NIHR | Applied Research Collaboration South West Peninsula

@penCHORD_UoE
@peninsula_ARC

Module 8 : Natural Language Processing
Session 8C : Sentiment Analysis
Dr Daniel Chalk

"How Appropriate. You fight like a cow."

HSMA 5

#hsma5isalive

Sentiment Analysis

Important :

You must download the Large Movie Review Dataset v1.0 available here :

https://ai.stanford.edu/%7Eamaas/data/sentiment/

and extract to the same directory as your .py files for this session

You must remove the "Unsup" folder in the "train" directory. This is data that can be used for *unsupervised learning*, but we won't use this here, and if you leave it in, the code will try to use these examples too and you'll get very poor performance from your algorithm

Sentiment Analysis - Recap

Sentiment Analysis uses AI-based methods to try to automatically identify the ‘sentiment’ / ‘tone’ of a piece of text, to identify if it is positive, negative or neutral.

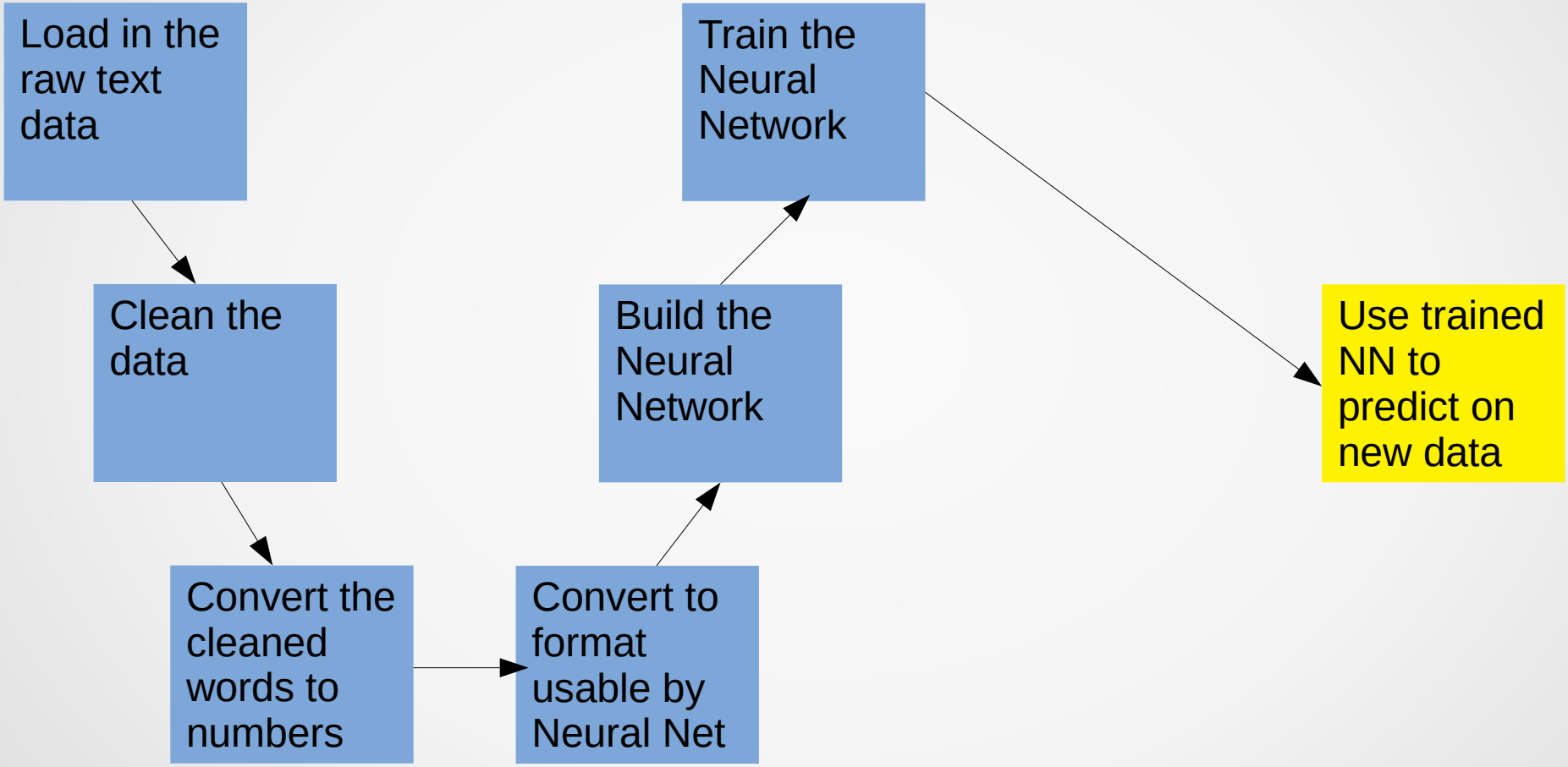
In a similar way to the way in which SpaCy uses AI to predict whether a word (or group of words) is a Named Entity, here we are predicting whether a piece of text is positive (1), or negative (0). In some cases, we may also assess sentiment as being positive (1), negative (-1) or neutral (0).

Applications of Sentiment Analysis

Potential Applications :

- automatically classifying service user survey data
- analysing reviews of movies, books etc
- analysing social media posts, and flagging up negative comments to be addressed
- looking for positive or negative references to your organisation on websites etc

Sentiment Analysis – The Overall Process



Clean the Data

When we undertake sentiment analysis, we need to carry out a number of steps first to process and clean up the data.

We don’t tend to do this with Named Entity Recognition, because things like the presence or absence of capital letters, and words like “and” and “the” might be important.

But when we’re trying to determine sentiment, we’re wanting to look at key words, and less about the syntactic structure of the text.

Therefore, there a number of things we normally want to do with our text before analysing the sentiment.

Clean the Data

1. Convert everything to lowercase. We don't care if someone wrote "Rubbish" or "rubbish" (or even "ruBbIsh") - for sentiment analysis, we want to consider them all as the same word.

2. Tokenize the text – break down text into list of individual words.

3. Remove punctuation (and, optionally, numbers) – these things are rarely useful for determining sentiment, although numbers might be, depending on the context.

4. Stem the words – find words with common stems and treat them all as the same word (e.g "watching", "watch", "watched" all have the same stem, and we want to treat them all the same way semantically for the purpose of determining sentiment)

5. Remove stopwords (such as "a", "the", "and" etc. They won't help us)

Convert cleaned words to numbers

Neural Networks (and computers in general) need to deal, at the basic level, in numbers, not text.

So we need to convert our words into numbers that can then be fed through our Neural Network and be numerically manipulated until combinations that should result in Positive reviews have an output of 1, and negative reviews have an output of 0, for example.

To do this, we represent words according to the inverse of their frequency in the text, so that the most frequent words have a number of 1. This allows us to easily chop out words that occur infrequently, from which learning will be difficult, and which are likely less important for learning anyway.

Convert to format for Neural Network

We need to convert the Neural Network inputs (now numbers) into a format acceptable to the Neural Network.

This will involve :

- Separating the data into “inputs” (the sequences of words (as numbers) making up each bit of text we want to classify) and “labels” (a 1 or 0 to indicate whether each input is positive or negative, for training).
- We need all inputs to be the same size, so we decide on a size (number of words), and truncate all longer inputs, whilst padding out all shorter inputs with a dummy word value of 0.
- Split the data we have into training, validation and testing sets, so the NN can learn, we can fine tune it, and then test out how well it might perform on wider data.

Build and Train the Neural Network

Essentially we :

- Build the Neural Network up by specifying one layer at a time
- Compile the Neural Network to assemble it

then :

- Specify how many “epochs” (learning time units) we want the NN to go through, and the “batch size”, which specifies how many inputs will be shunted through the network at once
- Start the model training
- Output metrics so we can monitor how well its learning (whilst looking out for potential issues, such as overfitting)

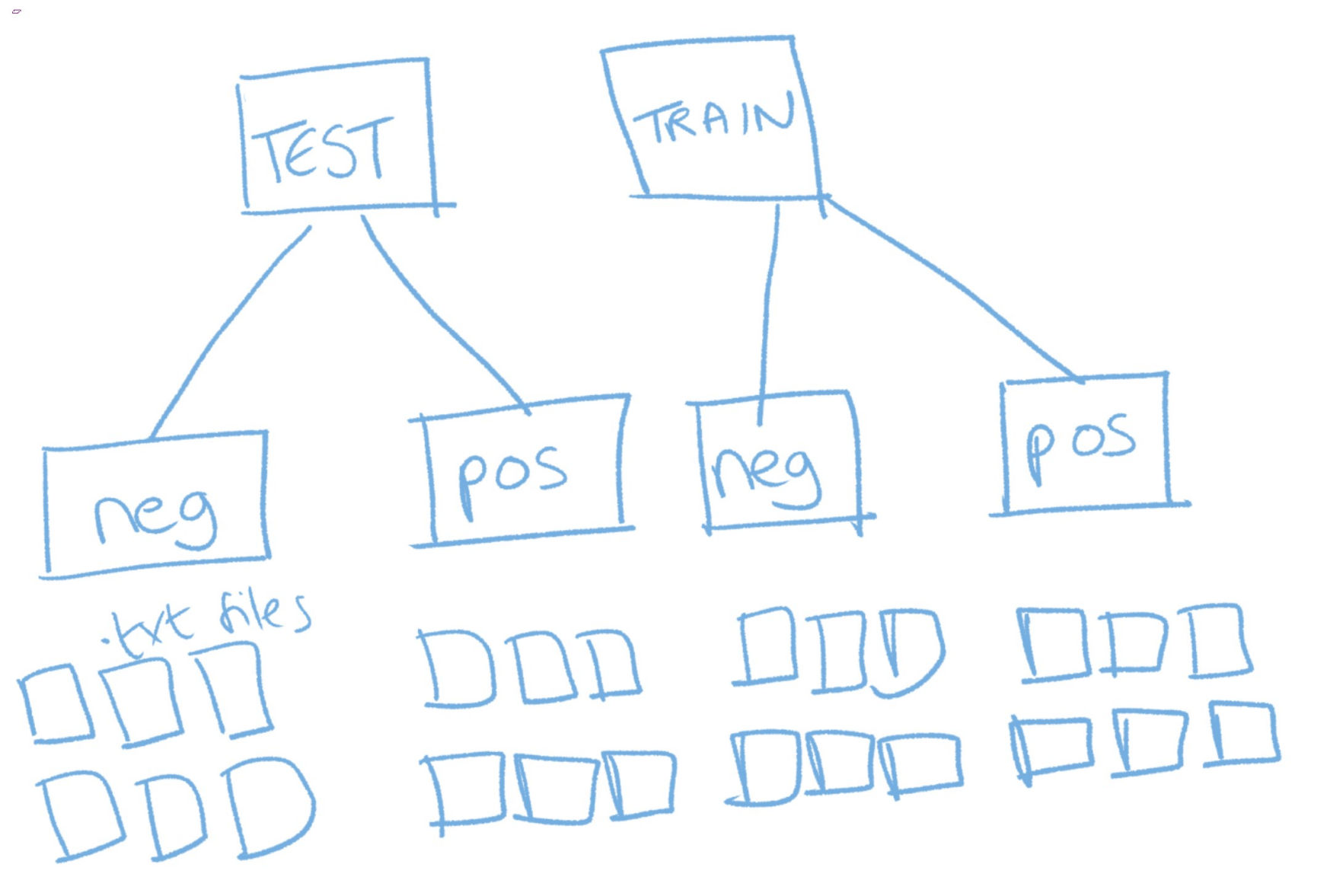
Analysing the Sentiment of Movie Reviews

Let’s look at an example of how we might use Sentiment Analysis to automatically classify movie reviews from the Internet Movie Database (IMDB) as “Positive” or “Negative”.
(sa_with_own_data.py)

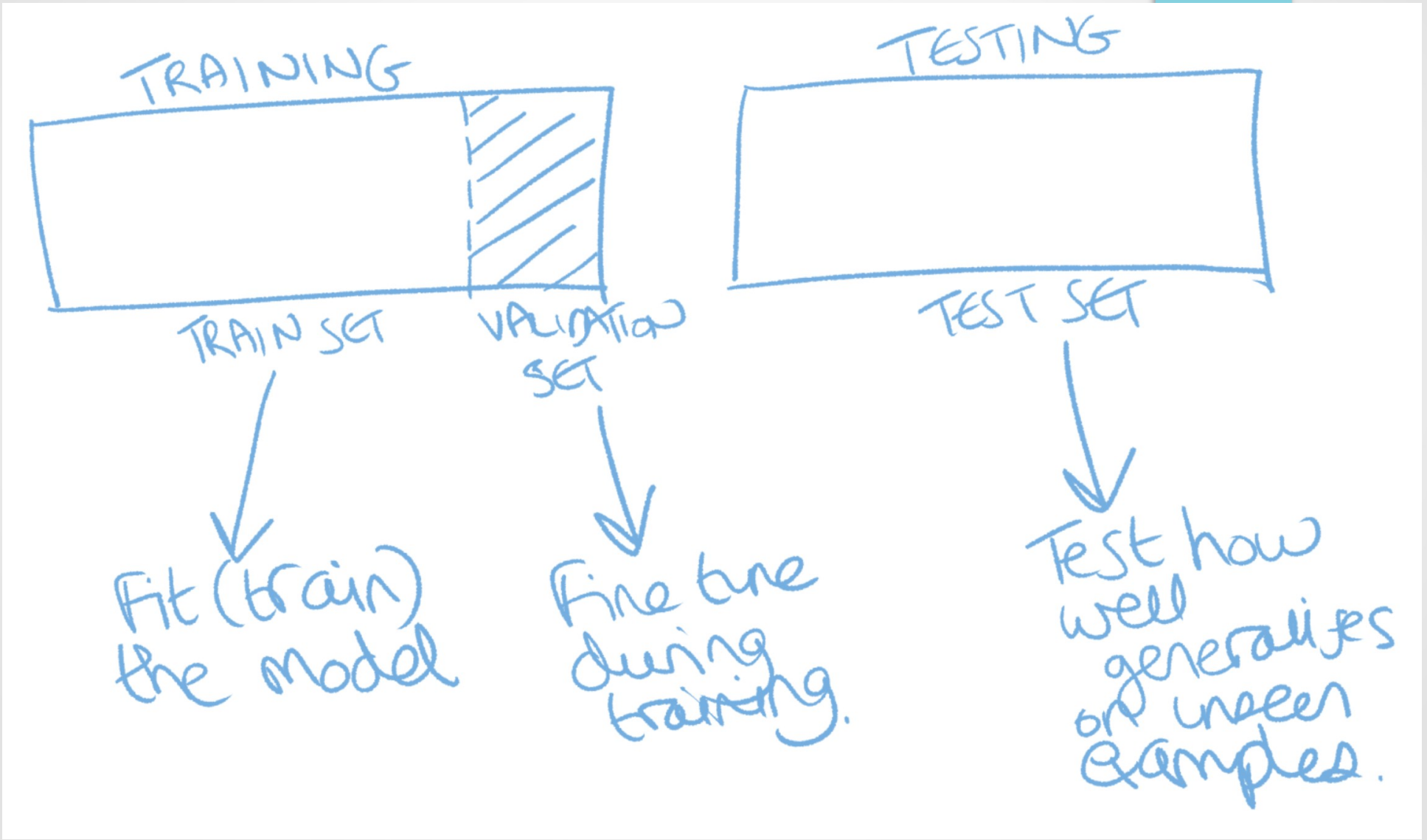
We’ll use *TensorFlow*, and the *Keras* API to access its features.

But first, let’s talk through what this code is going to do.

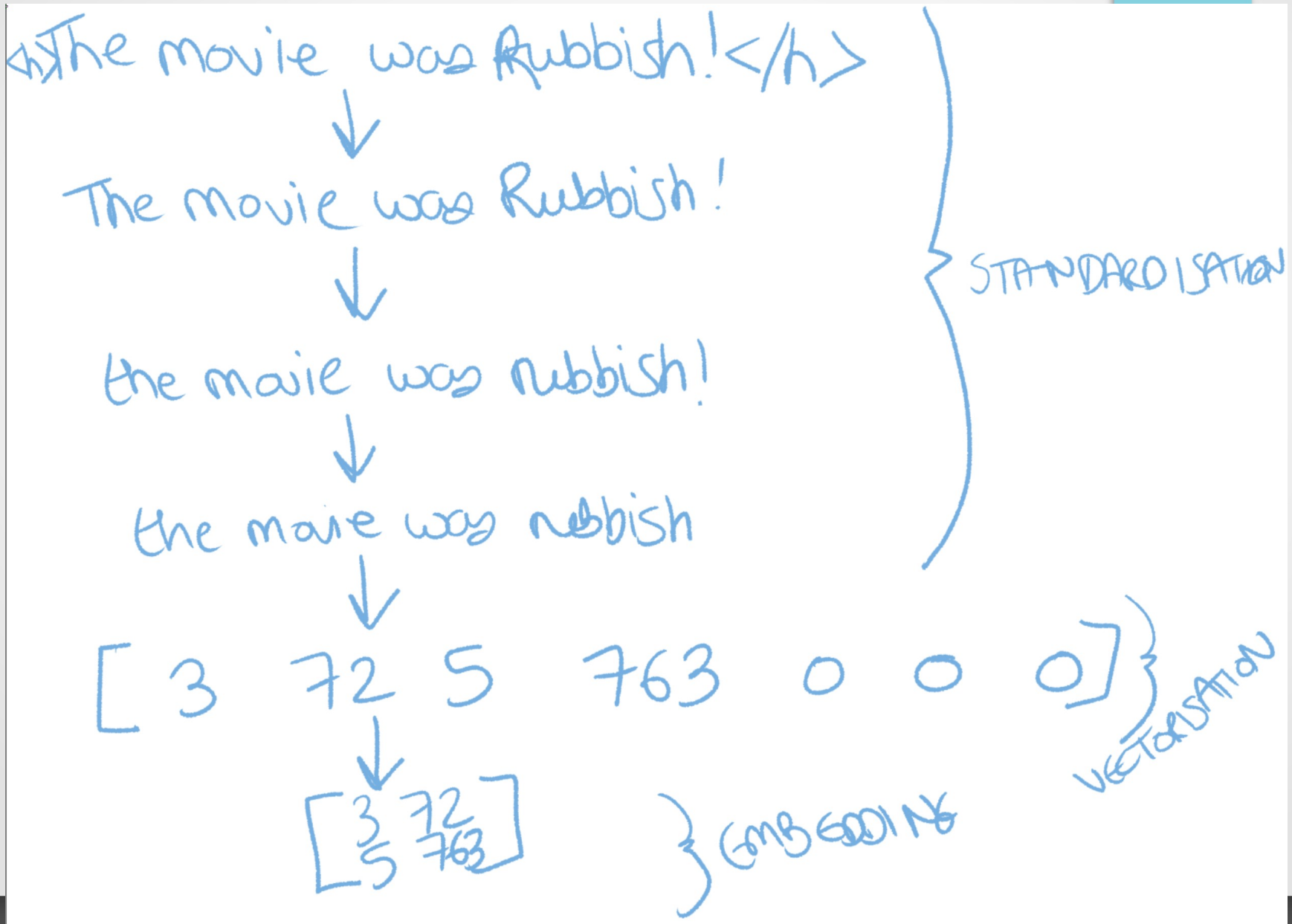
Directory structure for our examples



Training, Validation and Test Sets



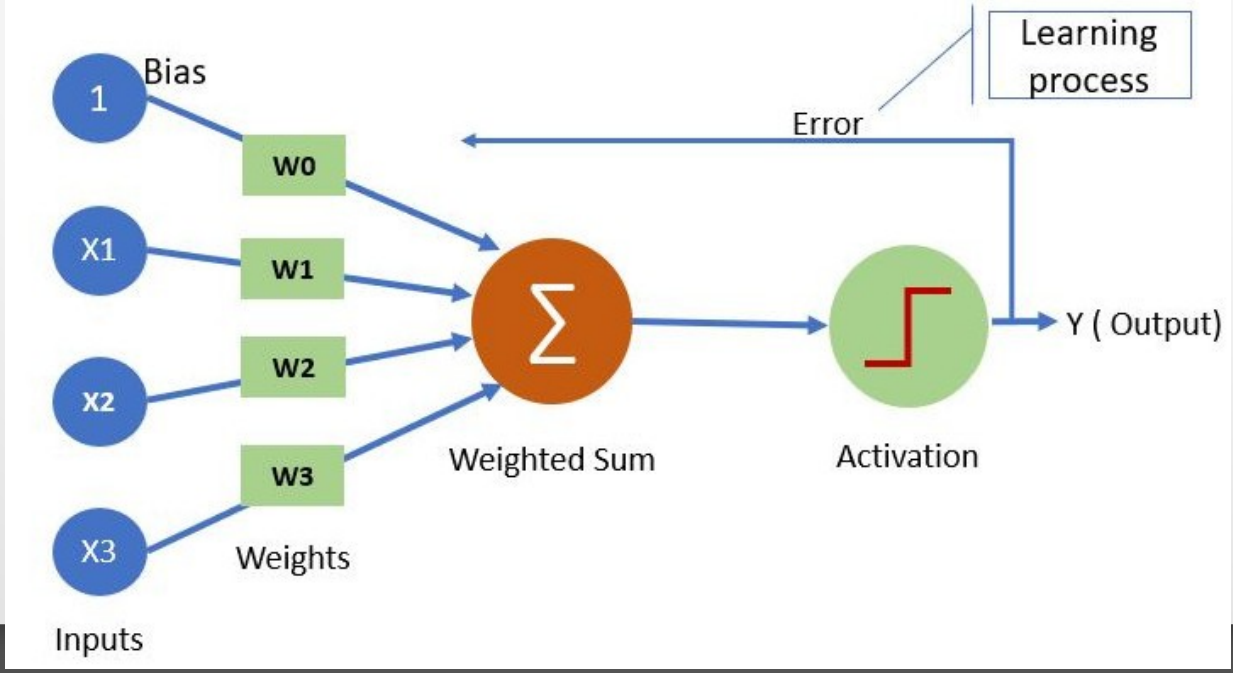
Standardisation, Vectorisation and Embedding



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.



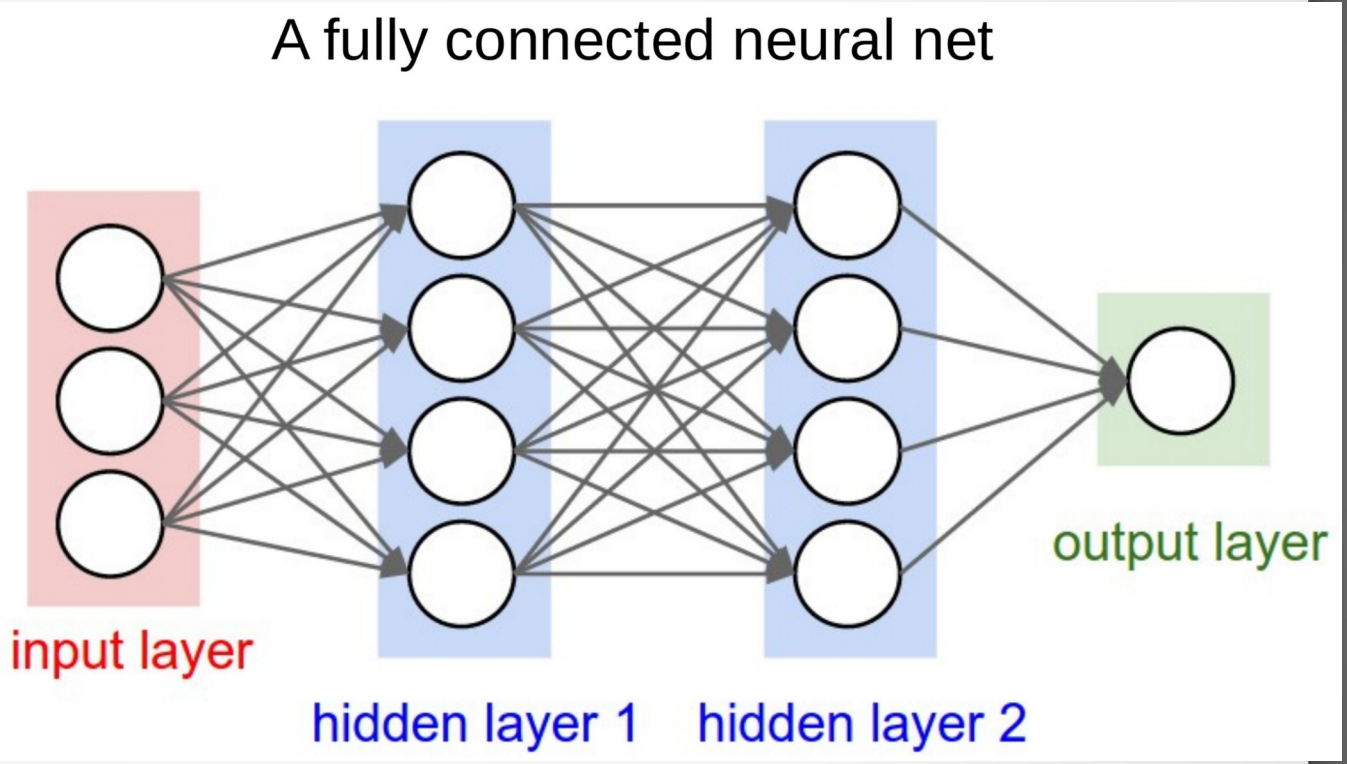
A Neural Network

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**

text_dataset_from_directory

Let’s look at the sa_with_own_data.py file.

I’ll broadly talk through the code now, but you’re also going to get a chance to look at this thoroughly in your groups shortly.

Overfitting

When we trained our model before we saw that, whilst training loss decreased and training accuracy increased in each new epoch, validation loss and accuracy seemed to peak early and we soon ran into “diminishing returns”.

This indicates overfitting – we’ve just taught the model to be excellent at predicting the features unique to the training examples, but we’re not improving predictive performance for unseen examples, thereby severely limiting our ability to *generalise* (which is the whole point of training the model).

To combat this, we can stop training before we reach that point. We could either experiment with the number of epochs to run, or we could use a handy tool called EarlyStopping to automatically stop training when we reach that point.

EarlyStopping

EarlyStopping is a *callback* and is imported from tensorflow.keras.callbacks

Callbacks operate meta to the training, and monitor the training process so they can take action if certain conditions are met.

EarlyStopping allows the training to stop early if a metric that is being *monitored* doesn't improve by a *threshold* with a certain level of *patience* (number of epochs).

Let's have a look at how we can add EarlyStopping to our code (sa_with_own_data_early_stopping.py; line 270 onwards)

Exercise 1

In your groups, I now want you to do the following :
a) Work through sa_with_own_data.py, with one person sharing their screen and talking through each line of the code and making sure everyone understands what each line of code is doing. Then move onto sa_with_own_data_early_stopping.py looking at line 270 onwards (this is the only stuff that has changed).

b) Try applying the model to predict the sentiment of new, unseen text. To do this (write all this at the end of the sa_with_own_data_early_stopping.py file – take a copy first though):

1. Store some text (such as a review, something you wrote etc) in a .txt file in the same directory as your .py files.
2. Open the file and read the whole thing into a string in one go using the *.read()* method of a file object (e.g. f.read())
3. Use the *.predict()* method of your exported and pass in a string of text for which you want to predict the sentiment (note – this must be passed in as a list, even if there is only one element)
4. Print a message specifying whether the sentiment is predicted to be positive or negative, based on a threshold of the output probability of 0.5 (the output probability is returned as an output from the *.predict()* method. Also print the output probability.

c) Try lots of different pieces of text to see how well it performs, including ones where sentiment is more unclear / mixed. Explore changing aspects of your model to try to improve the performance of the model. You could change training length and EarlyStopping parameters, try different batch sizes, different training / validation splits, change the structure of your Neural Network (add layers, add neurons, take them away, change dropout rates etc) etc.

You have 90 minutes, so make sure you take a break in there too! When we come back, we'll see if anyone managed to improve the performance of the original model!

A Further Challenge

Let’s consider another potential challenge in sentiment analysis.

Consider the following fictional data from a patient survey question asking “What did you feel about your stay in the hospital?”. Would you rate this as “Positive”, “Negative” or “Neutral”, and why?

“The care from the staff was absolutely fantastic – they made me feel safe and nothing was too much trouble. But when I was discharged I had to wait over 5 hours for my prescribed medications at the pharmacy – this is atrocious.”

Aspects

Clearly, there are both positive and negative elements to the above review.

If we were to consider the overall review as **positive**, then we are making the assumption that the care from the staff was more important to the reviewer than the wait at the pharmacy.

If we rate it as **negative**, then we are assuming that the wait at the pharmacy was more important than the care from the staff.

If we rate it as **neutral**, then we are assuming that both aspects are weighted equally by the reviewer, such that their overall feelings about their stay are considered neither positive nor negative.

So, what can we do…?

Aspects

There are two primary solutions that we could adopt :

- We could have a dual output to our sentiment analysis, such that a review can be considered both positive and negative (a *multi-label* model).

- We can break down the review into the individual **aspects** (e.g. the care from staff, the wait at the pharmacy) and analyse the sentiment of these individual aspects instead. This is known as **Aspect-Level Sentiment Analysis**.

Both approaches are valid, and, as ever, it depends on what you're hoping to achieve.

If we're trying to understand what things were good and what things were bad (vs just classifying reviews), then Aspect-Level Sentiment Analysis may provide a better solution.

Aspect-Level Sentiment Analysis

The key difference with Aspect-Level Sentiment Analysis is that we need to split our text into different *aspects* first. Once we've done that, we can simply analyse sentiment in the same way we would for any text.

Consider the following paragraph. What are the individual aspects in this text?

I tried to get through using the supplied telephone number first, but nobody answered after 10 minutes so I gave up. I then decided to go the surgery in person, but I was waiting for ages here too. The receptionist I eventually spoke to was very rude. When I finally saw a nurse, she was incredibly helpful, and put my mind at ease. I thought the coffee from their coffee machine was excellent too – makes a change!

Aspect-Level Sentiment Analysis

I tried to get through using the supplied telephone number first, but nobody answered after 10 minutes so I gave up. I then decided to go the surgery in person, but I was waiting for ages here too. The receptionist I eventually spoke to was very rude. When I finally saw a nurse, she was incredibly helpful, and put my mind at ease. I thought the coffee from their coffee machine was excellent too – makes a change!

Aspects :
1. The wait trying to get through on the telephone
2. The wait when attending the surgery in person
3. Rudeness of the receptionist
4. Helpfulness of the nurse
5. Quality of the coffee from the coffee machine

Noun Chunks

So how do we find aspects automatically? One option might be to look for **Noun Phrases** (also known as **Noun Chunks**).

Remember, noun phrases are phrases that have a noun as their head – the noun + the describing words around that noun.

Rather than do this manually, let's look at how we can use SpaCy to automatically identify the noun chunks (noun phrases) in our paragraph.

Let's look at noun_chunks_using_spacy.py

Noun Chunks

Let's compare the noun phrases we identified with the aspects we identified – we'll disregard the pronoun "I" / "she" etc noun chunks:

NOUN PHRASES / CHUNKS
the supplied telephone number
nobody
10 minutes
the surgery
person
ages
The receptionist
a nurse
my mind
ease
the coffee
their coffee machine
a change
ASPECTS
The wait trying to get through on the telephone
The wait when attending the surgery in person
Rudeness of the receptionist
Helpfulness of the nurse
Quality of the coffee from the coffee machine

Noun Chunks

We *might* be able to use noun chunks as a clue to finding individual aspects, but we'd need to be clever about removing the noise too!

In truth, the most difficult aspect (pun intended) of aspect-level sentiment analysis is trying to automate the extraction of the different aspects.

There continues to be a lot of research into this area, with many approaches using Deep Learning approaches, including Transformer-based models (that are able to use *attention* to determine context, as we discussed in the last session).

A word of caution

When undertaking sentiment analysis in the real-world, be mindful of a very real issue that you may encounter.

Any form of machine learning needs a good representation of all sides of the data to learn effectively. So, in the case of sentiment analysis, we need a good sample of positive and negative (and neutral, if we're including it) text.

But, in some cases, you will find that the vast majority of your data is overwhelming positive, for example. Which means that the machine will simply learn to say that everything is positive, because it'll be right most of the time by doing that.

So you may need to look at ways of mitigating this problem. For example, rebalancing your datasets, maybe even using synthetic data. At the very least, you should ensure you're not just using accuracy as your only performance metric for your model.