Master In Artificial Intelligence

Language Modelling – Year 2024/25

Lab Assignment Guide

Neural models for word vector representations

Word Representations in NLP

Discrete word representations:

1. Unable to handle semantic relations between words efficiently.
2. Semantically related words are not represented as close in the output spaces.
3. Very popular until the explosion of Deep Learning in NLP (~2013).

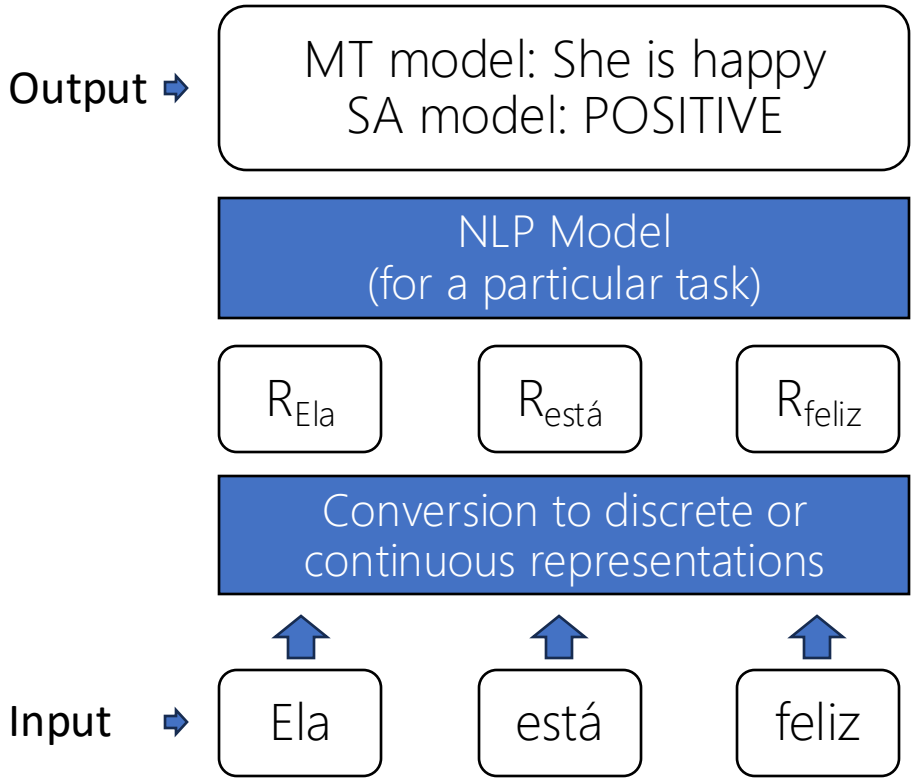
Continuous word representations:

1. They make possible to represent semantic relations between words in an efficient way.
2. Semantically similar words are represented by close values in the output space.
3. Very popular after 2013, and one of the main reasons (together with the computational power) that allowed us to achieve current models such as GPTs.

Word Representations in NLP

These representations, discrete or continuous, are used as input to solve different NLP tasks:

- Machine Translation
- Sentiment Analysis (POS/NEG/NEU)
- Question Answering
- Language Modelling
- …



Discrete Word Representations

Example of a Discrete Representation (*One-Hot Encoding*)

Let's assume a simple vocabulary:

{king, queen, man, woman, dog, cat}

king: [1, 0, 0, 0, 0, 0]
queen: [0, 1, 0, 0, 0, 0]
man: [0, 0, 1, 0, 0, 0]
woman: [0, 0, 0, 1, 0, 0]
dog: [0, 0, 0, 0, 1, 0]
cat: [0, 0, 0, 0, 0, 1]

Discrete Word Representations

Each word is assigned to a unique vector in which just one dimension has value 1, and all other positions are 0.

- Direct and easy to understand.
- It lacks the ability to represent relations between words: 'man' is not closer to 'woman', 'king', or 'queen', than to 'cat' or 'dog'.
- This was a major problem in developing NLP models that generalise well.

Continuous Word Representations

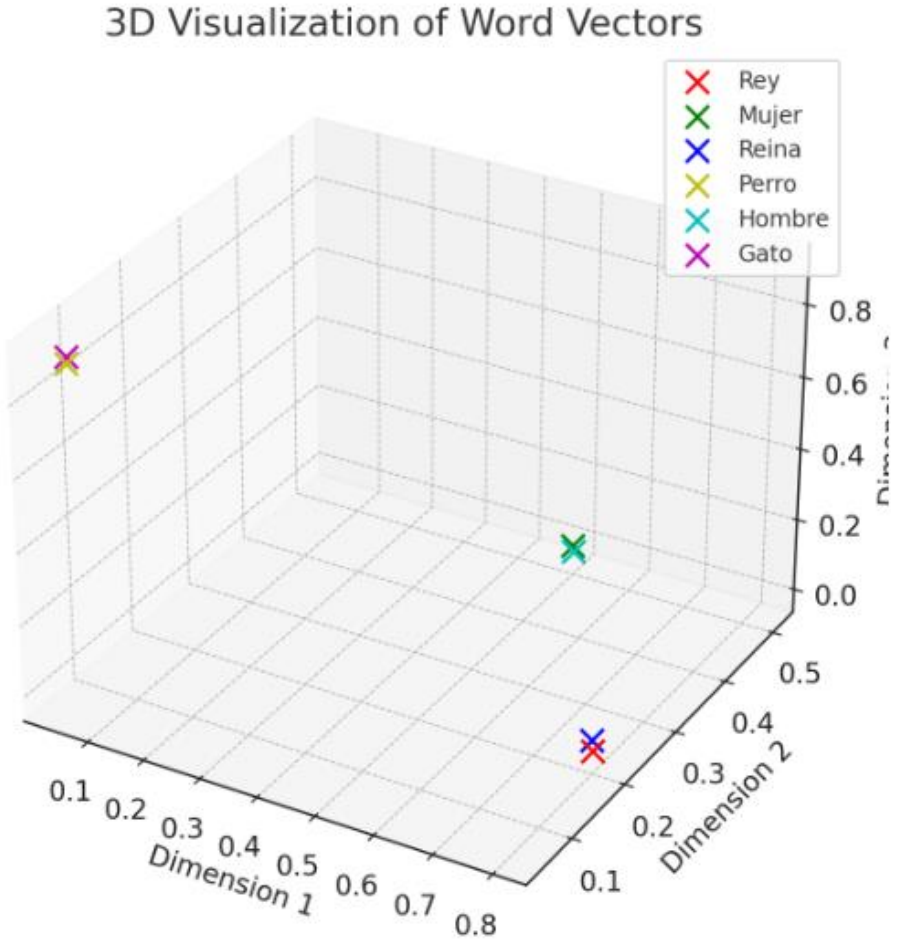
Words are represented in low-dimensional continuous vector spaces.

- Low-dimensionality: usually a few hundreds of dimensions.

Example (with only 3 dimensions):

king:	[0.8, 0.2, 0.0]	woman:	[0.49, 0.51, 0.0]
queen:	[0.79, 0.21, 0.01]	dog:	[0.05, 0.05, 0.9]
man:	[0.5, 0.5, 0.0]	cat:	[0.04, 0.06, 0.9]

Continuous Word Representations

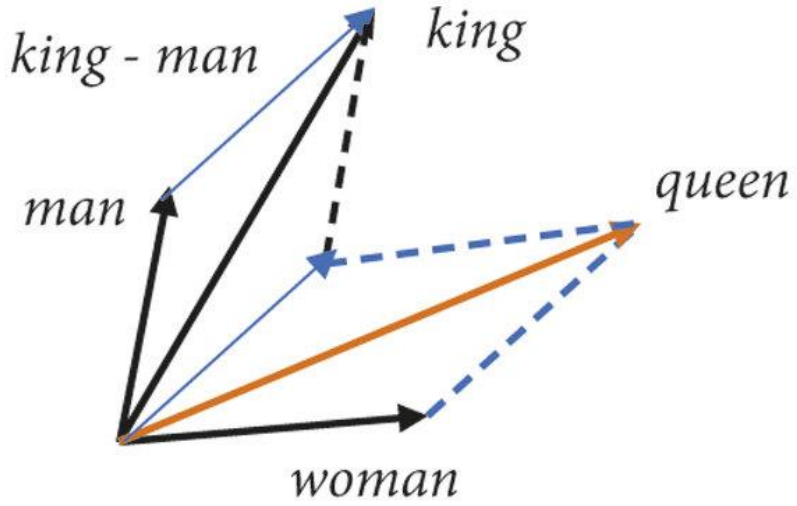


Continuous Word Representations

- "king" and "queen" are close together in the vector space, showing their semantic relatedness (e.g., monarchy).
- Similarly, "man" and "woman" are close together.
- "dog" and "cat" (both animals, potentially pets) are also represented in the same area of the vector space.

Continuous Word Representations

More complex relations (e.g., analogies), can also be captured by means of vector operations:

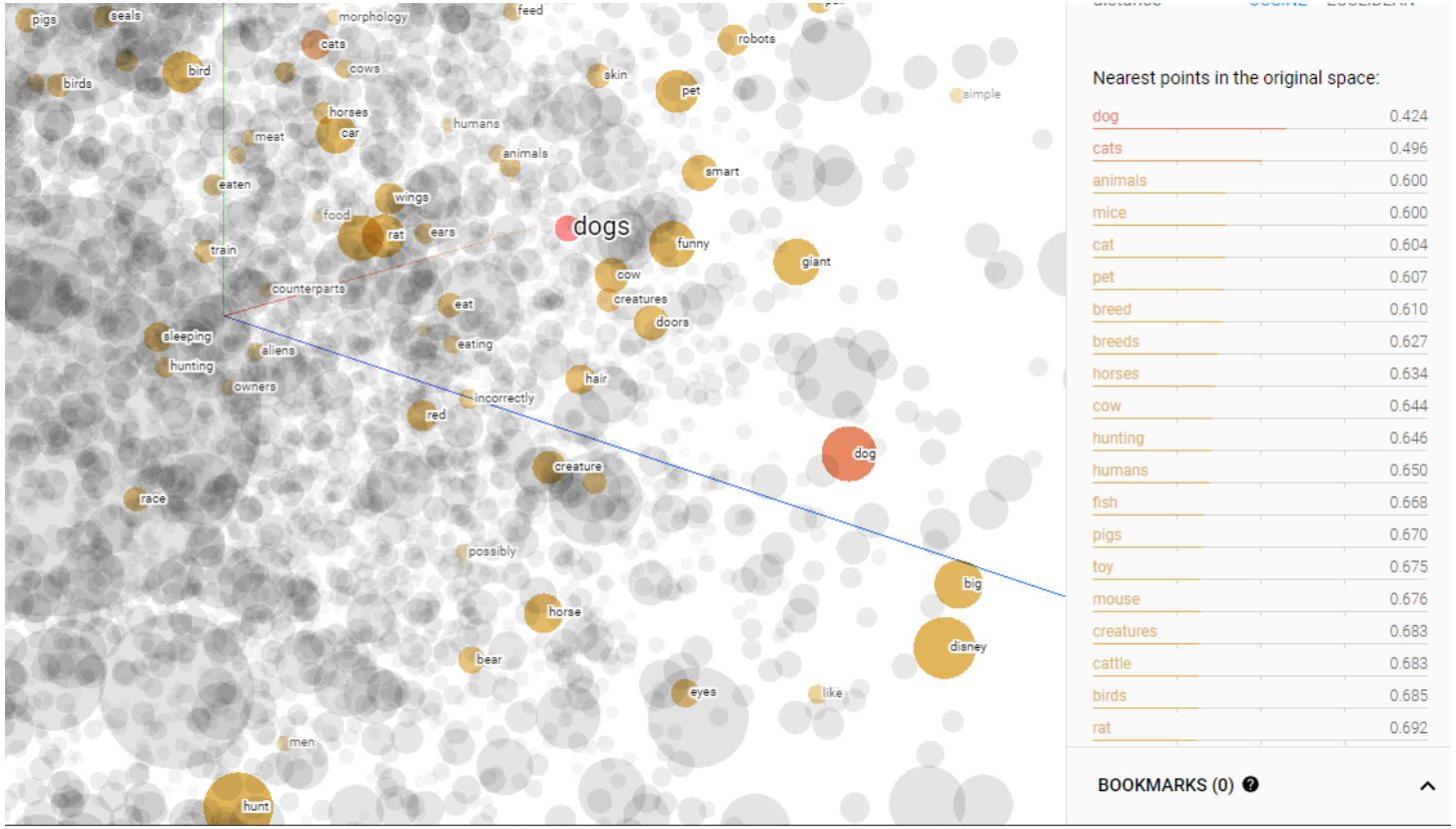


$king - man + woman \approx queen$

This is not possible using discrete representations, e.g. one-hot.

Continuous Word Representations

Visualisation demo: https://projector.tensorflow.org/



Continuous Word Representations

- How can we learn these representations?
- How can we make the learned vector of "dog" to be more similar to the one of "cat" than to the word vectors of "woman", "spoon" or "fly"?

In this assignment we will try to solve the following problem:

How to learn word representations that are useful to initialise deep learning models for a variety of NLP tasks.

Model based on target word prediction

- Input: window with *n* words at left and right of the target word.
- Output: target word.

Construction of training instances (n=2, both right and left contexts).

Example with some training samples for the sentence "Frodo jumped up and stood on a table":

- Input: ["Frodo", "jumped", "and", "stood"] Output: "up"
- Input: ["jumped", "up", "stood", "on"] Output: "and"
- Input: ["up", "and" "on", "a"] Output: "stood"

Model based on target word prediction

Conversion of strings of characters ("words") into numeric identifiers (to be handled by the Keras model).

- Input: ["Frodo", "jumped", "and", "stood"] Output: "up"
- Input: ["jumped", "up", "stood", "on"] Output: "and"
- Input: ["up", "and" "on", "a"] Output: "stood"

- Input: [101, 257, 58, 432] Output: 389
- Input: [257, 389, 432, 165] Output: 58
- Input: [389, 58, 165, 590] Output: 432

Model based on target word prediction

Training samples are submitted into batches to Keras. We can start with a batch size of 128, but you are free to test other sizes.

Large batches: more efficient, but potentially worse results.

Small batches: lower training, but possibly better generalisation.

Example of batch=3
With windows n=2

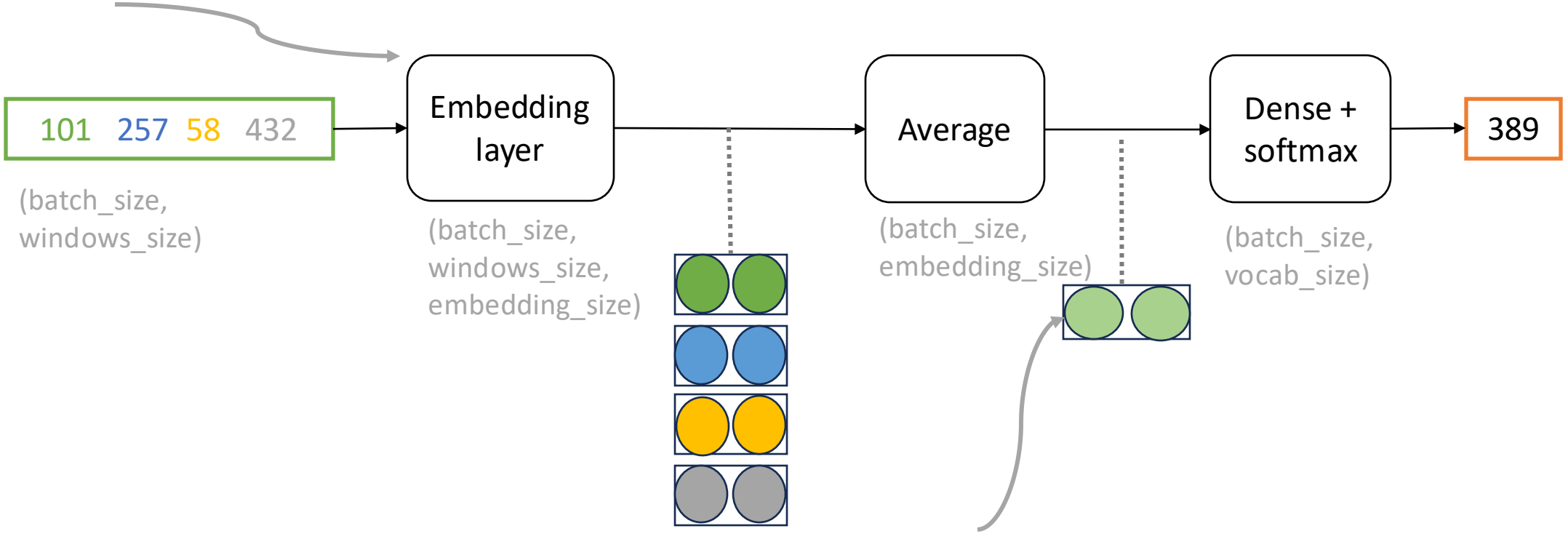
101 257 58 432
257 389 432 165
389 58 165 590

Associated outputs

389
58
432

Model based on target word prediction

The embedding layer obtains a vector for each word id. One can see this layer as a dictionary which maps word ids to vectors.



We compute the average of the batch embeddings for each vector dimension.

Model based on target word prediction

Keras layers potentially useful to develop this first model:

https://www.tensorflow.org/api_docs/python/tf/keras/Input

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Lambda
(using lambda x: K.mean(x, axis=1) as value for the *function* parameter, using import tensorflow.keras.backend as K)

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Extracting the word vectors

The dictionary which maps the word IDs to their embeddings can be extracted as follows:

```
model.get_layer(name_embeddings_layer).get_weights()[0]
```

Where *model* is the Keras model, and *name_embeddings_layer* the identifier given to the embeddings layer via its 'name' attribute.

Qualitative evaluation of word embeddings

Objectives:

1. To understand the structure and quality of the word embeddings generated by our models.
2. To identify semantic and syntactic relations captured for the word embeddings.

Techniques:

1. t-SNE visualisation: Non-linear dimensionality reduction technique useful to visualise high-dimensional data in two or three dimensions.
2. Cosine similarity: Measure to calculate the similarity between two vectors in an inner product space, by means of the cosine of the angle between them.

Visualisation with t-SNE

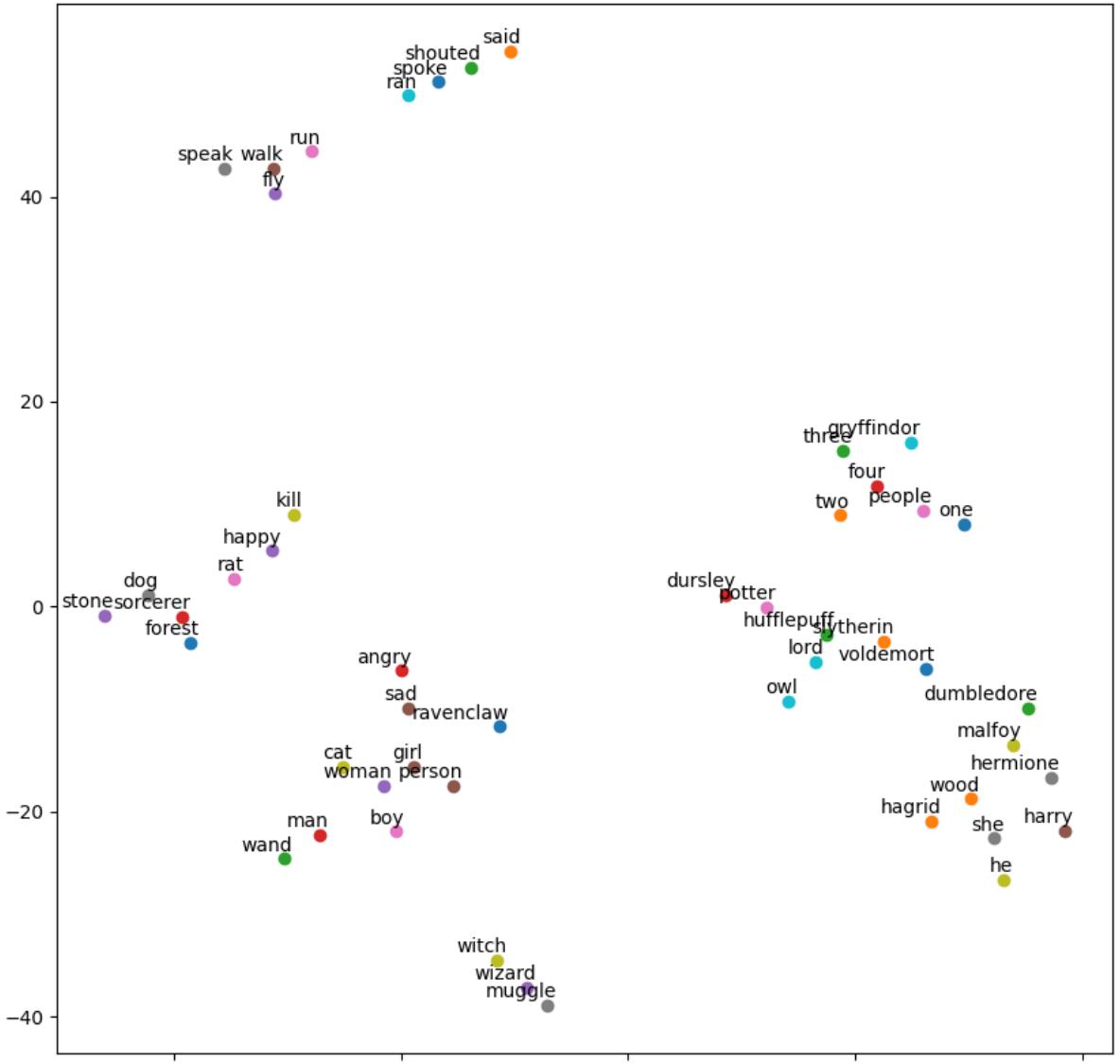
t-SNE reduces the vector dimensionality to a low dimensional space (2D or 3D), while keeping the similarity between the dots.

Steps:

1. Selection to the embeddings to visualise.
2. Application of t-SNE to reduce the dimensionality.
3. Visualisation (2D or 3D) to facilitate the interpretation of similarities and clustering of word vectors.

We provide sample code within the materials.

t-SNE visualisation example for some words of the 1st Harry Potter book



Cosine similarity

Rationale: Word vectors with high similarity correspond to words which occur in similar contexts, and therefore they are semantically related (in distributional terms).

Steps:

1. Selection of words on which we will compute the similarity.
2. Application of the cosine similarity function between each target word and the other words in the vocabulary.
3. For each target word, gather the *n* words with more similar vectors (e.g., the 10 most similar words).

Keras – keras.preprocessing.text

Module to preprocess natural language text, before being sent to the neural network.

In this assignment you can use it to:

1. Create and train the tokenizer (to split words into tokens in the dataset).
2. Transform words into IDs.
3. Known the size of the input vocabulary.

The dataset contains raw text with its original capitalization and punctuation. You must determine your approach for tokenization and preprocessing.

There is an example in materials/tokenizer.ipynb

Keras – tensorflow.gpu

To train neural models it is recommended to use a GPU to speed up the process. However, it should be possible to train the models also in a CPU.

To do so, it is possible to use the available resources provided by https://colab.research.google.com/

In materials/gpu.ipynb it is explained how to choose between GPU & CPU and how to check whether the GPU is being used.

Quantitative evaluation of word embeddings

Objectives:

1. To evaluate the impact of pre-trained word embeddings on the performance of a text classification model.
2. To compare performance against a baseline model trained without pre-trained embeddings
3. To analyse how embedding parameters (e.g. dimensionality, window size) influence model performance

Approach:

1. Train models with pretrained and randomized embeddings.
2. Measure performance of the models on a classification task.
3. Vary embedding settings to analyse their impact.

Creation and integration of embedding matrix

Dataset: Reuters

News classification dataset that can be downloaded using Keras. It requieres a slight preprocessing before using:

- Padding.
- Transform labels into one-hot encoding.

Create embedding matrix:

Adapt our embeddings to the dataset vocabulary:

- Load vocabulary indices from the downloaded dataset.
- Initialize a zeroes matrix and populate it: take the vectors from our pretrained vectors if the word exists, initialize randomly otherwise.

Evaluation and analysis

- Evaluate the trained model.
- How the pretrained embeddings affect the performance of the classification model? What are the potential benefits and drawbacks?
- How do different training parameters (dimensionality, window size or training epochs) affect the performance of the classification model?

Additional experiments

- Train on a larger corpus to improve model performance. You can use a bigger *Leipzig Corpora Collection's* (*https://wortschatz.uni-leipzig.de/en/download/English*) corpus than the one used, or create/gather your own combining multiple files.
- Compare the performance of your embeddings with some of the best-performing publicly available models, such as *fasttext* or *GloVe.* These models can be downloaded using *gensim* (https://radimrehurek.com/gensim/downloader.html).