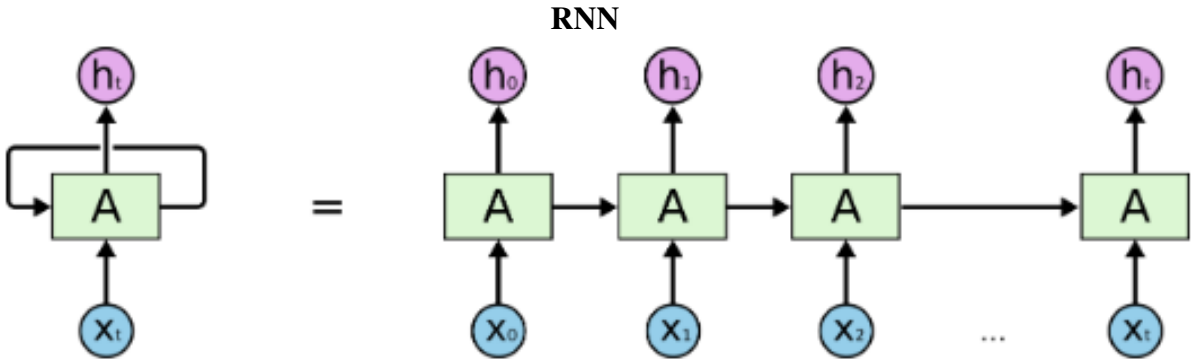
↓

Bag-Of-Words representation

	Dog	Start	Run	Cat	Jump	Fence	Food	Eat	Second	attack	Bird	Day
1	1	1	1	**2**	1	1	0	0	0	0	0	0
2	1	0	0	1	0	0	1	1	1	0	0	0
3	0	0	0	1	0	0	0	0	0	1	1	1

3.4 Recurrence

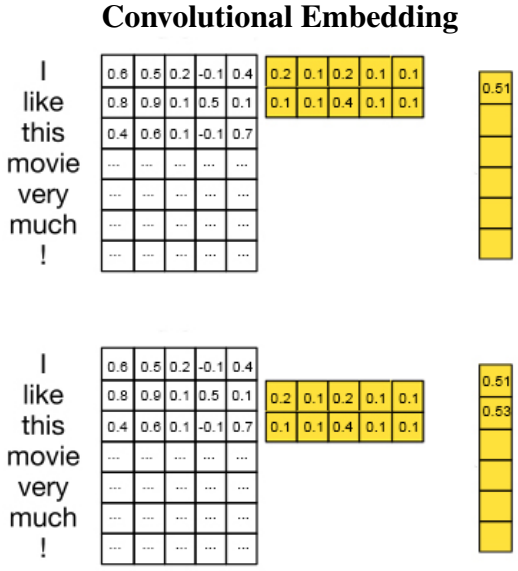
Instead of maintaining a fixed vector length, one can compute the fixed length hidden feature vector for the whole sentence using recurrent neural network. RNN's compute the hidden representation for each word in a recurrent manner, i.e the hidden state of a word in the previous time step is fed back into the RNN along with the current word. RNN's assume relationship of the current word with respect to all the previous words. which leads the final word to capture the meaning of all the previous words(the whole sentence). Hence, the feature vector of the final word outputted from an RNN can be used as the representation for the whole sentence.



3.5 Convolution

Convolutional Neural Networks on the other hand don't have memory of the past words. CNN's use filters to encode the information of the sentence by computing a representation of the sentence using a sliding window approach. These filters in further layers extract very high-level features and create multiple feature maps. The weights in CNN's are not fully connected like normal neural network but instead are connected to only a subset of the data. This reduces the number of parameters in the neural network.

Question : Compute the number of parameters in a convolutional filter of size (3, 3) which takes the [R, G, B] channels of an image as input and outputs a feature vector of size 10?



The sliding window goes over each word or a sequence of words to encode them into fixed size embedding.

4 The Probability of a Sentence?

Till now we've seen how the words can be represented as vectors which cab be used as input to the language model. Now we'll see what happens inside a language model given the vectors, i.e how a language model processes the sentence/word vectors. (pre-deep learning)

4.1 The Probability of Occurrence

Given the sequence of input vectors, language models inherently compute the probability of occurrence of a sentence. More specifically, it computes the probability of occurrence of specific sequence of words.

$$P\left(w_1, \ldots, w_m\right) = \prod_{i=1}^{i=m} P\left(w_i | w_1, \ldots, w_{i-1}\right) \approx \prod_{i=1}^{i=m} P\left(w_i | w_{i-n}, \ldots, w_{i-1}\right)$$

This is especially useful in machine translation and seq to seq models where the model should output the translated words in the right sequence. The model calculates scores for each sequence of words and gives higher probability/ score to the joint probability of the correct sequence over all possible combinations of sequences.

Language models can also be thought of as generating probability scores for the next word in the series given the past words.

$$P\left(\boldsymbol{w}^{(t+1)} | \boldsymbol{w}^{(t)}, \ldots, \boldsymbol{w}^{(1)}\right)$$

Where w^{t+1} can be any word in the vocabulary of the text.

4.2 N-gram Representation

Computing the probability of a word given all the previous words is hard to compute. Instead, if we limit the dependency to a few previous words instead of all the previous words, the computation becomes easier.
N-gram models assume Markov dependency which states that at any time(t+1) the word w^{t+1} only depends on the preceding N-1 words.

$$P\left(\boldsymbol{w}^{(t+1)}|\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(1)}\right) = P\left(\boldsymbol{w}^{(t+1)}|\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(t-N+2)}\right)$$

4.2.1 Statistical Analysis of N-gram

The probability of N-grams is computed by counting the N-grams and dividing them by N-1 grams.

$$P\left(\boldsymbol{w}^{(t+1)}|\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(1)}\right) \approx \frac{\text{count}\left(\boldsymbol{w}^{(t+1)},\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(t-N+2)}\right)}{\text{count}\left(\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(t-N+2)}\right)}$$

4.2.2 Sparsity Problem

The N-gram model suffers with the sparsity problem. It is not always that a particular sequence of words occur in the text corpus. In the case where the sequence of words doesn't appear in the text corpus, the numerator or the denominator count can go to zero while computing the probability of the sequence.

The numerator count can go to zero when the sequence $\boldsymbol{w}^{(t+1)},\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(t-N+2)}$ never occurs in the text corpus. In such cases a small value ϵ is added to the numerator.
If instead the sequence $\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(t-N+2)}$ never occurs in the text corpus, the denominator will become zero. To resolve this, we instead count the occurrence of $\boldsymbol{w}^{(t)},\ldots,\boldsymbol{w}^{(t-N+1)}$ and so on until we encounter a non-zero count.

5 Syntactical Analysis

Is the probability of sentence enough to derive everything from the sentence? **No**.
Consider the sentence "**pancakes made I morning the in**", this sentence doesn't make sense and it's grammatical structure is incorrect. Hence, it is important to understand the syntactical structure to make sense whether a sentence is grammatically correct or not.

Syntactical Analysis helps us understand the roles played by different words in a body of text. The words themselves are not enough. There is some evidence that human understanding of language is, in part, based on structural analysis.

Syntactic analysis is defined as analysis that tells us the logical meaning of certain given sentences or parts of those sentences. We also need to consider rules of grammar in order to define the logical meaning as well as correctness of the sentences. It is important for the model to understand the structure of the sentence so that it can output meaningful sentences.

5.1 Context Free Grammar

Context Free Grammar is a set of rules which define a well defined sentence in English language. Each rule has a left-hand side, which identifies a syntactic category, and a right-hand side, which defines its alternative component parts, reading from left to right.

Some of the common syntactic categories are:

- np - noun phrase
- vp - verb phrase
- s - sentence
- det - determiner (article)
- n - noun
- tv - transitive verb (takes an object)
- iv - intransitive verb
- prep - preposition
- pp - prepositional phrase
- adj - adjective

Let take an example of the sentence "**the giraffe dreams**". The syntactical structure can be represented with the help of a parse tree

Parse Tree

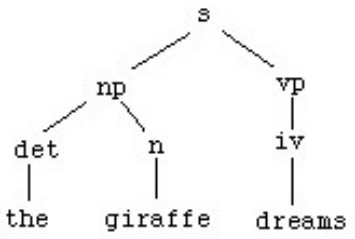
```
s    ⟶ np vp

np   ⟶ det n
vp   ⟶ tv np
     ⟶ iv

det  ⟶ the
     ⟶ a
     ⟶ an

n    ⟶ giraffe
     ⟶ apple

iv   ⟶ dreams
tv   ⟶ eats
     ⟶ dreams
```



the rule "s –> np vp" means that "a sentence is defined as a noun phrase followed by a verb phrase".

A sentence derived by Context Free Grammar can be derived by following the rules of the sentence from left to right (Top Down Parsing) or from right to left (Bottom Up Parsing).

5.2 Parse Tree

Parse trees represent hierarchical relationships between elements of text. A particular parse tree indicates how a particular grammar analyzes the syntactic structure of a particular sentence.

A sentence can be made sense by either following the parse tree in a Bottom Down approach or a Top Down approach.

5.2.1 Top Down Parsing

In top down parsing we start from the root node and work our way down to the actual sentence. We start with the assumption that it's a sentence (s) and prove that it's actually a sentence by following the rules from left to right.

If we want to prove that the input is of category sentence (s) and we have the rule S -> NP VP, then we will try next to prove that the input string consists of a noun phrase followed by a verb phrase. If we furthermore have the rule NP -> Det N, we try to prove that the input string consists of a determiner followed by a noun and a verb phrase.

For the sentence "**the giraffe dreams**", the top down parsing would seem something like

s -> np vp -> det n vp -> the n vp -> the giraffe vp -> the giraffe iv -> the giraffe dreams

We work from the top and reach the data following the rules in a left to right manner.

5.2.2 Bottom Up Parsing

In this approach we begin with a concrete data provided by the input data and build our way to the top and prove that it's a sentence (s). We move from low-level information to more abstract high level information until we satisfy the criteria that it's a sentence.In bottom up parsing, we use our Context Free Grammar rules right to left.

It means that if we have a rule "s-> np vp", if we find "np" and "vp" are next to each other in the dependency structure then we can conclude that we have found a "s".

For the sentence "**the giraffe dreams**", the bottom up parsing would seem something like

the giraffe dreams -> det n iv -> np vp -> s

We work from the bottom and reach the sentence(s) following the rules in a right to left manner.

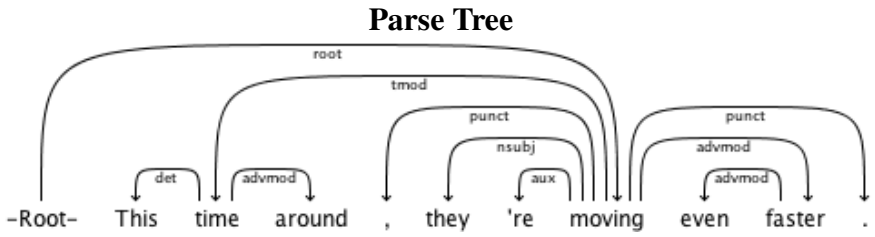
5.3 Dependency Grammar

Dependency Grammar is a class of grammar which focuses on the words rather than the parse structure (context free grammar). Grammars that are built primarily on constituents are known as phrase structure grammars. Phrase structure grammars are thus constituent-based, while dependency grammars are word-based. Dependency grammars assume that sentence and clause structure

derives from dependency relationships between words. A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words which modify those heads.

The grammatical relationships are represented as universal dependencies. Some of the common dependencies are:

- tmod: temporal modifier
- nsubj: nominal subject
- advmod: adverbial modifier
- punct: punctuation
- det: determiner
- aux: auxiliary



The arrow from the word moving to the word faster indicates that faster modifies moving, and the label "advmod" assigned to the arrow describes the exact nature of the dependency.

5.4 Part of Speech Tagging

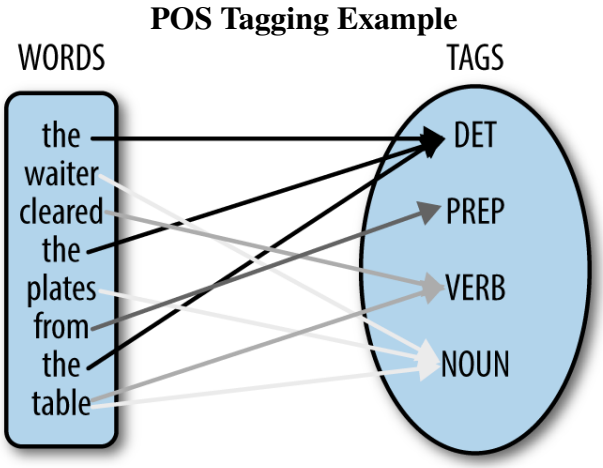
Now that we know the syntactic representations of sentences, we can extract useful information about the sentences such as Part of Speech tags.

POS tagging is the process of assigning a speech tag to each word in the sentence. USeful for applications such as information retrieval and text to speech. POS tagging removes ambiguity about the word in a sentence. For example:

Plants(Noun) need light and water.

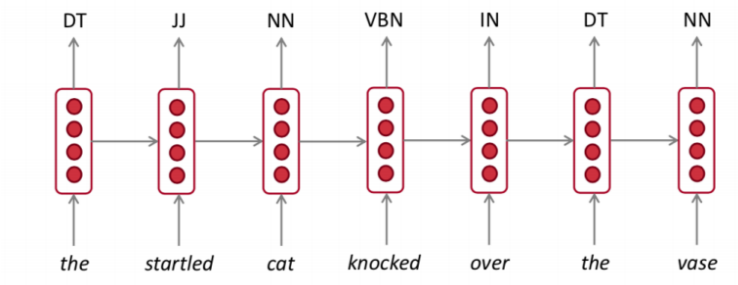
Each one plant(Verb) one.

The word plant pays different roles in both the sentences. To remove this kind of ambiguity we can tag the words with their respective part of speech.

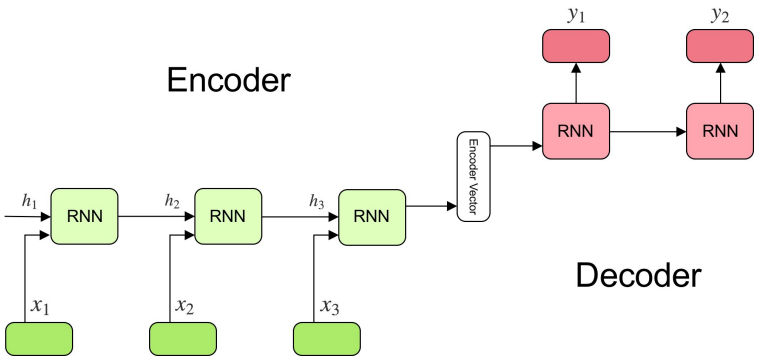


6 Sequence to Sequence Learning

An typical deep learning based technique is the sequence to sequence learning (Seq2seq), and it has wide applications in NLP. To begin with, we give an example of part-of-speech tagging (POS tagging) task in Figure 17a. The input to the model is a sentence, and the ouput is a sequence of tags representing each word's function in the sentence, nouns, verbs, adjectives, etc.



(a) POS tagging



(b) RNN structure

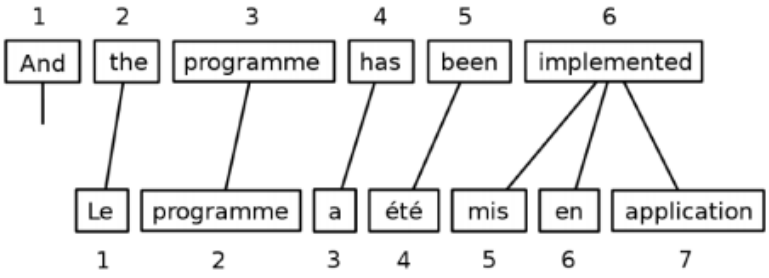
This kind of problems can be approached using an recurrent neural network (RNN). The input sentence is encoded into a vector, and the vector is further decoded to another sequence of tags.

6.1 Attention Models

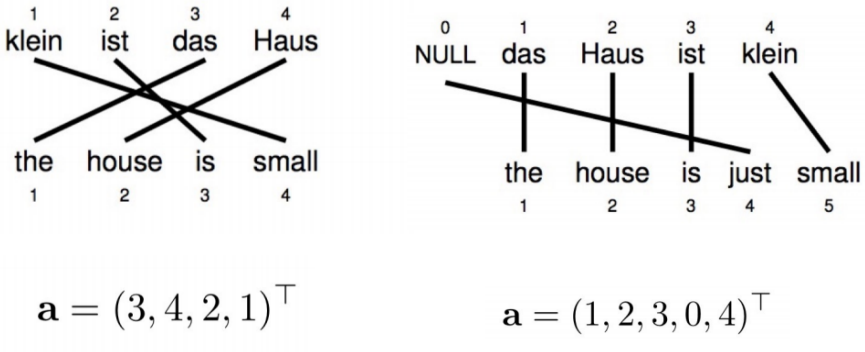
Before introducing the model, we introduce an important application in NLP, which was developed long before deep learning: the machine translation (Figure 18a). In this application, each word in the original text only has one or two corresponding words in the target text. In statistical machine translation, the algorithm needs to calculate the "alignment" between source and target sequence, i.e. which word corresponds to each word in the source, in Figure 18b.

To do this task using neural network, we could use the RNN structure, like Figure 18c. so it makes more sense if the embedding vector has narrow influence on the output, i.e. to focus on a small region, which gives rise to the attention model.

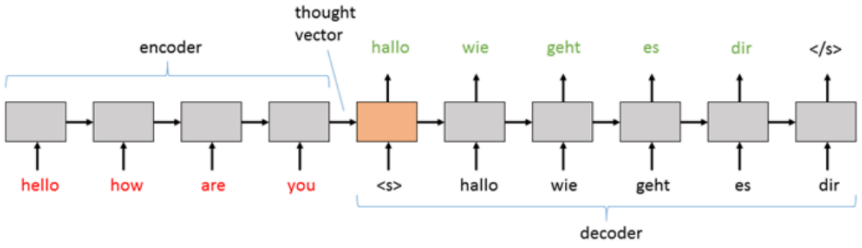
The attention model is a neural network equivalency of the alignment in the context of machine translation.



(a) Example of machine translation

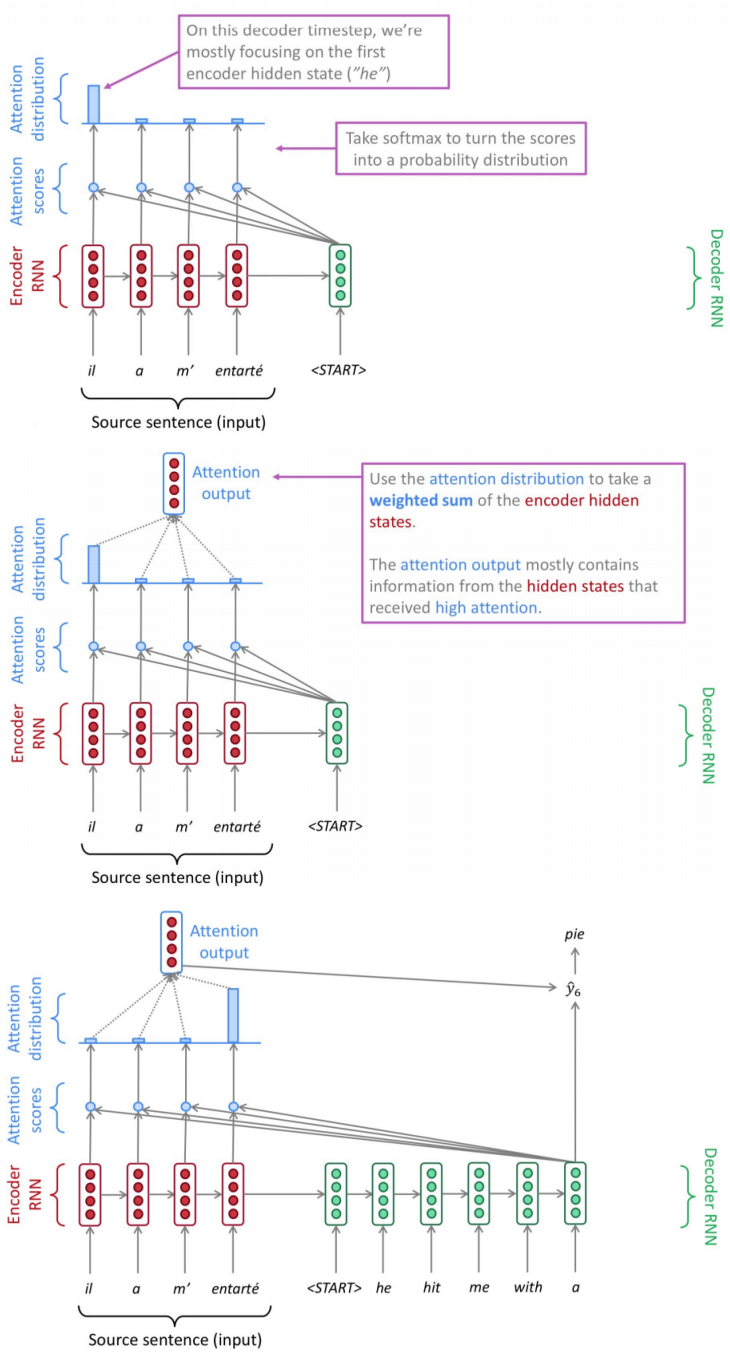


(b) Example of alignment



(c) RNN structure for translation

The Essence of this attention model is to: the network focus on a small part of the source sentence instead of the whole sentence while decoding.



References

[1] Dwarampudi Mahidhar Reddy, and N V Subba Reddy. EFFECTS OF PADDING ON LSTMS AND CNNS

[2] Understanding LSTM Networks

[3] Bags of Words

[4] Natural Language Processing with Deep Learning (CS224n), Stanford NLP

[5] Daniel Jurafsky James H. Martin. Speech and Language Processing

[6] Allen B. Tucker. Overview of NLP: Issues and Strategies

[7] Patrick Blackburn,and Kristina Striegnitz. Bottom Up and Top Down Parsing