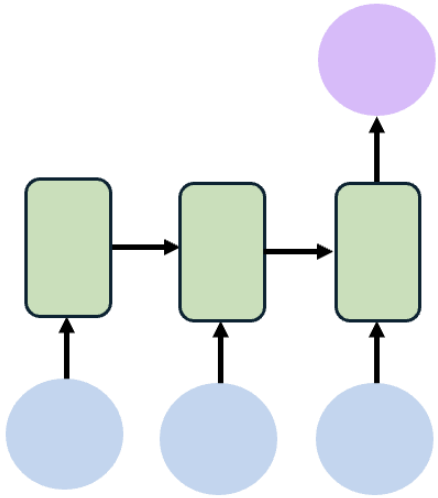
Introduction to Natural Language Processing (NLP)

Natural Language Processing

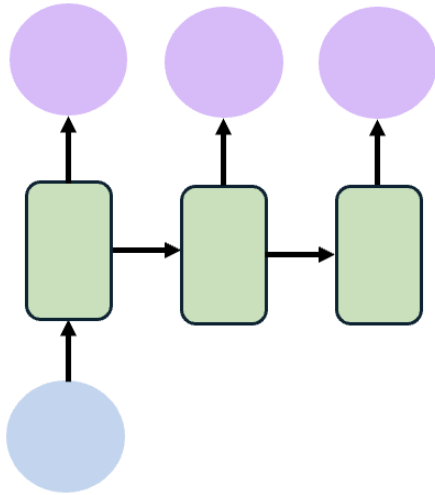
- Natural language processing (NLP) refers to the branch of data science concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
- Several NLP tasks break down human text and voice data in ways that help the computer make sense of what it's ingesting.
- Some of these tasks include the speech-recognition, part of speech tagging (POS), word sense disambiguation, sentiment analysis, text classification, natural language generation
- Text is unstructured data – typically a ‘stream’ - so, in order to be processed by a computer, it need to be properly pre-processed to extract numerical features out of it.

Applications of NLP

- Text Classification
- Machine Translation
- Summarisation
- Sentiment Analysis
- Chatbots!

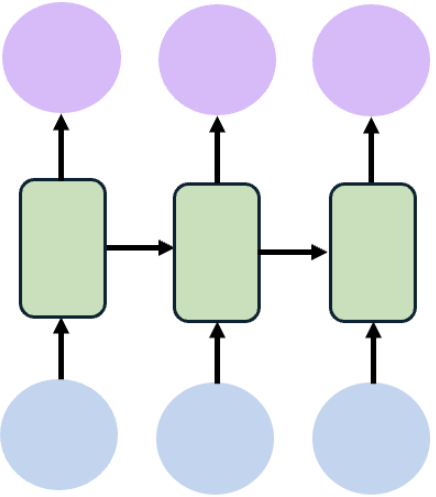


Many to One
Sentiment Classification



One to Many
Image Captioning

“A baseball player throws a ball.”



Many to Many
Machine Translation

Terminology

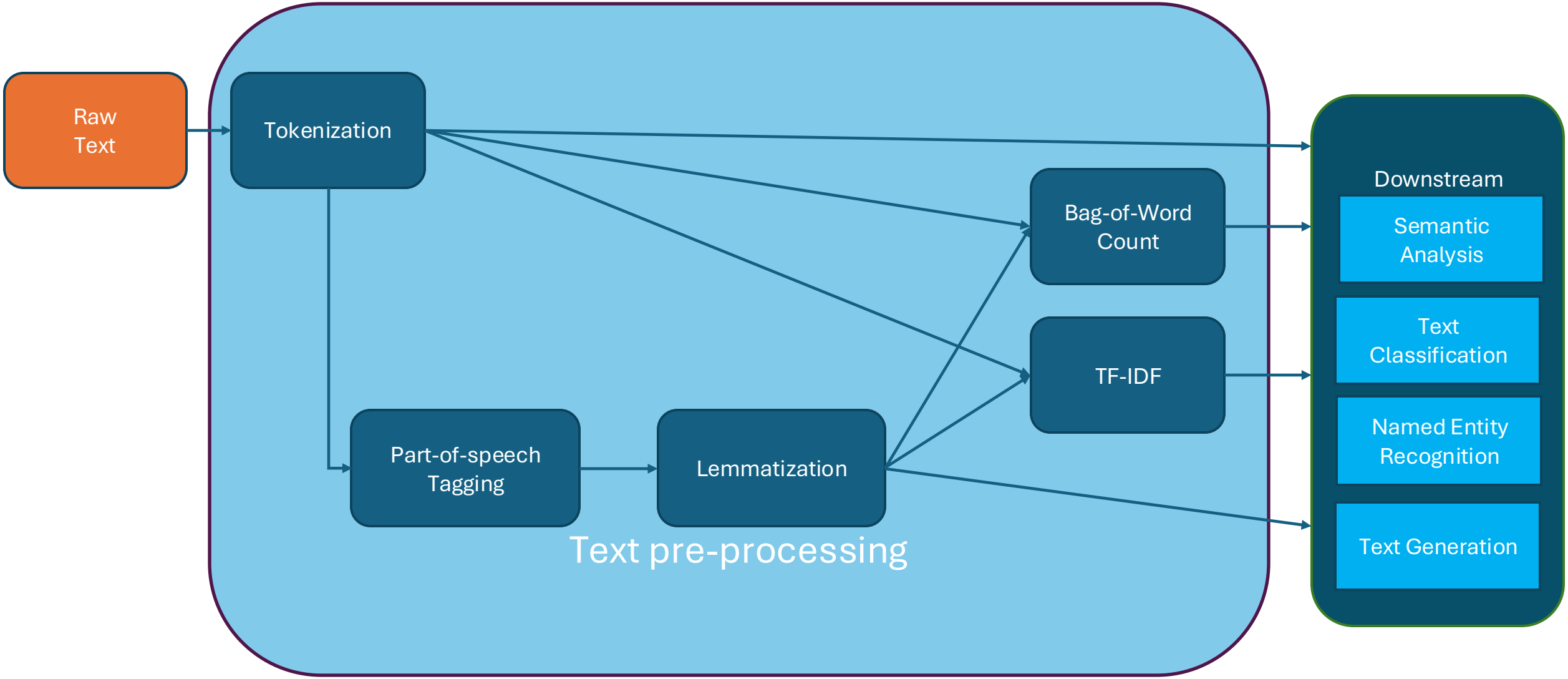
- **Document**: some text.
- **Corpus**: a collection of documents.
- **Dictionary**: a list of all the words in a document
- **Token**: a well-defined unit inside a document. It can be a word or a sub-word.
- **Lemma**: the root or base form of a word (e.g. played, playing => play)
- **Vector**: a mathematically convenient representation of a document.
- **Model**: an algorithm for transforming vectors from one representation to another.

Text pre-processing

Text pre-processing often consists of these steps:

- **tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
  - optionally tokenization can be followed by **stop-word removal** and **stemming** or **lemmatization**
- some form of **vectorisation**, such as**:**
  - **counting** the occurrences of tokens in each document (bag-of-words approach)
  - **normalizing** and weighting with diminishing importance tokens that occur in a plurality of samples. (TF-IDF approach)
  - **word embeddings** (Word2Vec, GloVe, FastText, etc.)

Example of NLP pipeline



Pre-processing: Text segmentation

Tokenization (breaking text into ‘tokens’)

- Tokenization is the process of breaking down text into smaller, meaningful units called **tokens**.
- This crucial step prepares the text for further natural language processing tasks, such as part-of-speech tagging and sentiment analysis.

- Input stream: “Hey Nick! How are you?”

tokens = ['Hey', 'Nick', '!', 'How', 'are', 'you', '?']

Tokenization

- Example with Natural Language ToolKit (NLTK):

```
import nltk
from nltk.tokenize import (word_tokenize,
                           sent_tokenize,
                           TreebankWordTokenizer,
                           wordpunct_tokenize,
                           TweetTokenizer,
                           MWETokenizer)
text="Hope, is the only thing stronger than fear! #Hope #Amal.M"
```

```
print(word_tokenize(text))
```

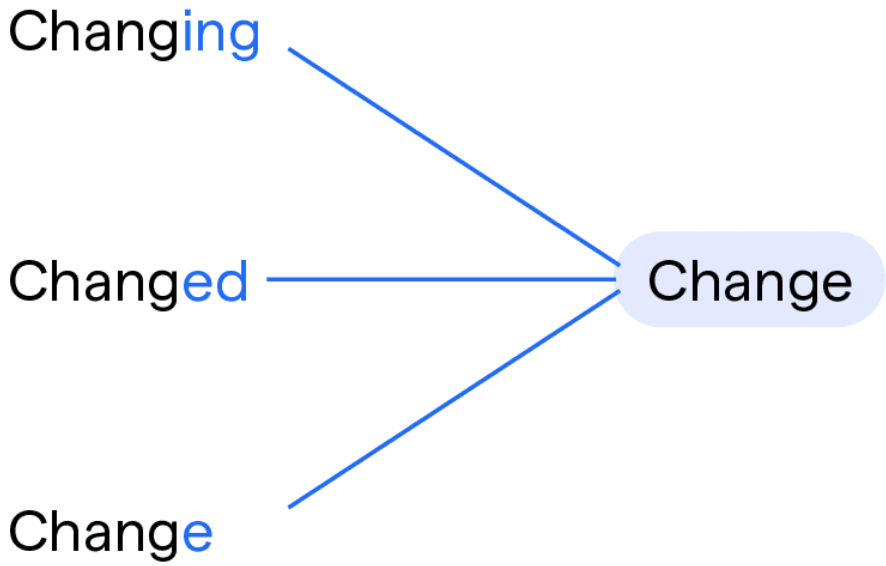
```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal.M']
```

```
print(sent_tokenize(text))
```

```
['Hope, is the only thing stronger than fear!', '#Hope #Amal.M']
```

Lemmatization (reduce word variation)

- Lemmatization is the process of reducing words to their base or dictionary form, known as the **lemma.**
- This helps to minimize the impact of word inflections and conjugations, allowing for more accurate analysis and improved performance in natural language processing applications.



Stemming vs Lemmatization

- ‘Stemming’ algorithms apply a set of rules or heuristics to **‘chop off’ prefixes and suffixes** from words to leave the ‘**stems**’.
- Advantages: ‘more efficient’ than lemmatization which considers the context and meaning of words and aims to transform them into their base form according to the rules of the language.
- Disadvantages:
  - **Overstemming** (False Positive): reduces separate inflected words to the same word stem even though they are **not related;**
    e.g. "universal", "university", and "universe" are not related…
  - **Understemming** (False Negative): stemming algorithm reduces inflected words to different word stems, but they **should be related**!
    e.g. “alumnus,” “alumnae,” and “alumni” are related.

Part of Speech (PoS) Tagging (identify word types)

Example: **“The quick brown fox jumps over the lazy dog.”**

- **“The”** is tagged as **determiner** (DT)
- **“quick”** is tagged as **adjective** (JJ)
- **“brown”** is tagged as **adjective** (JJ)
- **“fox”** is tagged as **noun** (NN)
- **“jumps”** is tagged as **verb** (VBZ)
- **“over”** is tagged as **preposition** (IN)
- **“the”** is tagged as **determiner** (DT)
- **“lazy”** is tagged as **adjective** (JJ)
- **“dog”** is tagged as **noun** (NN)

Stop word removal

- Words like ‘a’, ‘the’, and ‘is’ don’t contribute much meaning to the text.
- Prepositions: But words like "in", "on", and "under" that indicate relationships between entities.
- You may want to keep ‘engendered’ pronouns (‘he’, ‘she’, ‘his’, ‘hers’, ‘they’, ‘them’) if you are interested in analysing differences between genders.
- Identifying and removing stop words can improve the accuracy of text analysis by focusing on more meaningful terms.
- Stop word removal is commonly used in **search engines**, text summarisation, and sentiment analysis.

Pre-processing: Vectorisation

Vectorisation

- **Problem:** neural network models and statistical methods do not understand strings and characters!

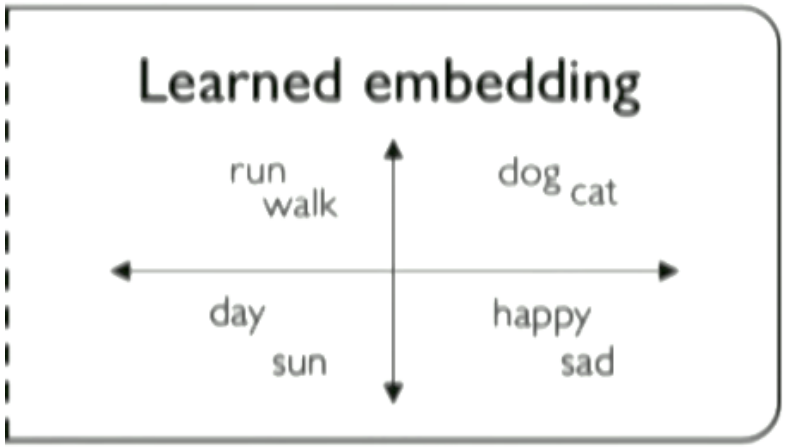
Neural networks cannot interpret words

Neural networks require numerical inputs

- Therefore, we would need to convert them to numbers so they can be used. This process is known as 'vectorisation'
- There are a number of 'vectorisation' methods:
  - Word Embeddings
  - Bag of Words (BoW)
  - Term Frequency-Inverse Document Frequency (TF-IDF)

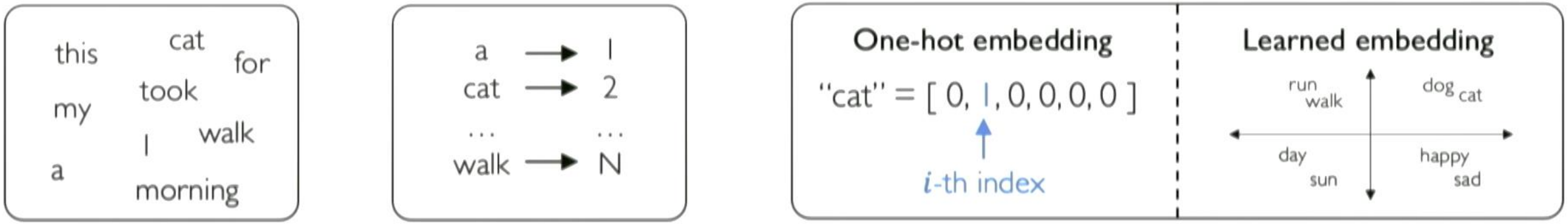
Word Embedding

- Word embeddings are dense, low-dimensional vector **representations of words** that capture **semantic information** about the words.
- Each word is represented as a fixed-size vector in a **continuous** vector space, where **similar words are closer together in the space**.
- Word embeddings capture syntactic and semantic relationships between words and are often used as feature representations in NLP tasks.



Word Embedding Process

- Embedding: transform indexes into a vector of fixed size:



1. Vocabulary: Corpus of words

2. Indexing: Word to index

3. Embedding: Index to fixed-sized vector

Helpful libraries

Libraries:

- **GloVe:** vector-based model that uses statistical co-occurrence information to learn word embeddings
- **Word2Vec:** popular neural network-based model that learns vector representations of words from text data.
- **FastText:** an extension of Word2Vec that considers subword information, improving performance on rare and out-of-vocabulary words.

Bag of Words (Bagging) – tallying (ensemble)

- BoW representation represents each **document as a vector where each element corresponds to the frequency** or presence of a word in the document.
- The vocabulary is constructed from all unique words in the corpus, and **each word is assigned a unique index**.
- The vector for each document is constructed by counting the occurrences of each word in the vocabulary.
- BoW representation does not consider the order of words in the document, only their frequency.

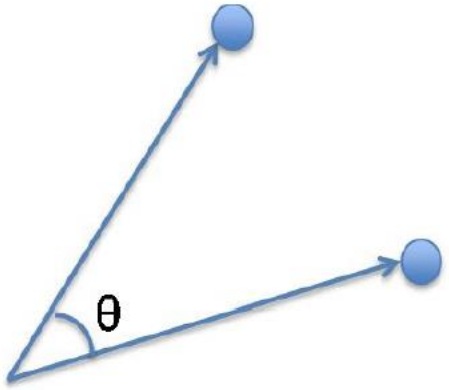
Simple Example:

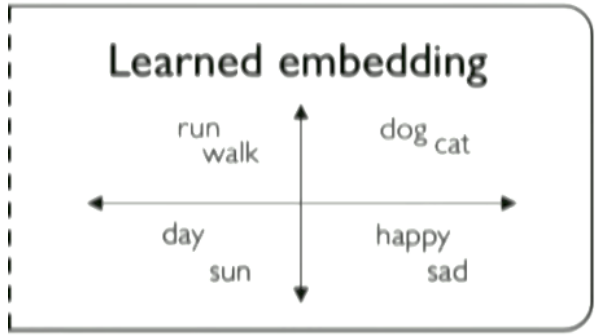
- You have two documents:
  - “Blue House”
  - “Red House”
- ‘Featurise’ on word count:
  - Blue House -> (‘red’, ‘blue’, ‘house’) -> (0,1,1)
  - Red House -> (‘red’, ‘blue’, ‘house’) -> (1,0,1)

Simple Example:

- A document represented as a vector of word counts is called a ‘Bag of Words’ (BoW)
  - Blue House -> (‘red’, ‘blue’, ‘house’) -> (0,1,1)
  - Red House -> (‘red’, ‘blue’, ‘house’) -> (1,0,1)
- You can use cosine similarity on the vectors made to determine similarity:

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$





Simple Example:

- Furthermore, we could improve upon ‘Bag of Words’ (BoW) by adjusting word counts based on their frequency in corpus (the group of all the documents).
- We can use Term Frequency – Inverse Document Frequency (TF-IDF)

Term Frequency–Inverse Document Frequency Measure of importance

- TF-IDF representation measures the **importance of a word** in a document **relative to its occurrence in the entire corpus**.
- It combines two components:
  - **term frequency (TF),** which measures **how often a term occurs** in a **document**,
  - **inverse document frequency (IDF),** which measures the **rarity (uniqueness) of the term across documents in the corpus**.
- TF-IDF assigns **higher weights to terms that are frequent in the document but rare across the corpus**, making it more discriminative than BoW.

TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency

Number of times term t appears in a doc, d

Inverse document frequency

$$\log \frac{1 + n}{1 + \text{df}(d, t)} + 1$$

n ← # of documents

df(d, t) ↖ Document frequency of the term t

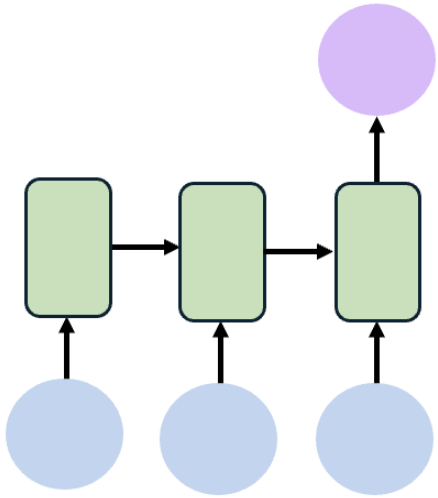
TF-IDF

	di	t	tc	tt	tf	df	idf	tfidf
0	0	the	2	7	0.285714	2	0.000000	0.000000
1	0	car	1	7	0.142857	1	0.176091	0.025156
2	0	is	1	7	0.142857	2	0.000000	0.000000
3	0	driven	1	7	0.142857	2	0.000000	0.000000
4	0	on	1	7	0.142857	2	0.000000	0.000000
5	0	road	1	7	0.142857	1	0.176091	0.025156
6	1	the	2	7	0.285714	2	0.000000	0.000000
7	1	truck	1	7	0.142857	1	0.176091	0.025156
8	1	is	1	7	0.142857	2	0.000000	0.000000
9	1	driven	1	7	0.142857	2	0.000000	0.000000
10	1	on	1	7	0.142857	2	0.000000	0.000000
11	1	highway	1	7	0.142857	1	0.176091	0.025156

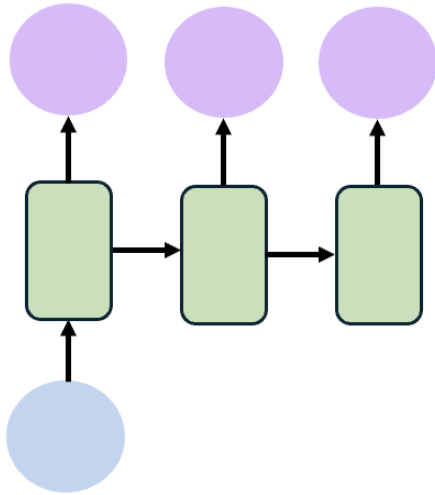
Prepared for applications…

Applications of NLP

- Text Classification
- Machine Translation
- Summarisation
- Sentiment Analysis
- Chatbots!

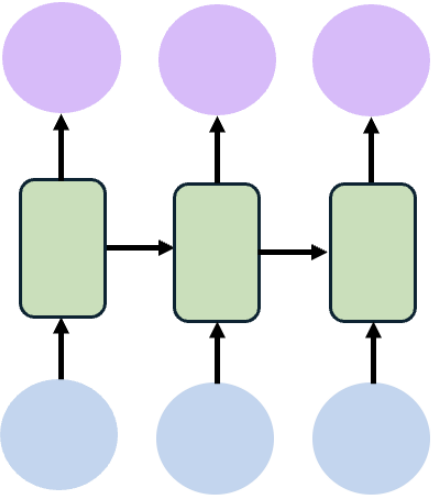


Many to One
Sentiment Classification



One to Many
Image Captioning

“A baseball player throws a ball.”

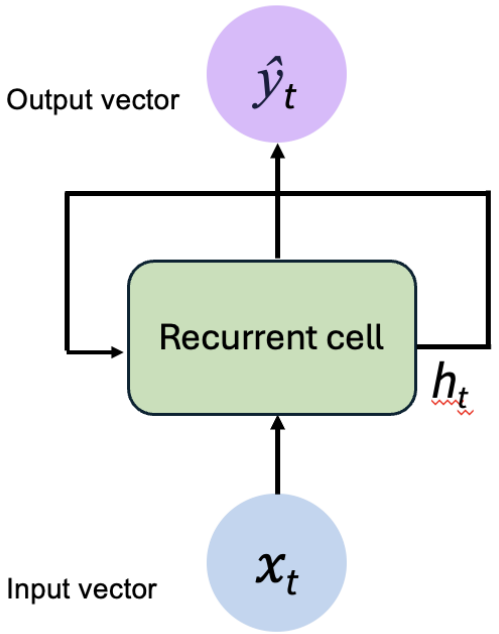


Many to Many
Machine Translation

Next Steps -> RNNs for NLP

- In this session we'll explore a naïve / 'primitive' version of an NLP project to get started
- Future sessions will explore the application of Recurrent Neural Networks (RNNs) and Long-Short Term Memory (LSTMs) for NLP.
- Post 2017 – the combination of 'self-attention' and the transformer architecture (encoder/decoder) – has led to generative pre-trained transformers (GPTs).

RNNs



LSTMs