Natural Language Processing

What is language?

Fixed vocabulary of words shared by humans

How do we express language as a mathematical model?

speech & text

NLP is broadly defined as the automatic manipulation of natural language through software

Tokenization, Stemming, Lemmitization

Tokenization: breaking down text into words
→ commonly at spaces.

Stemming → getting base words and removing derivational affixes.

Lemmatization: Remove inflection endings added to a word that change its meaning

N-Grams
combining n words for representation purposes

1-gram / unigram model tokenizes into one word combinations

bigram - tokenize into combinations of 2 words each token

Natural, Language, processing, is, essential

Natural Language, Language processing, processing is, is essential

up to n-gram model

Transformation methods:

tf-idf — Score vocabulary to provide adequate weight of a word in proportion to impact of its meaning in a sentence.

product of 2 independent scores

→ tf (term frequency)

→ idf (inverse document frequency) => common / rare across documents

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

i in document j,

$tf_{i,j}$ = # of occurrences of i in j

df_i = # of documents containing i

N = total # of documents

One hot encodings:

Represent words in numeric form

row length = length of vocabulary

Column length = length of observation

	The	cat	Sat	on	the	mat
the	1	0	0	0	1	0
Cat	0	1	0	0	0	0
on	0	0	0	1	0	0
⋮						

Word embeddings:

words & phrases are mapped to vectors of real numbers

want similar words to be projected closer to eachother

Bag of words:

Tabular format => columns representing the total vocabulary and row representing a single observation

Intersections are the count of the word of the column, in a particular observation.

=> Useful for Spam & sentiment analysis

observation ID	word 1	word 2	word 3
1111	4	6	3
0011	0	1	8
0222	0	2	3

Disregards order / grammar, losing the context that a word is used

The importance of a word is often inversely related to its use in a sentence (limitation / problem)

Embedding form of a word => each word in a hot encoded form is multiplied by the embedding matrix to give word embeddings for the sample. (more on next page)

After one-hot encoding: cat → [0,1,0] (one hot encoding)

Embedding matrix: each row corresponds to the embedding vector for a word → numbers to represent the "meaning" of a word. # of features → # of columns

$$\begin{bmatrix} 2.3 & 1.5 \\ 0.7 & -0.9 \\ 0.2 & 0.5 \end{bmatrix} \begin{matrix} \rightarrow \text{dog} \\ \rightarrow \text{cat} \\ \rightarrow \text{bird} \end{matrix}$$ (learned through training)

The result should be word vectors that capture the semantic meaning/context of each word in a continuous vector space, with each word being outputted w/ its context.

RNNs => often a word as input

=> can vary input length from words to sentences

Each input occurs at time t, and time t-1 is also inputted in addition to t=t input.

Many to many architecture

Many to one architecture

One to many architecture

Many to many architecture

Lexical tokenization: conversion of text into meaningful tokens which are then categorized

=> identifiers, operators, grouping symbols and data types

The lexer is responsible for tokenizing words

Infix: Affix inserted into word stem

↓

attached to word stem, that forms a new word.

Stop words => common words that are filtered out before NLP due to their insignificance

Sentiment analysis: understand the tone of a piece of writing

Tokenize the sentences → break down into words

Remove the stop words

Condense all forms of words into their single form

Vectorize text: Turn text into numerical representation for the classifier

① Tokenization: Breaking down chunks into smaller pieces.
=> Sentence & word tokenization

② Remove stop words

Normalize words
Stemming: A word is cut off at its stem

③ Lemmatization (stemming but keeps words in their simplest form rather than truncating)
only one offered by SpaCy

This happens automatically with speech tagging and named entity recognition when nlp() is called

④ Vectorization of a text: Transforms a token into a vector that uniquely represents the features of the token.

For sentiment analysis => use a dense array

Vectorization is automatically done with nlp() call.

For classification, a basic workflow looks like:
Split data into training and evaluation sets.
Select a model architecture
use training data to train model
use test data to evaluate performance of model
use trained model to generate predictions (-1, 1)

Spacy provides a pipelining functionality in a JSON file

under en_core_web_sm

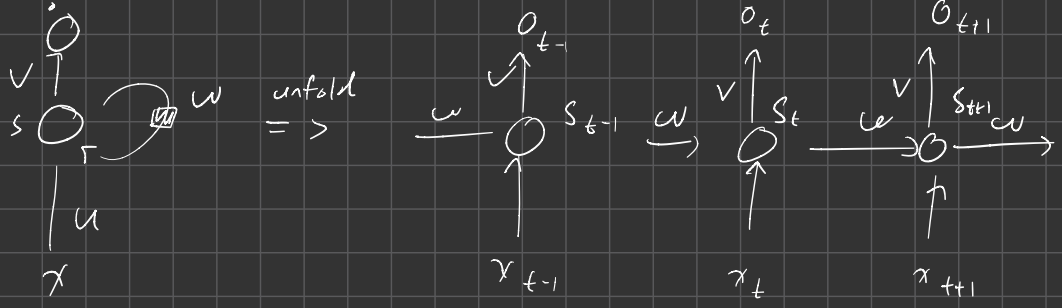
Built-in pipelining components includes textcat => enables you to assign categories / labels to a dataset, for training purposes

Add the text_cat component to the existing pipeline
Add labels to the textCat component
Load & shuffle data
Train model and evaluate each loop

Introduction to RNNs

RNNs make use of sequential information

Perform the same task for every element of a sequence with the output depending on previous computations



x_t is the input at time step t.

s_t is the hidden step at time t.

Calculated based on the previous hidden state and input at current time step

$$s_t = f(U x_t + W s_{t-1})$$

↓ usually non-linearity layer

O_t is the output step at time t

if we wanted

o_t is the output at step t. If we wanted to predict the next word in a sentence, it is a vector of probabilities across the vocabulary $o_t = \text{softmax}(V_{s_t})$

s_t can be thought of as the hidden memory of the network

$\Rightarrow$ captures what happened in all the previous time steps.

o_t is calculated solely based on memory at time t

A RNN shares the same parameters (M, V, W) across all steps

Sometimes the output O_t is not needed

Language modelling & Generating text

predict the next word, given the previous words

We set $O_t = x_{t+1}$ => output at step t to be the next actual word

Generating Image Descriptions

paired with CNNs

Similar to training a neural network

=> Backpropagation becomes Backpropagation through time

Vanilla RNNs have difficulty learning long-term dependencies

=> Vanishing / exploding gradient problem

LSTM Networks

memory are called cells

Black boxes that take as input the previous state h_{t-1} and current input x_t.

They decide what to keep and erase from memory.

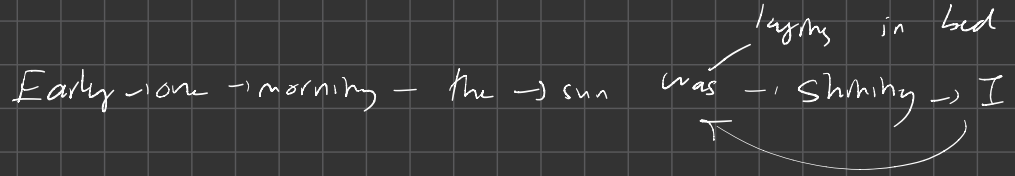
combine previous state current memory, and input

Language Models from Scratch

Goal: Assign a probability for all sentences

10^{50} (10 words per sentence)

Graph:



Can get crazy samples

$P(x_n \mid x_{n-1})$ (not very insightful)

Need to cover long range dependencies

Fourier / Taylor Series

Neural networks

$\sin(x) - \frac{x^2}{2}$, just need input/output pairs

Word2Vec, Glove

attention x words → probabilities

Context for every word + original words

Stacking Attention allows for complex relationships