NLP and Deep Learning
MAT3399

Lecture 1:
NLP Introduction &
Word Representation

Tuan Anh Nguyen @ Aimesoft
ted.nguyen95@gmail.com

Introduction and Goals

- Understand the fundamentals of Natural Language Processing (NLP) and Deep Learning
- Learn the key techniques in text processing and analysis
- Gain hands-on experience in working with NLP libraries and deep learning frameworks
- Complete a course project applying NLP and deep learning

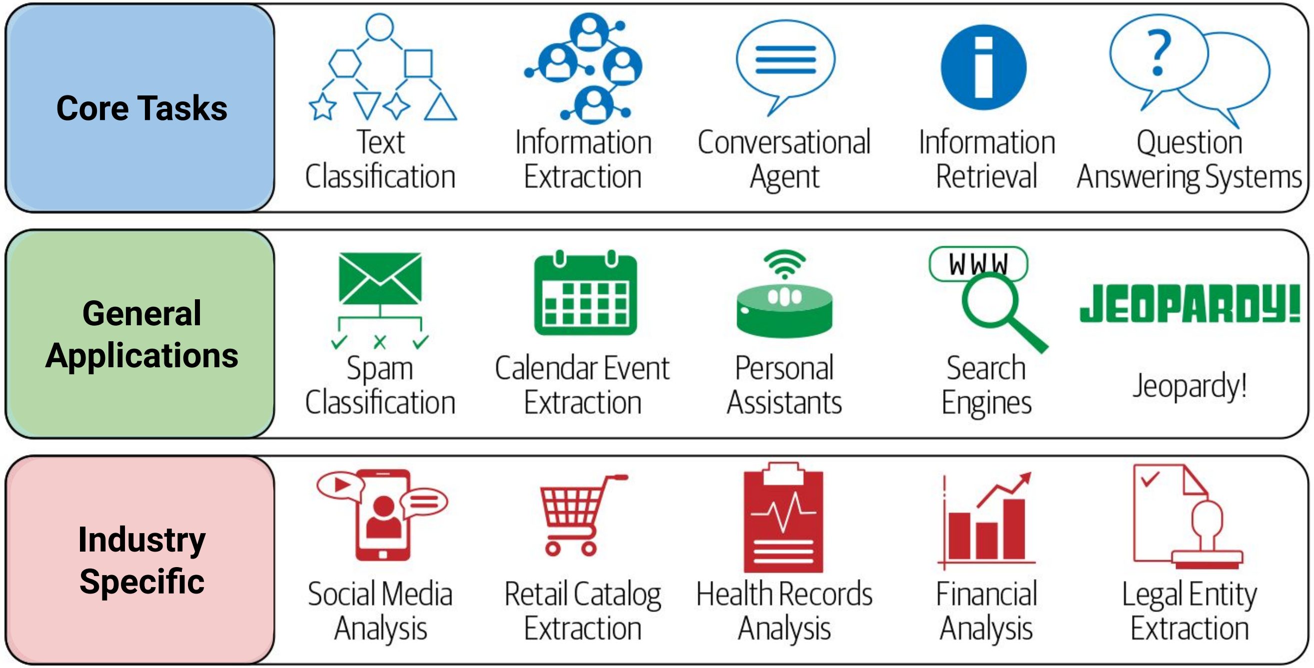
Lecture content taken from Stanford NLP with Deep Learning Course and other sources

See lectures plan HERE

Lectures and Announcements on Google classroom: *jgaxwor*

What is NLP?

Subfield of AI focused on the interaction between computers and humans through natural language



Word Representation - One-hot encodings

We want a good method to represent words as numbers to feed to our machine learning / deep learning model

One way to do it is one-hot encodings

I	am	going	to	school	learn	math
0	1	2	3	4	5	6

I am going to school -> *[1, 1, 1, 1, 1, 0, 0]*

I learn math -> *[1, 0, 0, 0, 0, 1, 1]*

Word Representation - TF-IDF

Term Frequency (TF): Frequency of a **term t** in a **document d**

Inverse Document Frequency (IDF): Inversely proportional to the number of documents that contain the **term t**

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

raw count of a term t in a document d

total number of terms in document d

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

total number of documents in the corpus

number of documents where the term t appears (plus one to both the nominator and denominator to prevent division by zero)

$$\boxed{\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)}$$

TF-IDF Example

Document 1: my cat is really cute

Document 2: my dog is really big

Document 3: i really love my dad

Can you calculate the TFIDF of all the terms in all there documents?

Question

What are the disadvantages of using TF-IDF and one-hot encodings?

One-hot encoding and TF-IDF issues

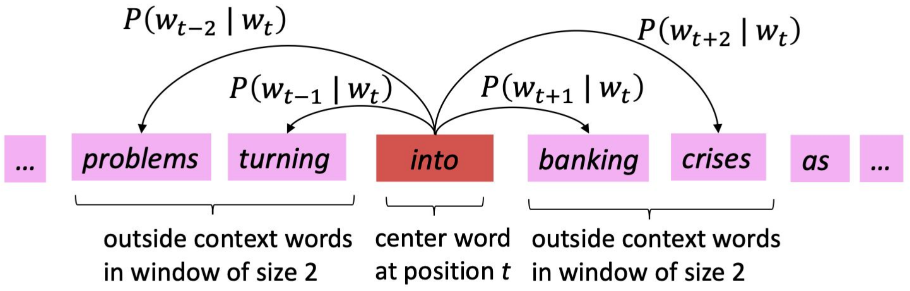
One-hot encodings and TF-IDF do not address these issues:

- Big vocabulary -> Big vector dimension
- Context does not matter
- Different terms lead to different vectors -> What about synonyms?

Word2vec overview

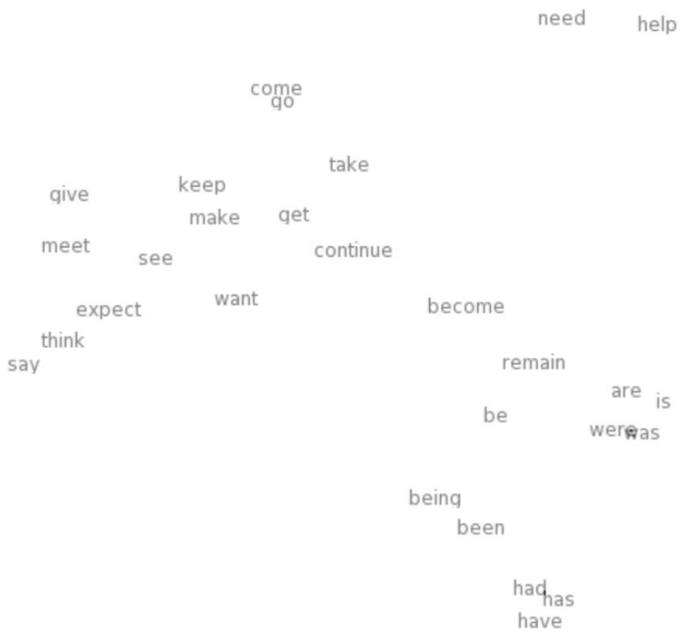
Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability



Word Embeddings Visualization

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



Coding Exercise

- Implement TF-IDF without using any TF-IDF library
- Download word2vec using gensim library and play around with the library

Sample code:

```
import gensim.downloader as api

model = api.load("word2vec-google-news-300")
model.most_similar("cat")
```

Advanced exercise: Apply PCA to the word2vec you just downloaded and visualize it