NIHR | Applied Research Collaboration South West Peninsula

@penCHORD_UoE
@peninsula_ARC

Module 8 : Natural Language Processing
Session 8B : Named Entity Recognition and Word Clouds
Dr Daniel Chalk

"Name that Plume"

HSMA 5

#hsma5isalive

SpaCy

In this session, we're going to be using a Python package called SpaCy (https://spacy.io/).

SpaCy is a **Free and Open Source** Python library for Natural Language Processing tasks. One of the tasks that it can achieve is Named Entity Recognition.

Whilst you don't need to understand the details of how SpaCy works to use most of its functions, we will give you a broad overview here.

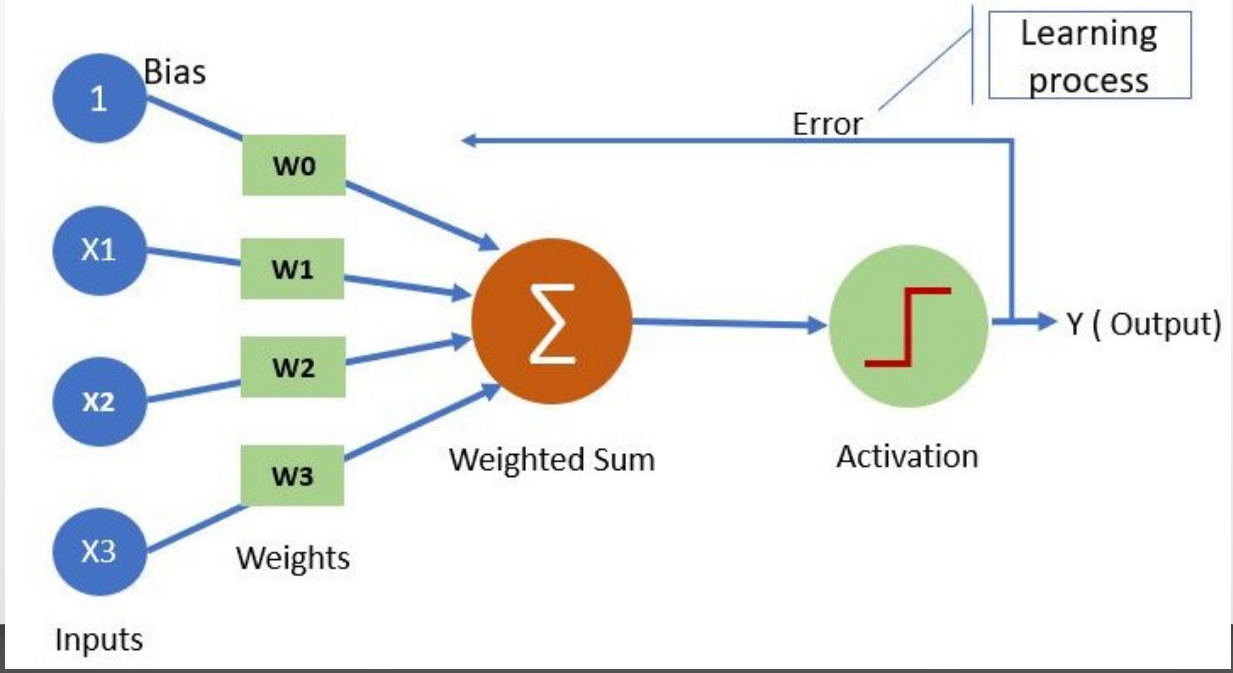
SpaCy used to use something known as a **Convolutional Neural Network**, but since Version 3, uses a **Transformer-Based Architecture.**

Before we try to explain that, let's just have a brief recap of Neural Networks from our Module 7 content.

The Neuron

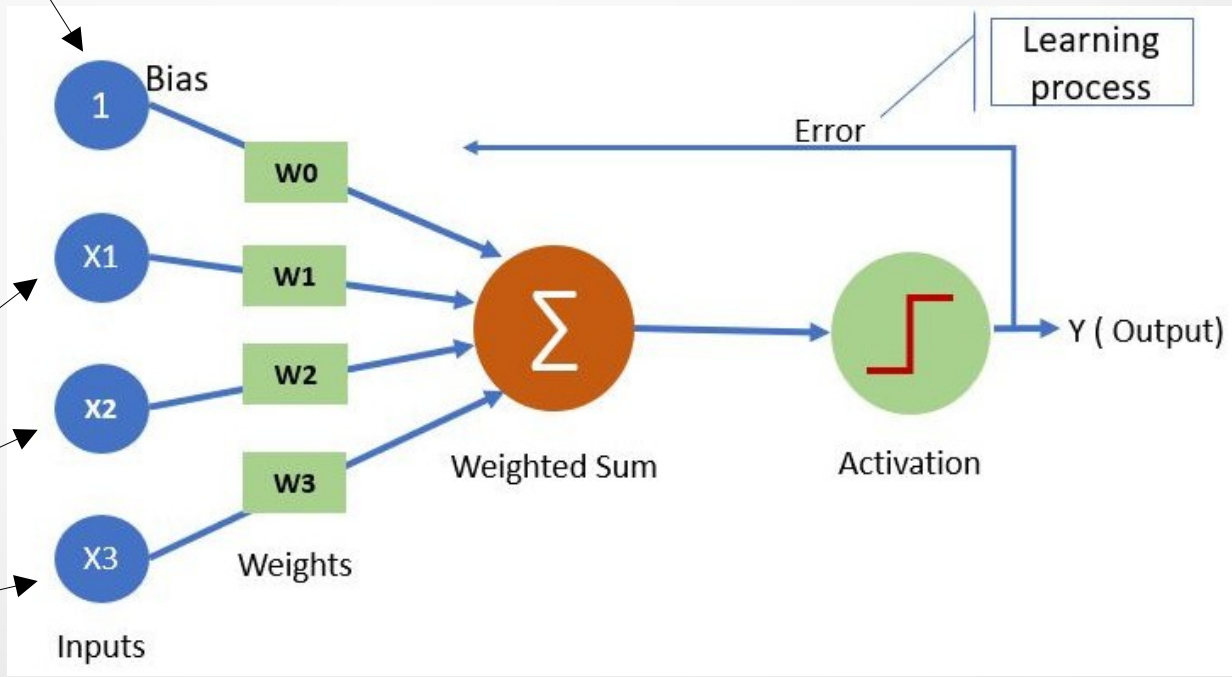
Neural Networks are made up of Neurons (originally named *Perceptrons*). Each Neuron performs a very simple task – to take all of its inputs (each multiplied by a *weight* representing the strength of a connection), add them all up, and spit them out according to an *Activation Function.*

The Activation Function takes the raw weighted sum, and converts it in some way.



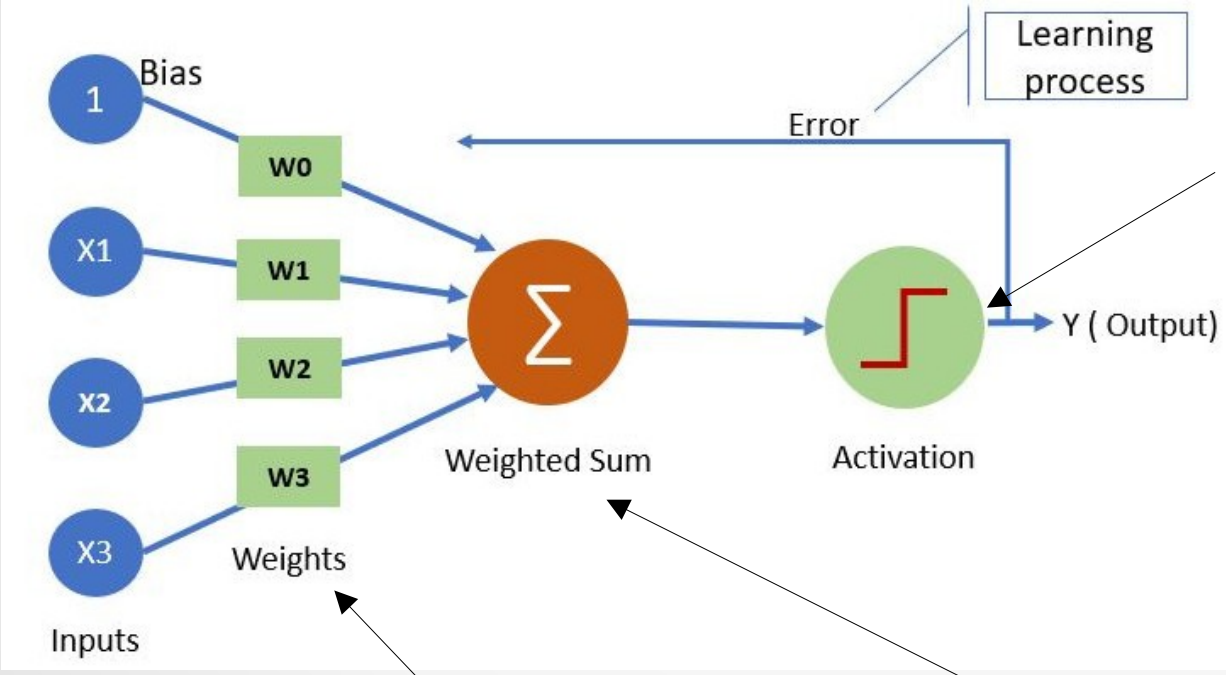
The Neuron

We have one input as a “Bias”. This is a bit like a constant in a linear function. It allows us to “shift” the activation function, which may help us learn better (e.g. if we shifted our activation function graph over to the left a bit, or the right a bit, it might be a better fit. The network will do this automatically – by including it as an input with a weight, the network can play around with it automatically to improve learning performance.



These represent values from our *features* (data in our “columns” – the information from which we’re trying to make a prediction)

The Neuron



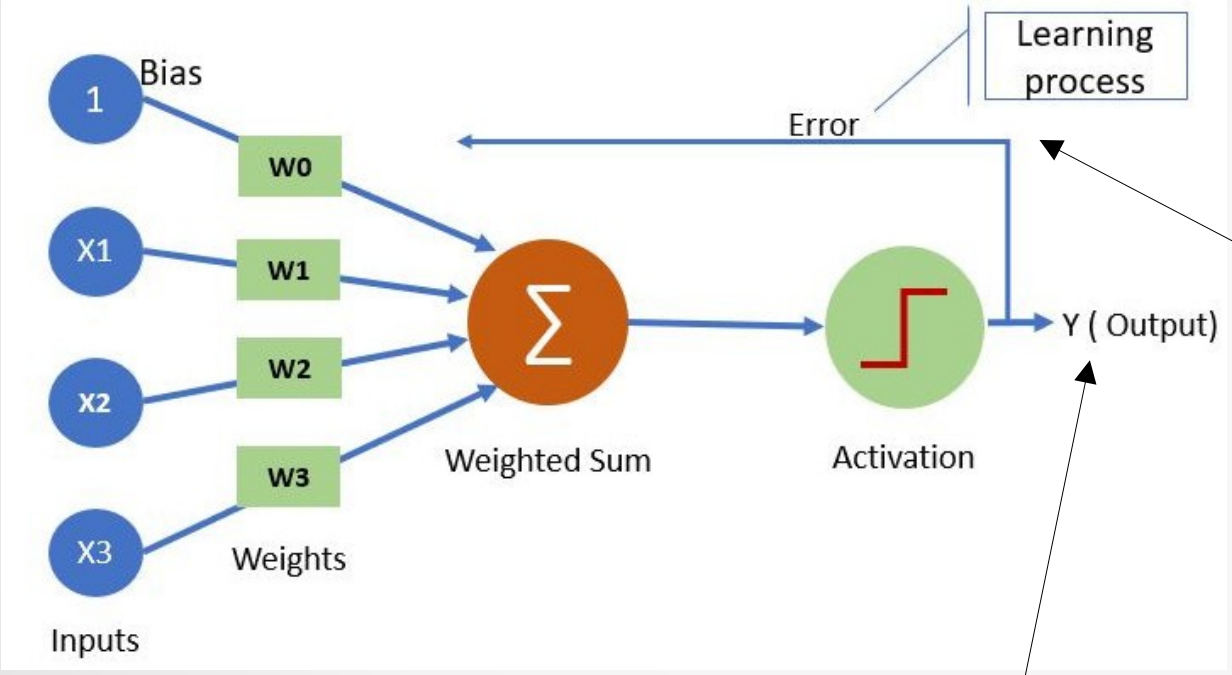
Once the neuron has performed its weighted sum of inputs, it fires it out via an “Activation Function”. This function transforms the output in some way (we’ll talk about that more shortly)

The original activation function was a “Step Function”, in which the output was either 0 or 1, depending on whether or not the weighted sum was higher or lower than a threshold value. This mimics a neuron in the brain being active or inactive.

Inputs are connected to a neuron along with a “weight”. These weights represent the relative importance of that input to the final calculation made by the neuron. The network plays around with these weights whilst it’s learning. Weights start with completely *random* values.

The neuron performs a “weighted sum” – it takes each input multiplied by its weight, and then adds up all those numbers

The Neuron



Key to learning in a Neural Network is “Error” – how much was the output (prediction) right or wrong? The “learning” of a Neural Network takes place as the network adjusts the weights to try to get the output prediction closer to the true value (in other words, minimise the error).

The Activation Function spits out an output (which is the weighted sum having been fed through the Activation Function). This output represents a “prediction”. This might be a classification prediction (is it this thing or this thing? - 0 or 1) or a probability (e.g. it’s 0.7 likely it’s this thing)

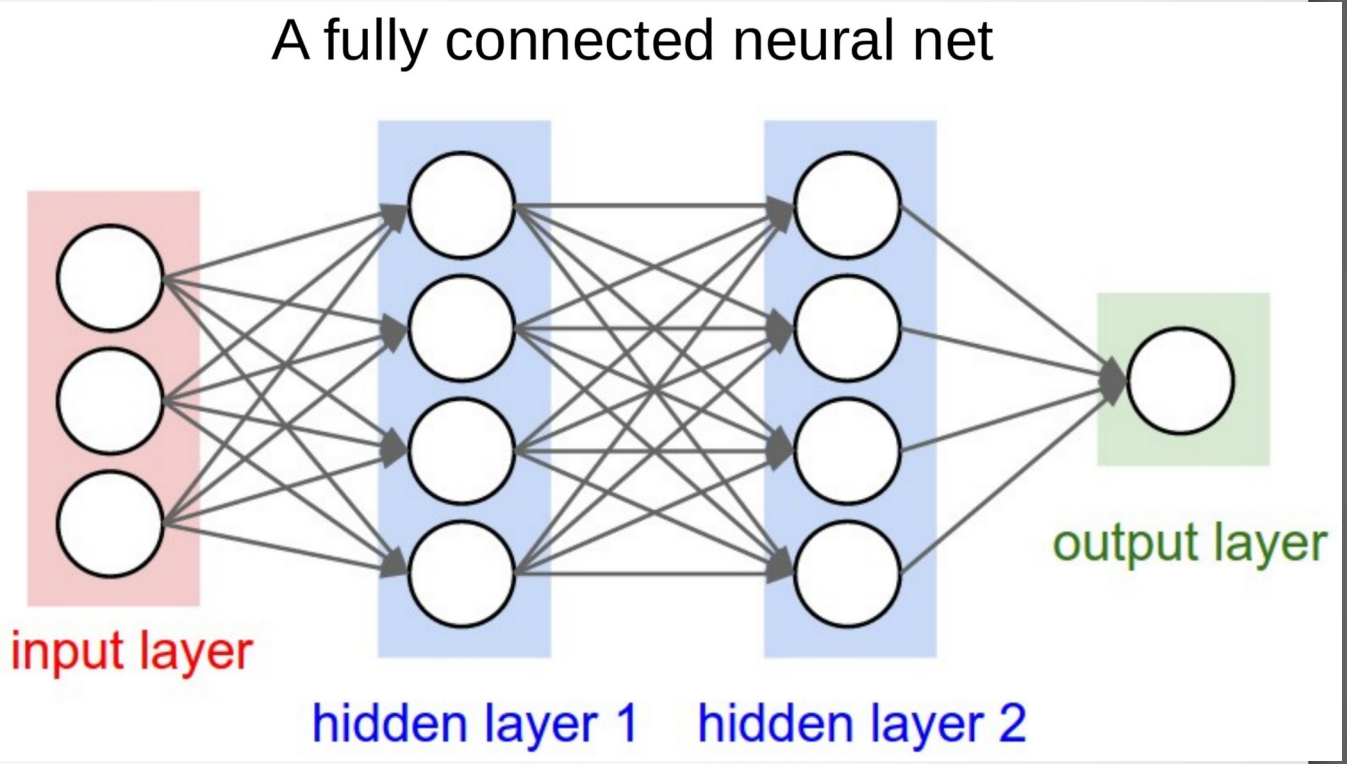
A Neural Network

A *Neural Network* is simply a network of *neurons* that are connected together across different *layers*. The outputs from neurons in one layer feed into neurons in the next layer (acting as their inputs).

A *fully connected* neural network is one in which every neuron in one layer is connected to every neuron in the next layer.

Each connection in the network has its own weight.

The network is trying to learn how to adjust the weights *across* the network so that it predicts the correct output for a given set of inputs, most of the time.



The multiple layers of a Neural Network give the name to the sub-field of AI in which they sit : **Deep Learning**

Convolutional Neural Networks

In the older version of SpaCy, it used something known as a **Convolutional Neural Network (CNN)**. This is a type of Neural Network that is used commonly in image processing and Natural Language Processing.

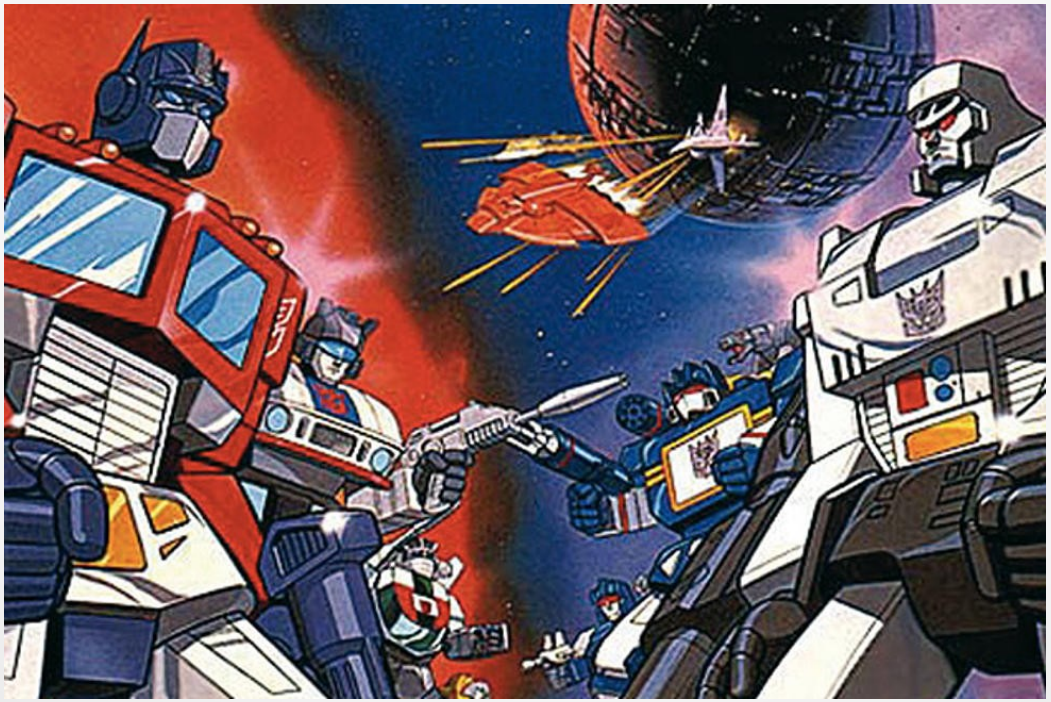
A CNN basically works by looking at only a subset of the data at any one time, and then gradually “sliding” the “window” that it looks at over the data, a bit a time.

In image recognition, this enables it to learn to identify if a cluster of pixels represents an object to identify.

In Natural Language Processing, this enables it to look at a subset of words in the sentence to see if this can help prediction. For Named Entity Recognition, this means trying to predict whether a word (or group of words) is a Named Entity or not based on the words around it, along with their POS tags, distance from the word(s) being assessed etc.

Transformers – Fun in Disguise

Now let’s talk about how SpaCy does thing now, and about something that has revolutionised Natural Language Processing – **Transformers**.



But before we do that, we need to understand the concept of **Attention** in Natural Language Processing.

Pay Attention!

Let’s study the following paragraph :

My cat is called Prince, and he’s very demanding. Every day, he asks for treats at 1100 (his “elevenses”). You could set your watch by him, so accurate is he in asking for them on time. Sometimes, when I’m teaching, he has to wait for them, and he gives me a look that he’s clearly not happy about this.

Ok now I’m going to ask you some questions.

Pay Attention!

“You could set your watch by him”

Who could you set your watch by?

Pay Attention!

“he gives me a look that he’s clearly not happy about this”

Not happy about what?

Pay Attention!

“so accurate is he in asking for them on time”

Asking for what?

Pay Attention!

As human beings, when we read things, we relate the things we read back to stuff we read earlier in the text :

My cat is called Prince, and he's very demanding. Every day, he asks for treats at 1100 (his "elevenses"). You could set your watch by him, so accurate is he in asking for them on time. Sometimes, when I'm teaching, he has to wait for them, and he gives me a look that he's clearly not happy about this.

Pay Attention!

But the Convolutional Neural Network approach doesn't allow for this. It looks at a bit of the text at a time – so the closeness of words to the thing you are trying to classify / predict becomes important.

Arguably, it becomes *too* important, and it becomes poor at identifying *long-range dependencies* (words that relate back to things much earlier in the text).

An alternative approach is to use **Attention**.

Pay Attention!

With Attention, we can put portions of text against each in a matrix, and represent the relationships of each word with each other.

This is useful for translation :

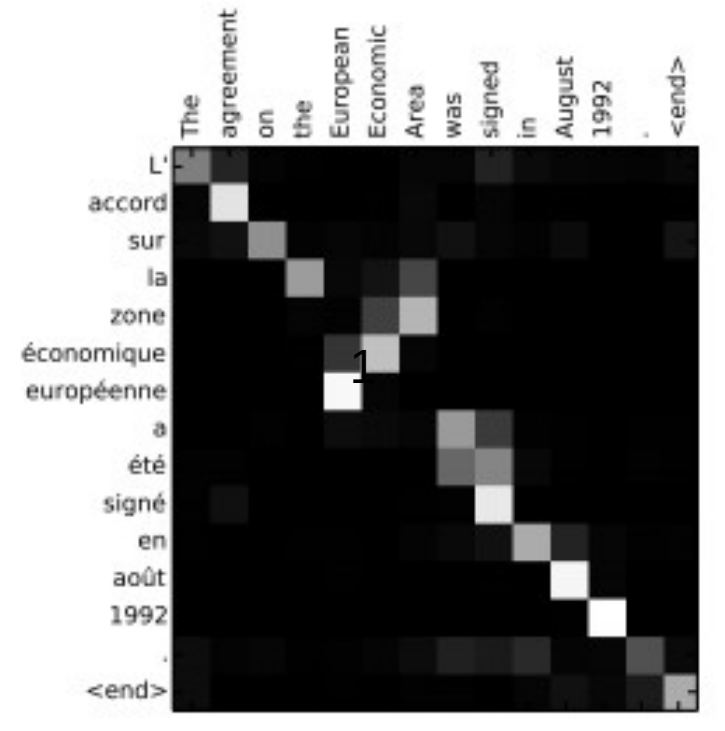


Image from https://wiki.pathmind.com/attention-mechanism-memory-network

But we can also put the same portion of text on each axis, and represent the relationships of words with other words in the text (*Self-attention)*. This emulates what we're doing as humans when we read some text.

Word Meanings and Context

Consider the following sentence :

“John asked Mary if he could borrow her pen.”

As human beings, it’s likely we’ll assume that the thing John wants to borrow from Mary is a writing implement, rather than a female swan, or an animal enclosure, even though the same word (“pen”) would be used in all of these cases. Sometimes some additional text can help us understand the context :

“This is because John was busy writing his memoirs.”

Word Meanings and Context

Consider the following two sentences :

“John was found eventually after hiding for some time from Mary.”

“John had got used to hiding his true feelings for Mary.”

In both sentences, the same word – “hiding” is used – but in different contexts. The first describes John physically hiding from Mary. The second refers to John concealing something intangible from Mary. The same word is used “hiding” with the same POS Tag (verb) but the context helps us understand the difference.

Word Meanings and Context

In natural language, the same word can be used in different places in the same text to mean very different things. The thing that helps us understand is the **context** of the word in the sentence / wider text.

Clues within the text (and external to it – common phrases etc) can help us to understand the context in which a word (and therefore the sentence) should be interpreted.

Transformers

Transformers use *Attention* to try to understand the context of a word, and then *encode* that context-information alongside the word.

This means that, unlike a Convolutional Neural Network, which only looks at words in the context of words that surround them within a defined proximity, Transformers can pull information from *anywhere* in the text to try to understand the context of a word or words.

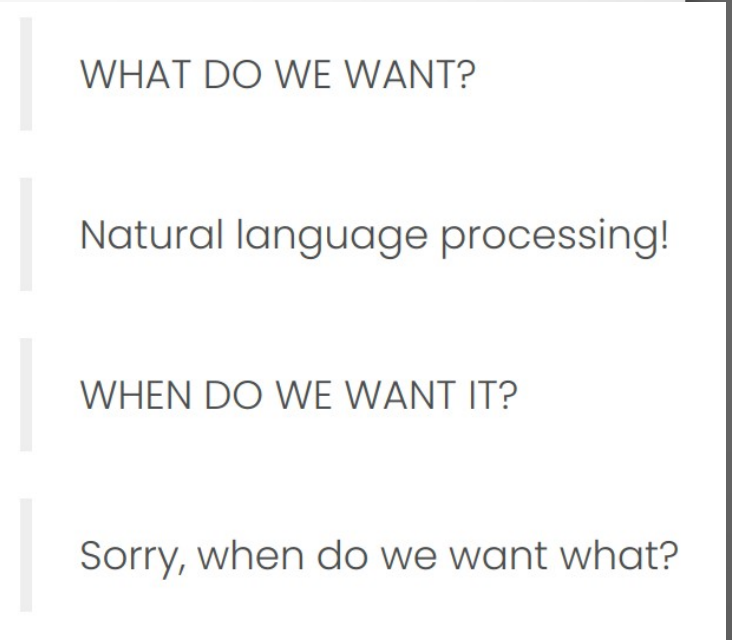
This makes them very powerful tools for NLP tasks, including POS-Tagging, identifying noun phrases and recognising things as *Named Entities*.

Transformers

There is *much* more to learn about Transformers, but for the purpose of understanding the gist of how SpaCy is “working its magic” this will suffice for the moment.

I’d highly recommend having a read of this very friendly introduction to Attention and Transformers in NLP (which I found very helpful in putting this material together) :

https://wiki.pathmind.com/attention-mechanism-memory-network



BERT

And if you want to read a bit about BERT (a transformer model that revolutionalised the world of NLP) and ELMo (which revolutionised context-embedding), there's a nice little intro here : https://jalammar.github.io/illustrated-bert/

OpenAI's GPT-2 and GPT-3 models are also famous and very powerful Transformer models : https://openai.com/api/

SpaCy Features

https://spacy.io/usage/spacy-101#features

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) **Tagging**	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of “was” is “be”, and the lemma of “rats” is “rat”.
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named “real-world” objects, like persons, companies or locations.
Entity Linking (EL)	Disambiguating textual entities to unique identifiers in a knowledge base.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.
Training	Updating and improving a statistical model’s predictions.
Serialization	Saving objects to files or byte strings.

SpaCy Process

When we're using SpaCy we typically have three main steps :

1. Load a pre-trained *model* (sometimes referred to as a *language*). It is often loaded into a variable called *nlp*.

2. Apply the pre-trained model (nlp) to some text to generate predictions of named entities, POS-tags, word dependencies etc. These predictions are stored in a SpaCy structure called a *Doc Object.* The Doc object is often named *article* or *doc*.

3. Explore the predictions stored in the Doc object, and use displaCy (SpaCy's visualiser) to visualise them.

Let’s try SpaCy

So, now we know how it works, let’s try out SpaCy!

Make sure you’ve installed SpaCy before proceeding :

pip install spacy

and you also need to download and install one of the learning models (pre-trained SpaCy model that’s been trained on general text) by going into your Command Prompt or Terminal and typing :

python -m spacy download en_core_web_sm

Let’s try SpaCy

You’ve been provided with a file called “TrumpArticle.txt”, which is text from a New York Times news article about the firing of an FBI agent.

We’re going to use SpaCy to extract the named entities (and their predicted classifications) from this news story.

Let’s open up the file trump_extract.py, which is the code that will extract the named entities from the .txt file. Make sure both the .py file and the .txt file are in the same directory.

trump_extract.py

```
# import spacy and the downloaded en_core_web_sm pre-trained model
import spacy
import en_core_web_sm

# Load the pre-trained model as a language model into a variable called nlp
nlp = en_core_web_sm.load()

# Read in the document for which we want to extract named entities
with open("TrumpArticle.txt", encoding='utf8') as f:
    # Read in the whole file as a single string but strip out any whitespace at
    # the start and end of the file
    raw_read = f.read().strip()

# Apply the pre-trained model to the raw text string to extract named entities
# All of SpaCy's predictions (including the named entity predictions) are
# stored as a Doc object.  Here, we call our Doc object 'article', but you
# may also see it commonly named 'doc'.
article = nlp(raw_read)

# Store the Named Entity categories (stored in label_ for each entity) in the
# article in a list. The named entities themselves are stored in article.ents
labels = [x.label_ for x in article.ents]

# Print each predicted named entity, along with its predicted category
for i in range(len(article.ents)):
    print (f"{article.ents[i]} : {labels[i]}")
```

Running the code

```
In [3]: runfile('/home/dan/Dropbox/HSMA 3/phase_1_training/11
WASHINGTON : GPE
Peter Strzok : PERSON
F.B.I. : ORG
Trump : PERSON
Hillary Clinton : PERSON
Russia : GPE
Strzok : PERSON
Monday : DATE
Trump : PERSON
2016 : DATE
F.B.I. : ORG
Lisa Page : PERSON
Russia : GPE
Strzok : PERSON
20 years : DATE
F.B.I. : ORG
the early months : DATE
Strzok : PERSON
F.B.I. : ORG
Trump : PERSON
Strzok : PERSON
last summer : DATE
Robert S. Mueller III : PERSON
Strzok : PERSON
Twitter : ORG
Monday : DATE
Trump : PERSON
June : DATE
Strzok : PERSON
F.B.I. : ORG
Hillary Clinton : PERSON
2016 : DATE
Strzok : PERSON
the bureau's Office of Professional Responsibility : ORG
Strzok : PERSON
60 days : DATE
Strzok : PERSON
House : ORG
July : DATE
Strzok : PERSON
F.B.I. : ORG
David Bowdich : PERSON
the Office of Professional Responsibility : ORG
Strzok : PERSON
F.B.I. : ORG
Strzok : PERSON
Trump : PERSON
F.B.I. : ORG
Bowdich : PERSON
F.B.I. : ORG
Christopher A. Wray : PERSON
Aitan Goelman : PERSON
Strzok : PERSON
Special Agent Strzok : ORG
Wray : PERSON
Congress : ORG
F.B.I. : ORG
Goelman : PERSON
Americans : NORP
Goelman : PERSON
Strzok : PERSON
Page : PERSON
```

displaCy

SpaCy comes with a fantastic visualiser called displaCy. You can use displaCy to easily visualise the SpaCy model's predictions of different aspects of the text – including the predictions of named entities.

To display named entity predictions using displaCy, we simply add this line of code anywhere after the pre-trained language model (which we named *nlp*) has been applied to generate the Doc object which stores the model predictions (which we named *article*).

```
spacy.displacy.serve(article, style="ent")
```

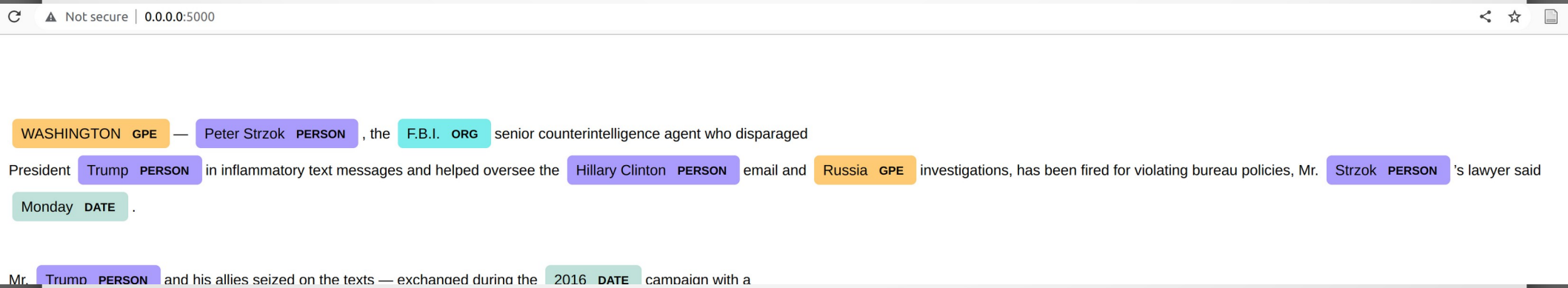
Name of the Doc object

What we want to visualise ("ent" gives us the named entity predictions)

displaCy

When we run the code, SpaCy will spin up a local server and tell us the address we need to paste into our browser in the iPython console (note this only applies to an IDE such as Spyder – if we're using Jupyter or CoLab it's different, see slides in a moment) :

```
Using the 'ent' visualizer
Serving on http://0.0.0.0:5000 ...
```



displaCy for Jupyter / CoLab

If we're using Jupyter or CoLab, the previous code will work, but we don't actually need to spin up a server (as we're already using one for Jupyter / CoLab). Instead, we can just use :

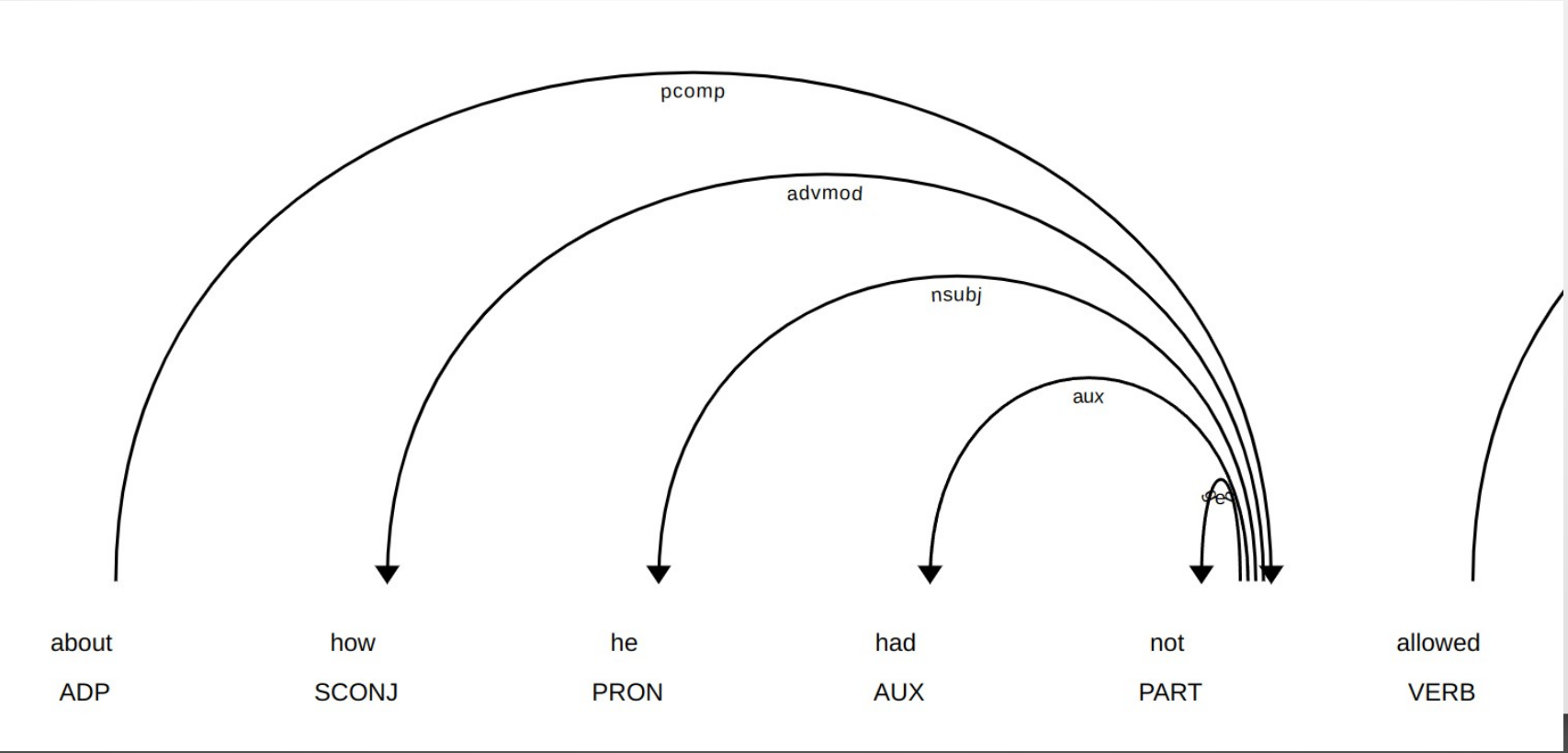
```
spacy.displacy.render(article, style="ent")
```

Either way, you'll see the visualisation will appear in the output below the code cell (if you use .serve in Jupyter / CoLab you'll just get a warning message too saying you don't need to do that).

Other displaCy visualisations

There are other visualisations that you can use in displaCy. For example, we can view the syntactic dependencies that have been parsed in a piece of text :

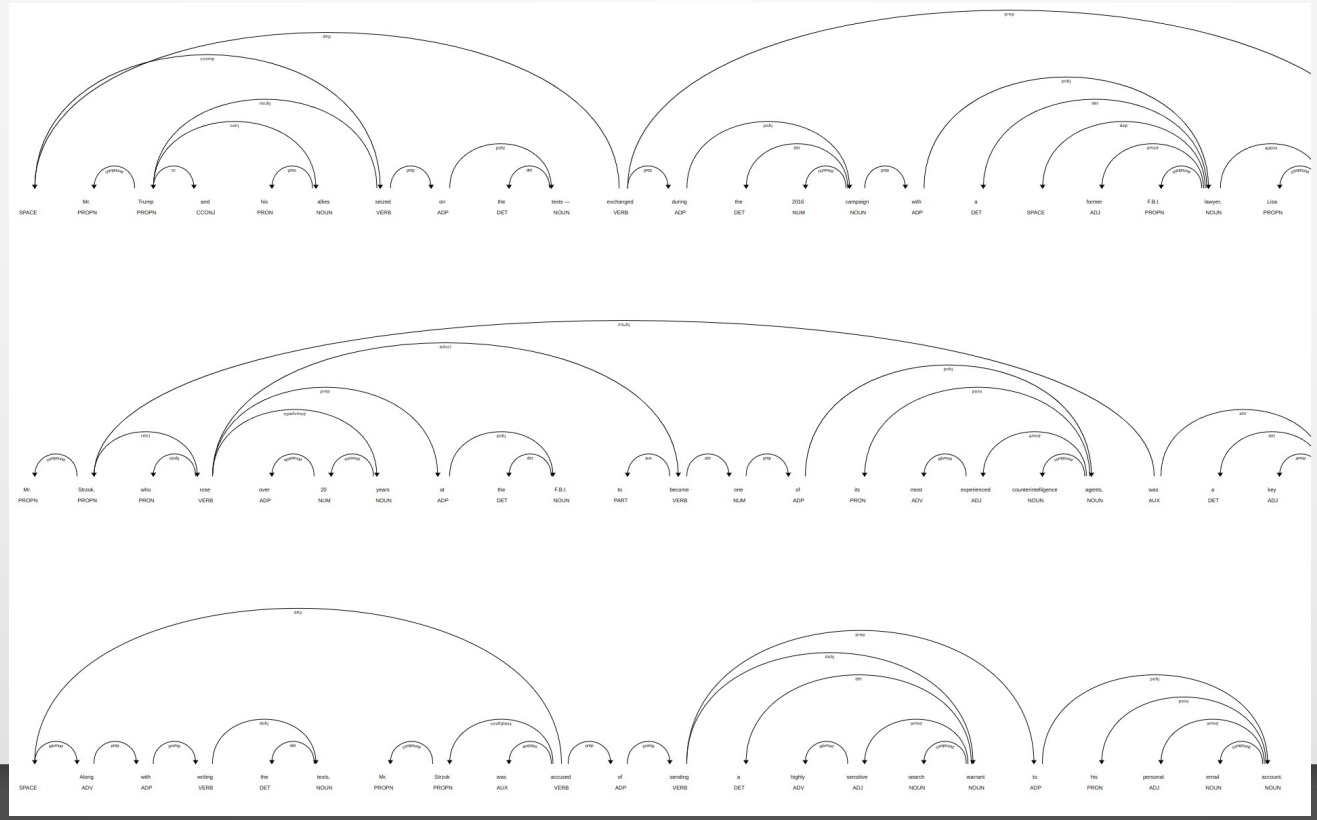
```
spacy.displacy.serve(article, style="dep")
```



Other displaCy visualisations

Or we can break down visualisations into sentences to make things easier to interpret :

```
sentences = list(article.sents)
spacy.displacy.serve(sentences, style="dep")
```



Other displaCy visualisations

You can read more about all the great things you can do with displaCy here : https://spacy.io/usage/visualizers

Exercise 1

For this first exercise, I want you to spend the next 20 minutes (+ 5 minute break) having a play around with what I've shown you in SpaCy. You could :

- run the code trump_extract.py and look at the predicted named entities and their categorisations
- run the code trump_extract_2.py and look at the displaCy visualisation of the named entities, and other visualisations
- try running the code on some text of your choosing – just put the text into a .txt file in the same directory, and apply the model to that text in the code instead of the Trump article

Work in your groups and discuss what you're exploring.

Updating Training with New Examples

Whilst the pre-trained models in SpaCy are good, they won't recognise everything. There will be words and syntactic patterns that it hasn't seen before in the training corpuses. There are two things that can be done to improve accuracy :

1) Select a pre-trained model that best suits the text from which you are trying to extract information. There are three core models in English on the spacy website that have been trained on increasingly larger corpuses (https://spacy.io/models/en) (sm = small, md = medium, lg = large).

The core models are trained on a range of general texts such as blogs, news and comments. Other models are available in the wild that have been trained in more specialist areas, such as tweets on Twitter or medical papers.

Updating Training with New Examples

2) We can add new examples to a pre-trained model so that it learns to identify new words and syntactic patterns that it hasn't seen in the past to help it improve.

Generally, you need lots of new examples to meaningfully improve things (ideally at least a few hundred).

It can also be quite difficult to update SpaCy training. But if you want to see how to do it, there are instructions here :

https://spacy.io/usage/training#training-data

Word Clouds

When we're undertaking NLP tasks such as Named Entity Recognition and Sentiment Analysis, it's often useful to get a visualisation of our text to understand some of the frequent things that are being said.

Word Clouds are a really nice visualisation technique that allow us to do exactly that.

In Python, we can use the WordCloud package to generate word clouds easily. To install WordCloud, we use :

pip install wordcloud

Word Clouds

Let’s have a look at how we can use the WordCloud package to generate word clouds. The code I’m going to be demonstrating is wordcloud_example.py, and we’re going to generate a word cloud from a random IMDB movie review, stored in the text file wordcloud_example_text.txt.

But before we do that, I’m just going to talk about a few important concepts in Natural Language Processing.

Stopwords

For many Natural Language Processing tasks, such as Sentiment Analysis and generating word clouds, we're not usually going to be interested in common words that are used in all text – words like "and", "you're", "the", "because" etc. (Although these are useful for Named Entity Recognition, because they help the model understand context etc)

In NLP, these kinds of words are referred to as *Stopwords*. When we generate a word cloud, we want to remove these common words, as they're not of interest.

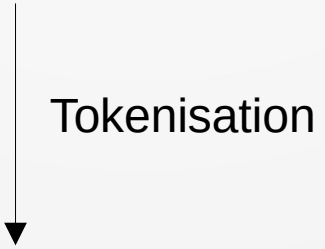
Fortunately, we can use pre-built lists of them to eliminate them from our analysis easily.

Tokenisation

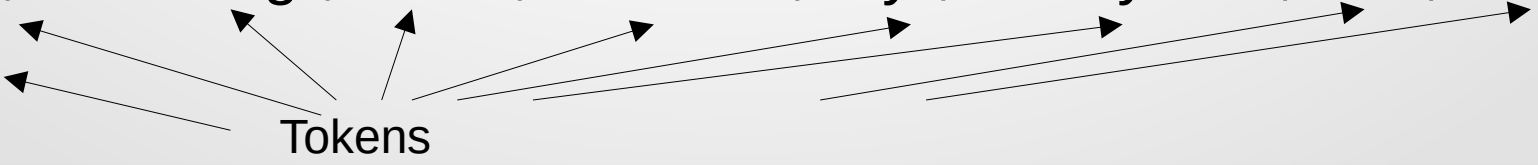
Another common thing that we need to do when processing natural language is to split text into smaller sub-components (usually individual words, but may sometimes be things like morphemes or even characters) and put them in a sequence, like a list.

In NLP this process is known as *Tokenisation* and the resultant components are known as *tokens*.

Dan’s teaching was viewed by everyone as first class!



[“Dan’s”, “teaching”, “was”, “viewed”, “by”, “everyone”, “as”, “first”, “class!”]



Translation Tables

The Python *string* class has a method named maketrans() which allows you to create a mapping table to map one character to another in some text. There are three ways you can use it :

1. If you pass in just one input (argument), then it must be a dictionary where each key value pair represents the character and how it should be translated

```
trans_dict = {"a":"1", "!":","}
str.maketrans(trans_dict)
```

2. If you pass in two inputs (arguments), then it must be two strings of equal length, where the *i*th character of the first string is mapped to the *i*th character of the second string

```
str.maketrans("a!", "1,")
```

3. If you pass in three inputs, it works like 2) above, but every character in the passed in third string represents characters to be removed.

```
str.maketrans("a!", "1,", "#")
```

In any case...

In NLP, the case of text matters! Therefore the strings “Cheese” and “cheese” (and even “chEeSe”) are treated as *different* strings that are **not** equivalent.

For tasks like word clouds and sentiment analysis, we want to consider the same word in the same way, regardless of case.

Therefore, we typically convert *everything* to lowercase, so there’s no longer any difference in case between words that are otherwise the same.

Wordcloud Demo

Let’s now look at the review text from which we want to generate a word cloud (wordcloud_example_text.txt) and then let’s walk through the code to do this (wordcloud_example.py).

Exercise 2

You have been provided with a file called exercise_2_article.txt. This file contains text from a news story featured on our website some time ago about some work that Sean and Kerry did with one of our HSMA alumni at Devon Partnership Trust.

In your groups, I want you to use the techniques I've shown you (Named Entity Recognition using SpaCy and Word Clouds) to generate some insights about this text. You should refine your approach as you go (for example, are there named entities that you are picking out that are less relevant and can be excluded? Are there types of words that are commonly being classified in one type of named entity that might be useful to know? What are some of the most common named entities and common words (excluding stopwords)? Can you optimise your word cloud and displaCy visualisations to be insightful but clear?).

You have 1 hour. I will ask a selection of groups to present what they've done, along with their findings / insights at the end of the exercise, so be sure to nominate someone to present. Imagine you are presenting the information to someone who has asked for this piece of analytical work to be undertaken.