Lecture 12: Word Embeddings and Large Language Models

Jack Blumenau

LSE THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Today's lecture

- Introduction to Word Embeddings
- Estimating Word Embeddings
- Using Word-Embeddings
- Applications
- Bias in Word-Embeddings
- Large Language Models
- Social Science Applications
- Conclusion

Final assessment

- Released at 6pm today (on Moodle)
- Due 6pm Monday 29th June
- Please work on the assessment alone
- Good luck!

Introduction to Word Embeddings

Learning from Context

Put these sentences into pairs on the basis of the similarity between the missing words

a. The concert was _____ and she was nearly moved to tears.
b. The _____ stood in the corner of the room.
c. They thought that the film was _____, particularly because the ending was so touching.
d. “What on earth is going on?”, _____ said.
e. _____ never knew how long it would last.
f. Sitting on the floor, beneath the _____, was a cat.

Learning from Context

Put these sentences into pairs on the basis of the similarity between the missing words

a. The concert was **beautiful** and she was nearly moved to tears.

b. The **chair** stood in the corner of the room.

c. They thought that the film was **charming**, particularly because the ending was so touching.

d. “What on earth is going on?”, **he** said.

e. **They** never knew how long it would last.

f. Sitting on the floor, beneath the **table**, was a cat.

Even without seeing the words themselves, you (hopefully) were able to infer something about their meaning from the words that surround them.

Sparse Representations of Words

Up until this point of the course, we have always implicitly used representations of words that are **sparse**.

Words were represented as *one-hot encoding*, i.e. word-specific vectors that take the value of 1 only for that word, and 0 for all others. E.g.

$$w_{\text{debt}} = [0, \quad 0, \quad 1, \quad 0, \quad \ldots, \quad 0] w_{\text{deficit}} = [0, \quad 0, \quad 0, \quad 1, \quad \ldots, \quad 0]$$

The problem with this representation is that they contain **no notion of similarity** between words.

The dot product between two word vectors is zero:

$$cos(\theta) = w_{\text{deficit}}^T w_{\text{debt}} = \frac{\mathbf{w}_{\text{deficit}} \cdot \mathbf{w}_{\text{debt}}}{||\mathbf{w}_{\text{deficit}}|| \, ||\mathbf{w}_{\text{debt}}||} = 0$$

This is true for any pair of words, which is clearly nonsense as some pairs of words are more similar to each other than other pairs of words.

Problems with Sparse Word Representations

Mechanical problems:

1. **Similarity**
    - Documents might have zero term overlap, but have nearly identical meanings
    - E.g. “Quantitative text analysis is very successful.” vs “Natural language processing is tremendously effective.”
2. **Classification/Dictionaries/Supervised scaling**
    - We may know or learn that one word is connected to a concept, but that doesn’t tell us anything about other similar words
    - If we learn that “turmeric” is highly predictive of the concept of interest, shouldn’t we also learn something about “garlic”, “saffron”, and “ginger”?
3. **Topic models/Unsupervised scaling**
    - If “bank”, “economy”, “interest”, and “rates” have high probability under a topic, shouldn’t “monetary” also have high probability?

Distributional Semantics

The distributional hypothesis: the meaning of a word can be derived from the distribution of contexts in which it appears.

- We can learn about the meaning of a word by investigating the distribution of words that show up around the word
  - “You shall know a word by the company it keeps!” J.R. Firth (1957)
  - “The meaning of words lies in their use” Ludwig Wittgenstein (1953)
- The hypothesis implies that words that appear in similar “contexts” will share similar meanings
- This simple (and old) idea is one of the most influential and successful ideas in modern natural language processing
- Word embedding approaches represent the distributional “meaning” of a word as a vector in multidimensional space

Distributional Semantics

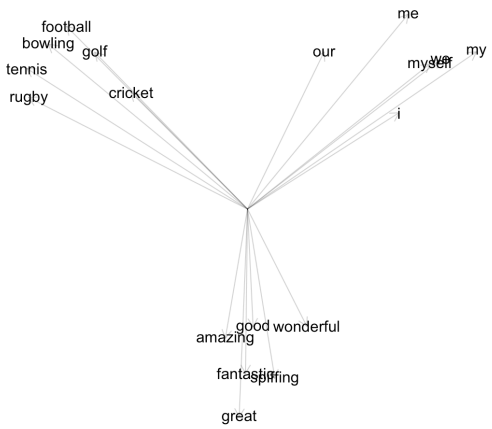
- When a word j appears in a text, its “context” is the set of words that appear nearby (within a fixed-size window)
- We use the many contexts of w to build up a representation of w

	pre	keyword	post
1	can be delivered for the	banking	industry in Europe . I
2	instance I am referring to	banking	. It is not only
3	that , if the second	banking	directive comes into force without
4	the future of the British	banking	industry within the European Community
6	the Government expect the second	banking	directive to come into force

	pre	keyword	post
1	during the passage of the	Finance	Bill , but I can
2	is referring to taxpayers '	finance	and public sector funding ,

Word Embedding Overview

1. The meaning of each word is based on the distribution of terms with which it co-occurs
2. We represent this meaning using a vector *for each word*
3. Vectors are constructed such that similar words are close to each other in “semantic” space
4. We build this space automatically by seeing which words are close to one another in texts



Dense Representations of Words

Our goal will be to build a dense vector for each word, chosen so that it is **similar** to vectors of words that appear in **similar contexts** (measuring similarity as the dot product)

$$w_{\text{debt}} = \begin{bmatrix} 0.73 \\ 0.04 \\ 0.07 \\ -0.18 \\ 0.81 \\ -0.97 \end{bmatrix} \qquad w_{\text{deficit}} = \begin{bmatrix} 0.63 \\ .14 \\ .02 \\ -0.58 \\ 0.43 \\ -0.66 \end{bmatrix}$$

These representations are known as **word embeddings** because we “embed” words into a low-dimensional space (*low* compared to the vocabulary size).

Advantages of Word Embeddings

Low-dimensional word embeddings offer three core advantages over simple word counts. They:

1. Encode similarity between words
    - We no longer have word similarities of zero!
    - Each word is a vector, with the vectors of similar words closer together than vectors of very different words
2. Allow for “automatic generalization”
    - Imagine that we discover “fantastic” is a good predictor of positive reviews, but we never observe the word “extraordinary” in our training corpus
    - Because “fantastic” and “extraordinary” will have similar word vectors, we can *share information across words*, and apply what we have learned about one word to our understanding of another
    - Can lead to large performance gains for prediction and topic modelling tasks
3. Provide a measure of meaning
    - We will often be interested in the meaning of words as a quantity in its own right

Contrasting Approaches

The material today is different on several dimensions from the approaches on the course so far:

	Traditional Unsupervised	**Traditional Supervised**	**Word Embeddings**
Bag of words?	Yes	Yes	No
Inputs	DFM	DFM	Feature co-occurence matrix
Outputs	Topics	Categorization	Word vectors

Estimating Word Embeddings

Design Choices in Word Embeddings

1. **Data**
    - High-quality embeddings require a large amount of training data
    - Usually trained on large external corpora (i.e. wikipedia; news articles; web pages)
    - Embeddings will reflect the language used in the training documents
2. **Context Window Size**
    - If meaning is defined by a word's context, we need to define context
    - Usually implemented as a symmetric window of some length around each word
    - The size of the window dictates the kind of information the embedding will capture
3. **Dimension of the Embedding**
    - The embedding for each word will typically be between 50 and 500 elements long
    - The embedding encodes information about the contexts a word appears in, so in theory larger embeddings are able to encode more information
    - In practice, medium-dimensional embeddings (100-300) work very well

Co-occurence Vectors

One simple “embedding” is produced by counting the occurrences of any term within a fixed window of any other term and store these in a feature-cooccurence matrix:

```
debates_fcm <-
  debates_corpus %>%
  tokens() %>%
  tokens_tolower() %>%
  fcm(context = "window",
      window = 3,
      tri = FALSE)

save(debates_fcm, file = "../data/debates_fcm.Rdata")

debates_fcm
```

```
Feature co-occurrence matrix of: 327,761 by 327,761 features.
          features
features   before     the  house proceeds      to choice      of       a speaker       ,
  before      948  119351  17939       89   28688    213   21405   18847     277   47978
  the      119351 3792520 580515     4167 3629935  13840 6681402  628635   18558 3962528
  house     17939  580515   2006      112  150334    176  211732   33340    1401  124961
  proceeds     89    4167    112        6     817      1    2572     197       0     870
  to        28688 3629935 150334      817 1053898   7231  736710  957469    5363 1291597
  choice      213   13840    176        1    7231    298    9407   10100      31    8430
  of        21405 6681402 211732     2572  736710   9407  755200 1244591    2787 1491028
  a         18847  628635  33340      197  957469  10100 1244591  161468    2751  985450
  speaker     277   18558   1401        0    5363     31    2787    2751     412   83721
  ,         47978 3962528 124961      870 1291597   8430 1491028  985450   83721 1766522
[ reached max_feat ... 327,751 more features, reached max_nfeat ... 327,751 more features ]
```

Co-occurence Vectors

Given this representation, we can calculate the cosine similarity between the word vectors of target words to find the closest other words in the embedding space:

```
library(quanteda.textstats)

word_similarities <- textstat_simil(debates_fcm,
                                    debates_fcm[which(featnames(debates_fcm) %in% c("election", "health", "ba
                                    method = "cosine",
                                    margin = "documents")


sort(word_similarities[,1], decreasing = TRUE)[1:10]
```

```
  election  elections referendum    general  electoral    elected  manifesto      party       vote     labour
 1.0000000  0.2697683  0.2190709  0.2078489  0.1907036  0.1864098  0.1858062  0.1847441  0.1841158  0.1833788
```

```
sort(word_similarities[,2], decreasing = TRUE)[1:10]
```

```
    health     mental        nhs       care    service   services     social healthcare  education    medical
 1.0000000  0.3688996  0.2943339  0.2893435  0.2541648  0.2453370  0.2348707  0.2285000  0.2264886  0.2239171
```

```
sort(word_similarities[,3], decreasing = TRUE)[1:10]
```

```
     banking        banks      lending    financial         bank institutions    corporate     consumer
regulatory      markets
   1.0000000    0.2769608    0.2318190    0.2301063    0.2215138    0.2132605    0.2086125    0.2057390
0.2057376    0.2046487
```

Co-occurence Vectors

- Co-occurence vectors clearly capture something about the meaning of words
- However, these vectors increase in the size of the vocabulary of the corpus
  - The vectors above each have length 327761
- One consequence is that they tend to be very sparse (most words fail to occur with most other words)
  - E.g. the sparsity of the example above is 99.9%
- In most applications, *sparse* vectors like these tend to perform less well than *dense* vectors
  - Similarity; classification; unsupervised learning, etc
- We would therefore prefer a low-dimensional representation that didn't suffer from these sparsity issues

Word-2-Vec Overview

Word2Vec (Mikolov et al, 2013) is a set of related methods for learning **dense** word vectors.

One version of Word2Vec – skip-gram with negative sampling – follows this basic process:

1. Start with a very large corpus of text (i.e. all of Wikipedia)
2. Represent each word in the vocabulary as a vector, μ_j
    - Initialise each vector with random numbers
3. Go through each *position*, t, in the text, where each position has
    - A center word, t ("target" words)
    - Context words, o ("outside" words)
4. Calculate the probability of observing o given t (or vice versa), using the similarity of the word vectors for o and t
5. Adjust the values of the word vectors, μ_j, to …
    - …maximize the probability of observing true context words
    - …minimize the probability of observing other words from the corpus

Word-2-Vec Intuition

What is the probability of observing the context words given the center wo

important to get UN agreement as $\underbrace{\text{the last}}_{\text{Context words window of size 2}}$ $\underbrace{\text{hope}}_{\text{Center word at position } t}$ $\underbrace{\text{of demonstrating}}_{\text{Context words window of size 2}}$ in

$$p(w_o = \text{the} | w_t = \text{hope})$$
$$p(w_o = \text{last} | w_t = \text{hope})$$
$$p(w_o = \text{of} | w_t = \text{hope})$$
$$p(w_o = \text{demonstrating} | w_t = \text{hope})$$

Word2Vec Objective Function

- The objective of the Word2Vec model is to maximise the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t)$$

where probability, $p(w_{t+j}|w_t)$, is defined as:

$$p(w_{t+j}|w_t) = \frac{\exp(v_o^T \cdot v_t)}{\sum_{w=1}^{W}\exp(v_o^T \cdot v_t)}$$

- This is an example of the **softmax** function, which maps arbitrary values to a probability distribution.
  - $v_o^T \cdot v_t$ is the dot product between word o and word t – measures the similarity between the two vectors
  - $\sum_{w=1}^{W}\exp(v_o^T \cdot v_t)$ – normalization factor to make things add up to 1

Learning Skip-Gram Embeddings (Negative Sampling)

The denominator of the softmax function defined above is very computationally expensive to evaluate repeatedly and so Word2Vec recasts the problem as supervised learning problem.

1. Select word from position t and select the "positive" words ($Y = 1$) that fall in its context (i.e. ± 2 words)

2. For each true context word, select K "negative" context words ($Y = 0$) at random from the entire corpus

3. Run a logistic regression, with the positive/negative variable as outcome, and the dot product between the words' embeddings as predictor ($v_o^T \cdot v_t$)

t	o	Y	$v_o^T \cdot v_t$
UN	to	1	0.279
UN	get	1	0.465
UN	agreement	1	0.36
UN	as	1	0.164
UN	castle	0	-0.174
UN	when	0	-0.23
UN	chair	0	0.043
UN	yoyo	0	-0.19
UN			

Word2Vec Intuition

Question: Why do the estimated word-embeddings encode information about word similarity?

- The predicted probability of a context word is high when the dot product between the context word's embedding and the target word's embedding is high
  - $p(w_{t+j}|w_t) = \frac{\exp(v_o^T \cdot v_t)}{\sum_{w=1}^{W} \exp(v_o^T \cdot v_t)}$
  - $v_o^T \cdot v_t = \sum_{i=1}^{n} v_{o,i} v_{t,i}$
- This encourages the model to find embedding vectors that are similar to one another for words that occur together frequently in the corpus
- It *also* encourages the model to find embedding vectors that are similar to one another for words that appear in similar *contexts* in the corpus, even if they rarely appear together. E.g.
  - "worldcom" and "scandal" appear frequently together
  - "enron" and "scandal" appear frequently together
  - But "worldcom" and "enron" appear infrequently together

GloVe: Global Vectors for Word Representation

- The Glove algorithm builds directly on the idea of the co-occurence vectors that we discussed previously
- Weighted least squares model that learns **dense** vectors from the word-word co-occurence counts
- GloVe models the log of the number of times that each word appears in the context of each other word:

$$\min_{\theta} J(\theta) \quad \text{where} \quad J(\theta) = \sum_{i=1}^{V} \sum_{i,j=1}^{V} f(X_{i,j})(v_i^T \cdot v_j - \log(X_{i,j}))^2$$

- Where
  - X is a word-word co-occurence matrix

GloVe vs Word2Vec

- The core difference between the two models is that GloVe is a model for the global co-occurence counts, while Word2Vec is an “online” model which trains progressively on a moving window
- The GloVe model has some advantages over Word2Vec
  - Very fast
  - Easily scales to very large corpora
  - Good performance on small corpora
- However, across many practical applications, there is no clear evidence that one model outperforms the other (Rodriguez and Spirling, 2020)
- In both models, the researcher has to make several decisions that can be consequential to the estimated word vectors
  - Context-window size
  - Embedding dimensionality

Glove Embeddings

```
1 glove <- readRDS("../data/glove.rds")
```

```
1 str(glove)
```

```
 num [1:400000, 1:300] 0.0466 -0.2554 -0.1256 -0.0769 -0.2576 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:400000] "the" "," "." "of" ...
  ..$ : NULL
```

```
1 glove[1,]
```

```
  [1]  0.0465600  0.2131800 -0.0074364 -0.4585400 -0.0356390  0.2364300 -0.2883600  0.2152100 -0.1348600
-1.6413000 -0.2609100  0.0324340  0.0566210 -0.0432960 -0.0216720  0.2247600 -0.0751290 -0.0670180 -0.1424700
0.0388250 -0.1895100  0.2997700  0.3930500  0.1788700 -0.1734300 -0.2117800
 [27]  0.2361700 -0.0636810 -0.4231800 -0.1166100  0.0937540  0.1729600 -0.3307300  0.4911200 -0.6899500
-0.0924620  0.2474200 -0.1799100  0.0979080  0.0831180  0.1529900 -0.2727600 -0.0389340  0.5445300  0.5373700
0.2910500 -0.0073514  0.0478800 -0.4076000 -0.0267590  0.1791900  0.0109770
 [53] -0.1096300 -0.2639500  0.0739900  0.2623600 -0.1508000  0.3462300  0.2575800  0.1197100 -0.0371350
-0.0715930  0.4389800 -0.0407640  0.0164250 -0.4464000  0.1719700  0.0462460  0.0586390  0.0414990  0.5394800
0.5249500  0.1136100 -0.0483150 -0.3638500  0.1870400  0.0927610 -0.1112900
 [79] -0.4208500  0.1399200 -0.3933800 -0.0679450  0.1218800  0.1670700  0.0751690 -0.0155290 -0.1949900
0.1963800  0.0531940  0.2517000 -0.3484500 -0.1063800 -0.3469200 -0.1902400 -0.2004000  0.1215400 -0.2920800
0.0233530 -0.1161800 -0.3576800  0.0623040  0.3588400  0.0290600  0.0073005
[105]  0.0049482 -0.1504800 -0.1231300  0.1933700  0.1217300  0.4450300  0.2514700  0.1078100 -0.1771600
0.0386910  0.0815300  0.1466700  0.0636660  0.0613320 -0.0755690 -0.3772400  0.0158500 -0.3034200  0.2837400
-0.0420130 -0.0407150 -0.1526900  0.0749800  0.1557700  0.1043300  0.3139300
[131]  0.1930900  0.1942900  0.1518500 -0.1019200 -0.0187850  0.2079100  0.1336600  0.1903800 -0.2555800
0.3040000 -0.0189600  0.2014700 -0.4211000 -0.0075156 -0.2797700 -0.1931400  0.0462040  0.1997100 -0.3020700
0.2573500  0.6810700 -0.1940900  0.2398400  0.2249300  0.6522400 -0.1356100
[157] -0.1738300 -0.0482090 -0.1186000  0.0021588 -0.0195250  0.1194800  0.1934600 -0.4082000 -0.0829660
0.1662600 -0.1060100  0.3586100  0.1692200  0.0725900 -0.2480300 -0.1002400 -0.5249100 -0.1774500 -0.3664700
0.2618000 -0.0120770  0.0831900 -0.2152800  0.4104500  0.2913600  0.3086900
[183]  0.0788640  0.3220700 -0.0410230 -0.1097000 -0.0920410 -0.1233900 -0.1641600  0.3538200 -0.0827740
0.3317100  0.2473800  0.0489280  0.1574600  0.1898800  0.0266420  0.0633150  0.0106730  0.3408900  1.4106000
```

Context-window Size

The size of the context window determines which *type* of word meaning is represented in the embedding space

- Small context windows ($\pm$ 1-3 words) $\rightarrow$ syntactic meaning
  - E.g. putting, bringing, taking, giving, providing, etc
- Medium context windows ($\pm$ 5-10 words) $\rightarrow$ semantic meaning
  - E.g. crimes, crime, offences, offence, prosecutions, murder, etc
- Large context windows ($\pm$ 10+ words) $\rightarrow$ topical meaning
  - E.g. tourism, visitors, museum, holiday, cafe, etc

$\Rightarrow$ the size of the window will depend on the research question.

Embedding Dimensions

The size of the embedding vectors determines how complex the model is that we wish to fit

- We have an embedding for each word, so increasing the embedding dimension by 1 multiplies the number of parameters to estimate by V (the total number of words in the vocabulary)
- Too many dimensions: higher chance of modelling noise
- Too few dimensions: higher chance of missing important subtleties in meaning
- General guidance: about 150-300 is fine (though this is not a very satisfying answer)

What do word-embedding dimensions "mean"?

- We can now generate multidimensional vectors for each of our words which, we will see shortly, are very successful in capturing semantic relations among words
- This implies that a meaningful semantic structure must be present in the respective vector spaces
- However, it is very difficult to answer questions such as "what do high and low values of the ith embedding dimension mean?"
- Example:

```
rownames(glove)[order(glove[,1], decreasing = F)][1:6]
```

```
[1] "samiul"     "stuffit"    "guangwei"   "decompress" "resend"     "sife"
```

```
rownames(glove)[order(glove[,22], decreasing = F)][1:6]
```

```
[1] "cheesecloth" "globe.com"   "zubaie"      "metrohealth" "25-march"    "30-aug"
```

```
rownames(glove)[order(glove[,300], decreasing = F)][1:6]
```

```
[1] "republish"     "12,000-page"   "transmittable" "affray"        "6-pica"        "spongiform"
```

Local Versus Pre-Trained

Either the Word2Vec or Glove methods can be applied to *any* large corpus of data. For applied research, there are typically two choices:

- Locally-trained embeddings
  - Collect a large corpus
  - Estimate a word-embedding model
  - Use the word-embeddings
  - Advantages: Can capture “local” meanings of words which may differ from more general use
  - Distadvantages: More computationally expensive and requires more coding decisions/effort
- Pre-trained embeddings
  - Download a pre-trained set of word-embeddings
  - Use the word-embeddings
  - Advantages: Usually high-quality embeddings trained on billions of texts

Visualisation

- It is common to see visualisations of word embeddings in lower dimensions
- There are many approaches to dimensionality reduction
  - Principal Component Analysis (PCA)
  - t-distributed stochastic neighbour embeddings (t-SNE)
- These visualisations often give helpful insights into the ways that language is used in the data that was used to train the models

Using Word-Embeddings

Similarity

- A key advantage of word embeddings: we can compute the similarity between words (or collections of words)
- The similarity between two words can be calculated as the cosine of the angle between the embedding vectors:

$$cos(\theta) = \frac{\mathbf{w}_i \cdot \mathbf{w}_j}{||\mathbf{w}_i|| \, ||\mathbf{w}_j||}$$

- We can then sort the words in order of their similarity with the target word and report the “nearest neighbours”

Similarity Demonstration

```
library(text2vec)

# Extract target embedding
target <- glove[which(rownames(glove) %in% c("taxes", "quantitative", "enron")),]

# Calculate cosine similarity
target_sim <- sim2(glove,
                   target)

# Report nearest neighbours
sort(target_sim[,1], decreasing = T)[1:10]
```

```
    taxes       tax    income    paying  taxation       pay  revenues      fees    excise     costs
1.0000000 0.8410683 0.6800698 0.6292870 0.6281890 0.6165617 0.5964931 0.5957133 0.5911929 0.5867159
```

```
# Report nearest neighbours
sort(target_sim[,3], decreasing = T)[1:10]
```

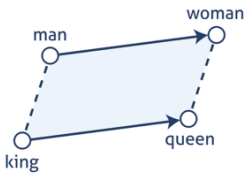
```
 quantitative   qualitative     empirical   measurement      analysis   methodology    analytical      analyses
methodologies     numerical
    1.0000000     0.6452297     0.5144293     0.4902190     0.4807779     0.4792444     0.4602726     0.4536292
0.4420934     0.4269905
```

```
# Report nearest neighbours
sort(target_sim[,2], decreasing = T)[1:10]
```

```
     enron    worldcom    skilling      fastow      dynegy    andersen executives accounting         aig    auditors
1.0000000   0.6551277   0.6350623   0.5476151   0.5255642   0.5219762   0.5188566   0.5134948   0.4963848   0.4742722
```

Analogies

- One surprising feature of word-embeddings is that they can capture more nuanced features of language than simple similarity
- Among the most widely discussed features of word embeddings is their ability to capture analogies via their geometry
- Analogies are linguistic expressions which describe processes of transfering information from one subject (the analogue) to another (the target)
- Example:
  - Apple is to tree as grape is to ____
  - King is to man as _____ is to woman



$$\text{vector}(\text{king}) - \text{vector}(\text{man}) + \text{vector}(\text{w}$$

Example process:

1. Compute vector $\text{vector}(\text{king}) - \text{vector}(\text{man}) + \text{vector}($
2. Calculate cosine similarity between new ve all word vectors
3. Report most similar vectors (normally exclu those for king, man and woman)

Analogies Demonstration

```r
# Extract vectors
taller <- glove[which(rownames(glove) == "taller"),]
tall <- glove[which(rownames(glove) == "tall"),]
thin <- glove[which(rownames(glove) == "thin"),]

# Generate analogy vector
target <- taller - tall + thin

# Calculate cosine similarity with all other vectors
target_sim <- sim2(glove,
                   matrix(target, nrow = 1))

# Print output
sort(target_sim[,1], decreasing = T)[1:10]
```

```
  thinner       thin    thicker     taller    slimmer   narrower noticeably     softer   slightly     weaker 
0.6823011 0.6795648 0.6409053 0.4766686 0.4659788 0.4645112 0.4491847 0.4438605 0.4299319 0.4269721 
```

Implication: Word-embedding vectors encode certain linguistic regularities that relate to the relation *between* different words.

Caveats:

1. Only works with reasonably common words
2. Only works for certain relations, but not others
3. Understanding analogy is an open area for research

Dictionary Expansion

- One helpful application of the word-similarity properties we have just discussed is that we can use them to automatically build more complete dictionaries
- Process
  1. Start with a small “seed” dictionary
  2. Calculate the average embedding of the words in the dictionary
  3. Calculate the cosine similarity between the dictionary embedding and all other words
  4. Report the most similar words and use them to extend the original dictionary
- This approach can enable us to find words associated with our concept of interest but which may not occur to the research *a priori*

Dictionary Expansion Application

```r
# Define seed dictionary
seed_dictionary <- c("hate", "dislike", "despise")

# Extract seed words
hate_words <- glove[which(rownames(glove) %in% see

# Calculate mean embedding
hate_words_vec <- colMeans(hate_words)

# Calculate cosine similarity with all other vecto
target_sim <- sim2(glove,
                   matrix(hate_words_vec, nrow = 1

# Print output
names(sort(target_sim[,1], decreasing = T))[1:40]
```

```
 [1] "despise"     "dislike"     "hate"
"loathe"      "hatred"      "hated"       "detest"
"hates"       "disdain"     "distrust"    "adore"
"resent"      "disliked"    "distaste"    "hating"
"dislikes"    "admire"      "loathing"    "feelings"
"bigotry"     "antipathy"
[22] "affection"   "envy"        "animosity"
"intolerance" "equate"      "abhor"       "despises"
"hostility"   "despised"    "profess"     "admiration"
"criticize"   "liking"      "fear"        "perceive"
"disrespect"  "mistrust"    "disliking"   "hateful"
```

Applications

Application 1

Are female politicians less aggressive than male politicians? (Hargrave and Blumenau, 2022)

In lecture 2 we investigated the claim that male and female politicians have distinct styles. Previously, we applied an existing sentiment dictionary to a corpus of parliamentary texts. Today, we will supplement this approach by using word-embeddings to automatically expand the set of words we use to score speeches.






Aggressive Word Dictionary

```
library(quanteda)
aggression_words <- read.csv("aggression_words.csv")[,1]
```

```
print(aggression_words)
```

```
  [1] "irritated"         "stupid"            "stubborn"          "accusation"        "acuse"
"accusations"       "accusing"          "anger"             "angered"           "annoyance"         "annoyed"
"attack"            "insult"            "insulting"
 [15] "insulted"         "betray"            "betrayed"          "blame"             "blamed"
"blaming"           "bitter"            "bitterly"          "bitterness"        "complain"
"complaining"       "confront"          "confrontation"     "fibber"
 [29] "fabricator"       "phony"             "fibber"            "sham"              "deceived"
"deceive"           "disgrace"          "villain"           "good-for-nothing"  "hypocrite"         "deception"
"steal"             "needlessly"        "needless"
 [43] "criticise"        "criticised"        "criticising"       "blackened"         "fiddled"
"fiddle"            "problematic"       "lawbreakers"       "offenders"         "offend"
"unacceptbale"      "leech"             "phoney"            "appalling"
 [57] "incapable"        "farcical"          "absurd"            "ludicrous"         "nonsense"
"laughable"         "nonsensical"       "ridiculous"        "outraged"          "hysterial"
"adversarial"       "aggressive"        "shady"             "stereotyping"
 [71] "unhelpful"        "unnatural"         "assaulted"         "assault"           "assaulting"        "half-
truths"       "petty"             "humiliate"         "humiliating"       "confrontational"   "hate"
"hatred"            "furious"           "hostile"
 [85] "hostility"        "nasty"             "obnoxious"         "sleeze"            "sleezy"
"inadequacy"        "faithless"         "neglectful"        "neglect"           "neglected"         "wrong"
"failure"           "failures"          "failed"
 [99] "fail"             "scapegoat"         "cruel"             "cruelty"           "demonise"
"demonised"         "tactic"            "trick"             "trickery"          "deceit"            "dishonest"
```

Although this is a reasonable-looking list of aggressive words, are there other words that MPs might use to criticise each other in parliamentary debate?

Estimating Word Embeddings

- In this instance, we will use the Glove model to estimate a local set of word-embeddings
- The hope is that this will allow us to pick up on the ways in which aggressive words are used in the specific context of parliamentary debate

```
library(text2vec)

# Load data

load("debates_fcm.Rdata")

## Fit GLOVE model

glove = GlobalVectors$new(rank = 150, x_max = 2500L)
debate_main = glove$fit_transform(debates_fcm, n_iter = 500, convergence_tol = 0.005, n_threads = 3,   learni

## Extract word embeddings
debate_context = glove$components

word_vectors = debate_main + t(debate_context)

save(word_vectors, file = "word_vectors_150.Rdata")
```

```
str(word_vectors)
```

```
num [1:14683, 1:150] 0.0983 0.1094 0.2122 0.1579 0.1368 ...
- attr(*, "dimnames")=List of 2
 ..$ : chr [1:14683] "house" "proceeds" "choice" "speaker" ...
 ..$ : NULL
```

Incorporating Word Embeddings

With our word-embeddings in hand, we can then use them to create a dictionary embedding by averaging over the embeddings for each word:

```
## Extract word embeddings of words in dictionary
target_words <- word_vectors[aggression_words,]

## Calculate mean embedding for this dictionary
target_vector <- colMeans(target_words)

## Distance between each word in the vocabulary and the mean embedding
cos_sim <- sim2(word_vectors,
                matrix(target_vector, nrow = 1))

## Store results
word_scores <- data.frame(score = cos_sim[,1],
                          in_original_dictionary = dimnames(cos_sim)[[1]]%in%aggression_words)
```

Incorporating Word Embeddings

```
word_scores <- word_scores[order(word_scores$score, decreasing = T),]
head(word_scores, 30)
```

```
                  score in_original_dictionary
disgraceful    0.6828765                  TRUE
shameful       0.6611104                  TRUE
outrageous     0.6553395                  TRUE
scaremongering 0.6348777                  TRUE
utterly        0.6147277                 FALSE
cynical        0.6142637                 FALSE
frankly        0.6091110                 FALSE
scandalous     0.6077674                  TRUE
dishonest      0.6039909                  TRUE
embarrassing   0.5921001                 FALSE
absurd         0.5897929                  TRUE
ridiculous     0.5887514                  TRUE
ludicrous      0.5873914                  TRUE
deplorable     0.5846311                  TRUE
incompetence   0.5773249                 FALSE
misguided      0.5683095                 FALSE
irresponsible  0.5675159                 FALSE
pathetic       0.5667197                 FALSE
appalling      0.5536031                  TRUE
dreadful       0.5514060                 FALSE
nonsense       0.5435853                  TRUE
bizarre        0.5403646                 FALSE
```

Scoring Speeches

- In addition to using this approach to finding words we might have missed, we now have scores associated with each word that indicate the relevance of the word to the concept of interest

- We can use these word-weights to score individual speeches

$$Score_i = \frac{\sum_w^W Sim_w N_{w,i}}{\sum_w^W N_{w,i}}$$

- Sim_w is the similarity score for each word embedding and the dictionary embedding

- $N_{w,i}$ is the tf-idf count of the word in the speech
- $Score_i$ therefore represents the fraction of words in sentence i that are relevant to the concept contained in the seed dictionary

- **Key advantage**: speech scores reflect the ways that aggressive words are used in the context of parliamentary debate

Comparison with Traditional Dictionaries

Bias in Word-Embeddings

Bias in Word-Embeddings

- An important substantive finding about word-embedding methods is that they can learn human biases in the semantic relationsips they encode into the vector space
- This occurs because they are trained on human-generated data: if biased relations between words occur frequently in natural language texts, the word-embeddings learn those biases

> "There is nothing about doing data analysis that is neutral. What and how data is collected, how the data is cleaned and stored, what models are constructed, and what questions are asked – all of this is political." Danah Boyd, NYU

- Another important theme of current work is in "de-biasing" word-embedding methods

Bias in Word-Embeddings, Example

```r
# Extract vectors
doctor <- glove[which(rownames(glove) == "doctor"),]
father <- glove[which(rownames(glove) == "father"),]
mother <- glove[which(rownames(glove) == "mother"),]

# Generate analogy vector
target <- (doctor - father) + mother

# Calculate cosine similarity with all other vectors
target_sim <- sim2(glove,
                   matrix(target, nrow = 1))

# Print output
sort(target_sim[,1], decreasing = T)[1:10]
```

```
   doctor     nurse   doctors     woman   patient    mother physician  pregnant  hospital   medical 
0.8397708 0.6648028 0.6255664 0.5923487 0.5839312 0.5719679 0.5527085 0.5417390 0.5404372 0.5336439 
```

Racial Bias in Word-Embeddings (Garg et al., 2018, PNAS)

Table 3. Top Asian (vs. White) adjectives in 1910, 1950, and 1990 by relative norm difference in the COHA embedding

1910	1950	1990
Irresponsible	Disorganized	Inhibited
Envious	Outrageous	Passive
Barbaric	Pompous	Dissolute
Aggressive	Unstable	Haughty
Transparent	Effeminate	Complacent
Monstrous	Unprincipled	Forceful
Hateful	Venomous	Fixed
Cruel	Disobedient	Active
Greedy	Predatory	Sensitive
Bizarre	Boisterous	Hearty

Break

Large Language Models

Dependencies in Language

- In this course we have largely focused on bag-of-words models
  - Tend to be a good choice for a wide range of datasets and tasks in text analysis in the social sciences
  - Dictionaries; topic models; Naive Bayes; etc
- Bag-of-word classifiers based on term frequencies can achieve good performance
- Yet, when the interdependent nature of words becomes important, more advanced models can become helpful that capture the dependencies in language

Language modelling

- We have seen several examples of **language models**, which we have taken to be probabilistic descriptions of word counts in documents
  - Naive Bayes: a distribution over words for each category
  - Topic models: a distribution over words for each topic; a distribution over topics for each document
- In all instances, we have considered bag-of-words models – models that do not take word order or dependency into account
- More advanced language models provide probabilistic descriptions for word *sequences*
- For instance, given a sequence of words, a language model might try to predict the word that comes next

Language Models

Language modelling: the task of teaching an algorithm to predict/generate what comes next

> the students opened their minds (?)

More formally: given a sequence of words $x^1, x^2, \ldots, x^t$, compute the probability distribution of the next word x^{t+1}:

$$P\left(x^{t+1}|x^t, \ldots, x^1\right)$$

where x^{t+1} can be any word in the vocabulary V.

Language Modelling Applications

Language models should be very familiar to you!

Why Should Social Scientists Care About Language Models?

- Language modelling has become a benchmark test that helps us measure our progress on predicting language use
- More relevantly to social scientists, language modelling is now a subcomponent of many NLP tasks, including those we have studied on this course
  - Topic modelling
  - Document classification
  - Sentiment analysis
  - etc
- Virtually all state-of-the-art natural language processing tools are based on language models of different types
- Social scientists are beginning to adopt these methods!

n-gram Language Models

Question: How might we learn a language model?

Old Answer: Use n-grams!

Idea: Collect statistics about how frequent different n-grams are…

a. the students opened their books
b. the students opened their laptops
c. the students opened their exams
d. the students opened their minds

…and use these to predict the next word when we see the phrase “the students opened their…”.

$$P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

Note: n-gram models require the *Markov* assumption: the word at x^{t+1} depends only on the preceding $n-1$ words!

n-gram Language Models

Suppose we are learning a 4-gram language model.

> ~~as the proctor started the clock, the~~ students opened their ____

$$P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

Example:

- Suppose we have a large corpus of text
- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
  - $\rightarrow P(\text{books}|\text{students opened their}) = 0.4$
- “students opened their exams” occurred 100 times
  - $\rightarrow P(\text{books}|\text{students opened their}) = 0.1$

In this example, discarding the word “proctor” results in the wrong prediction!

n-gram Language Models

The core problem with n-gram language models is *sparsity*.

- What if "students opened their w" doesn't occur in the data?
  - $\text{count}(\text{students opened their } w) = 0$
  - $P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})} = 0$
  - The probability of word w is zero!
- What if "students opened their" doesn't occur in the data?
  - Then we can't calculate the probability for *any* word!
- Increasing the size of the n-gram makes these sparsity problems worse
  - if "students opened their" only occurs 1000 times, "as the proctor started the clock, the students opened their" will occur many fewer times!
  - Trade-off between model accuracy and sparsity
- Increasing the size of the corpus helps with this problem a bit, but not much

$\rightarrow$ n-gram models are good for clarifying the intuition behind a language model but are

Neural Language Models

- These are models that can capture dependencies between words without running into the sparsity problems that affect n-gram models
- These models process sequences of inputs and predict sequences of outputs
  - *Input*: Each context word is associated with an embedding vector. The input vector is a concatenation of those vectors.
  - *Output*: probability distribution over the next word
- The key innovation is that they are based on *dense* representations of words (embeddings), rather than sparse representations
  - Removes the sparsity problem!
  - Better treatment of out-of vocabulary words
- Each word in a sequence updates a set of parameters which are then used to predict the final word in the sequence
  - Model architecture is often an RNN (recurrent neural network), that allows for longer sequences and for words further away from the target word to have predictive power

Neural Language Models

Advantages

1. Can process inputs of any length (not limited to 3 or 4 n-grams)
2. The prediction for x^{t+1} can use information from many steps back (at least in theory)
3. The model size does not increase for longer input sequences

Disadvantages

1. Computation is very slow
2. In practice, it tends to be that predictions are dominated by words close to the target word in the sequence (i.e. we still lack a way of seeing the importance of “proctor”)

Attention

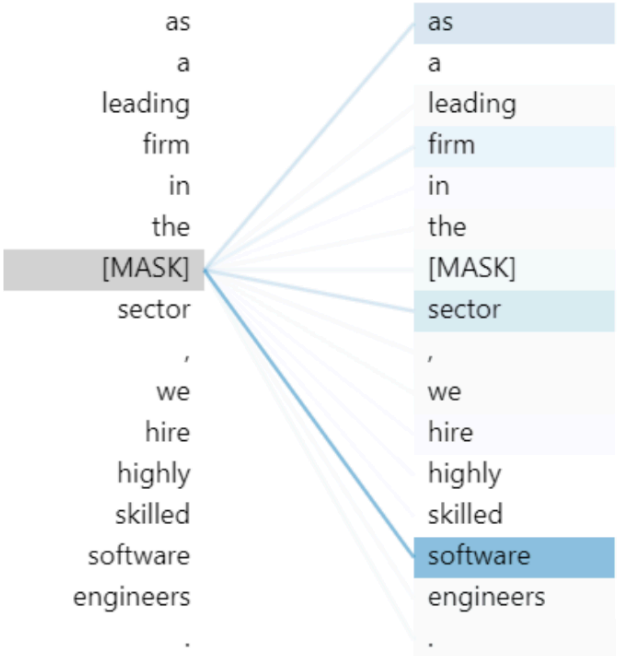
Consider these two sentences:

> As a leading firm in the ___ sector, we hire highly skilled software engineers.

> As a leading firm in the ___ sector, we hire highly skilled petroleum engineers.

- A human finds it easy to predict the missing words on the basis of the difference between “software” and “petroleum”.
- Word-embedding methods like Word2Vec, which also tried to predict words from their context, struggle because they weight all words in the context window equally when constructing embeddings
- Major breakthrough in modern NLP: train algorithms to also “pay attention” to the relevant features for prediction problems (Vaswani et al. 2023)

Attention



Word	Probability
IT	8.0%
Automotive	7.9%
Technology	7.2%
Healthcare	5.8%
Insurance	5.3%

Words have a darker shading when they are given more weight in the prediction problem.

Attention

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformers

- **Start with a Sequence of Words**: $x_1, x_2, \ldots, x_n$
- **Transform Words into Embedding Vectors**: We transform these words into embedding vectors, $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n$, which are numerical representations capturing the meaning of each word.
- **Add Positional Encoding**: We add *positional* encodings to the embedding vectors to provide the model with information about the position of each word in the sequence.
- **Iteratively Adapt Embedding Vectors**: We iteratively adapt these embedding vectors through multiple layers by combining:
  - **Information from the Embedding of Word** i: The current embedding vector of word i.
  - **Contextual Information via Self-Attention**: The embeddings of all other words in the sequence, using the attention mechanism to weigh their importance and relevance.
  - **Multiple Perspectives via Multi-Head Attention**: Different attention heads capture

Why Does Attention Help?

The key innovation of transformer models consists of introducing the concept of attention into a neural-network architecture.

- **Focus on Relevant Information:**
  - Attention allows the model to focus on the most relevant parts of the input sequence when making predictions. It enables the model to weigh the importance of each word relative to others in the sequence.

- **Contextual Understanding:**
  - By looking at different parts of the input simultaneously, the attention mechanism helps the model understand the context and relationships between words, even if they are far apart in the sequence.
  - E.g. The embedding vectors are updated such that instances of the same word that appear in different contexts will have different embeddings

- **Parallel Processing:**
  - Unlike traditional models that process text sequentially, attention mechanisms can

Transformer-based Language Models

Autoregressive Models (e.g., GPT):

- **Pretraining Task**:
  - Predict the next token in a sequence, having seen all the previous ones.
- **Attention Mechanism**:
  - During training, attention heads only attend to previous tokens, not subsequent ones.
- **Ideal Use Case**:
  - Best suited for text generation tasks where the model generates text one token at a time, predicting the next token based on the preceding context.

Autoencoding Models (e.g., BERT):

- **Pretraining Task**:
  - Mask some input tokens and train the model to reconstruct the original sequence.
- **Attention Mechanism**:
  - Utilize bidirectional representations, attending to both previous and subsequent

R packages

You need a python installation, and to call python code from R. Here some initial steps:

```r
# For importing python code
library(reticulate)

# Specify the path to your Python installation
use_python("/usr/bin/python3")

reticulate::py_install('transformers', pip = TRUE)

transformer = reticulate::import('transformers')
tf = reticulate::import('tensorflow')
builtins <- import_builtins() #built in python methods

tokenizer <- transformer$AutoTokenizer$from_pretrained('bert-base-uncased')
bert.model <- transformer$TFBertModel$from_pretrained("bert-base-uncased")
```

Social Science Applications

A Model To Rule Them All?

- One of the major strengths of LLMs is that they can perform a wide variety of tasks using the same modelling infrastructure
- Transformer infrastructure can be used for classical NLP tasks:
  - Classify topics (e.g. Schöll, Gallego, Le Mens)
  - Detect sentiment (e.g Simmons, Shaffer)
  - Measure Ideology (e.g. Baly et al)
- Transformer models give us an ability to *generate* text, not just measure latent concepts
- Generative AI can be used for new applications, such as
  - Complementing human labels (e.g. Wang et al)
  - Simulating human samples (e.g. Argyle et al)

Conclusion

Summing Up

- Word embedding methods provide a representation of word “meaning” by encoding information about the contexts in which words occur
- These vector result in a rich representation that allow us to measure the similarity between different words
- There are several modelling decisions to make when estimating word embeddings, including modelling approach, context-window size, and embedding length
- Language models describe a story about how texts are generated, using probabilities
- Modern large language models are built on a transformer infrastructure which make use of dense language representations
- You can use them to solve many classic NLP tasks

THANK YOU!