Overview of Natural Language Processing Part II & ACS L90

Lecture 7: Word Representations

Andrew Caines

based on slides by Ann Copestake, Simone Teufel, Paula Buttery, Weiwei Sun

Department of Computer Science and Technology
University of Cambridge

Michaelmas 2021/22

Overview

- **Lecture 6**: from *feature engineering* to *representation learning*
- $\approx$ *NLP in the last decade*
- **Today**: learning *word representations* from linguistic context, count-based approaches, prediction approaches
- ... evaluating word (token) reps, static vs dynamic, applications, problems.

Part 1: How to represent words

The Role of Context

The distributional hypothesis:

You shall know a word by the company it keeps.

J. R. Firth (1957) A synopsis of linguistic theory

the amount of meaning difference between two words "correspond[s] roughly to the amount of difference in their environments"

Z. Harris (1954) Distributional structure

the linguist's 'meaning' of a morpheme ... is by definition the set of conditional probabilities of its occurence in context with all other morphemes

M. Joos (1950) Description of language design

Distributional semantics: a family of techniques for representing word meaning based on contexts of use.

Word Representations

What is a word (token) representation?

- **Cognitively**: e.g. moon, mouse, Cambridge, king, King's, King's Parade, Sesame Street, the, be, of, *piscine, bocadillo, ö*
- **Formally**: Phil. tradition of MOON, MOUSE, KING; Or features e.g. mouse:[+furry, +mammal, +longtail, -rocky, -interplanetary, -large] vs moon:[-furry, -mammal, -longtail, +rocky, +interplanetary, +large]; Or prototypes for mouse, moon, etc.
- **Machine learning**: More abstract, more objective, learned from examples (attn. source of these)
- Note polysemy: mouse, net, etc
- Error: cockerel $\neq$ rooster
- And phrases: King's Parade, Sesame Street, Computer Science, CO_2 monitoring service, etc

Word representations via usage contexts (humans)

How to learn a word representation from scratch

Thought experiment: You keep hearing about NINDIN

it was authentic nindin, rather sharp and very strong

we could taste a famous local product — nindin

spending hours in the pub drinking nindin

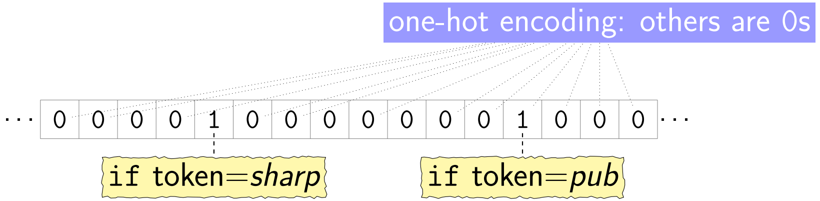
- Use linguistic context to represent word and phrase meaning (partially).
- Meaning space with dimensions corresponding to elements in the context: e.g. authentic, sharp, strong, local, product, **pub, drinking**

Word representations via usage contexts (machines)

How to learn a word representation from scratch

Task: learn about NINDIN

- Most computational techniques learn word vectors based on contexts of use: aka *semantic space models*, *vector space models*, *embeddings*.



- Roots in 1950s work by linguists such as Harris, Firth, Joos
- & Osgood et al's work on 3-dimensional rep of affective meaning via scores for valence (pleasantness), arousal (intensity of emotion), dominance (degree of control)

Distributional representation

it was authentic nindin, rather sharp and very strong

we could taste a famous local product — nindin

spending hours in the pub drinking nindin

[0 0 0 1 0 0 0 0 0 0 1 0 0 0 0] ▷sparse

$\Downarrow$

[0.456 0.193 5.39 1.235 -93.0] ▷dense

$\rightarrow$ *nindin* [. . . , pub 5.4, drink 1.2, strong 0.5, *joke* 0.2, *mansion* -0.93, *zebra* -0.93, . . .]

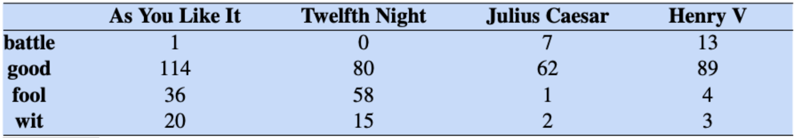
- We can now derive high-dimensionality *semantic space models*, *vector space models*, *embeddings* for word tokens from large corpora.
- Note that dimensionality easily gets very large: e.g. vocab of 50K in Penn TreeBank, 13.6M in Google 1T
- Even after filtering the target vocab: sparse vector with few 1s and many 0s

Part 2: Count-Based Approaches

Co-occurrence matrices

The *term-document matrix*:

- Words in rows
- Documents in columns
- Designed for information retrieval: identifying document relevance to key word searches



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

- $\rightarrow$ *Julius Caesar* represented as $[7, 62, 1, 2]$
- 4 dim. vector: in practice they are $|V|$ wide (size of selected vocabulary: might be a parameter)
- Normally: vocab filtering of ‘stopwords’ (*the, a, of, is, will*, etc) and others by e.g. frequency (TF-IDF), bespoke lists, noise cancelling

Co-occurrence matrices

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

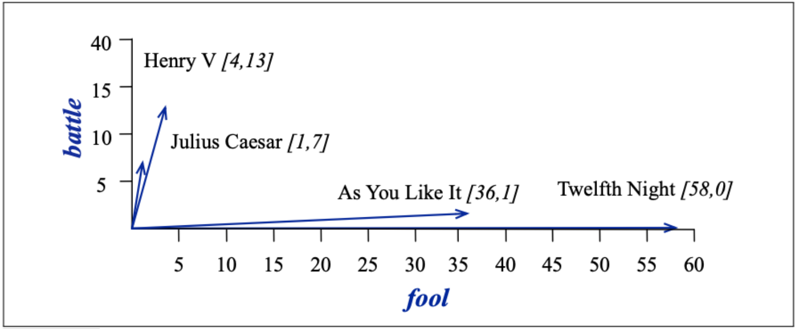


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

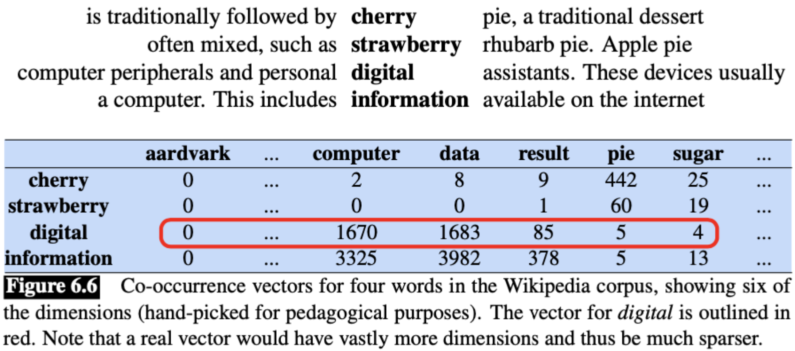
Co-occurrence matrices

The *term-term matrix*:

- Words in rows & columns
- Dimensionality $|V| \times |V|$
- Count co-occurrence of words in contexts using a corpus
- Context could be documents, sentences, or most often: windows of $\pm N$ word tokens
- The size of windows depends on your goals
    - Shorter windows $\Rightarrow$ more syntactic representation
    - Longer windows $\Rightarrow$ more semantic representation

The context window

An example with window size of ± 4:



is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Measuring word similarity

- Can measure similarity between 2 words, **v** and **w**, with vectors of same width $|V|$
- Most commonly: use *cosine similarity*
- Based on the **dot product** (or 'inner product') of **v** and **w**:

$$dotProduct(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_N w_N$$

- e.g. 5D vectors: $[1, 3, 2, 1, 5] \cdot [2, 4, 0, 1, 4] = 2 + 12 + 0 + 1 + 20 = 35$
- Captures high values in same dimensions ($\rightarrow$ same word co-occurrences)

Measuring word similarity

- But biased towards longer vectors (with greater frequency mass); we want a metric unaffected by word frequency
- e.g. high frequency word, 10D $[0, 10, 3, 12, 0, 0, 1, 5, 8, 0]$
- e.g. low frequency word, 10D $[0, 1, 0, 0, 1, 0, 2, 0, 0, 0]$
- Vector length (magnitude) defined as:

$$||\mathbf{v}|| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

- Thus **normalized dot product**:

$$\frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \, ||\mathbf{b}||} = \cos\theta$$

$\rightarrow$ the same as the cosine of the angle between 2 vectors **a** and **b**

Measuring word similarity

- *Cosine similarity metric* between two vectors **v** and **w** is thus:

$$cosine(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{||\mathbf{v}||\,||\mathbf{w}||} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

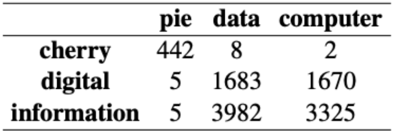
- Cosine from 1 (vectors in same direction, angle of $0°$) to 0 (orthogonal, $90°$) to -1 (opposite directions, $180°$)
- Word frequency counts are non-negative, therefore cosine similarity values from 0 to 1
- Calculate cosine sim for target word and other words in vocabulary, sort and look at top n

Measuring word similarity

Cosine similarity:

$$cosine(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{||\mathbf{v}|| \, ||\mathbf{w}||} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

Toy example from J&M:



	pie	**data**	**computer**
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$cos(cherry, info) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$cos(digital, info) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

The problem with raw counts

Raw word frequency is not a great measure of association between words

the and *of* are very frequent, but maybe not the most discriminative ... (could use stop-word lists)

$\rightarrow$ *go*, *get*, *today*, *tomorrow*, *extra*, *nice* ... (limitation of stop-word lists)

Two common solutions:

- **TF-IDF** (tf-idf): term frequency – inverse document frequency (often used for document reps)
- **PPMI**: positive pointwise mutual information (often used for word reps)
- Term-document matrices with tf-idf weighted values often a good baseline for document classification (scikit-learn functionality)
- tf-idf introduced by Karen Spärck Jones (1972), ex-CL

tf-idf

- *term frequency*: count of word token t in document d
- Often log_{10} to compress frequency range, since frequency does not linearly correlate with semantic importance; $+1$ to deal with zero values:

$$tf_{t,d} = \log_{10}(count(t,d) + 1)$$

- *inverse document frequency*: used to give a higher weight to words that occur in relatively few documents
- e.g. imagine the CompSci Tripos as a corpus of lecture notes: the word 'syntax' may be quite frequent in your corpus as a whole, but it probably occurs in many documents and does not therefore tell you a great deal about the meaning of an individual document d;
- whereas 'morphology' may be less frequent than 'syntax', but it will probably occur in a few documents, and its rarity makes it more distinguishing $\rightarrow$ thus we know that documents containing 'morphology' are likely to be about natural language (in a CS corpus)

tf-idf

- *inverse document frequency*:

$$idf_t = \frac{N}{df_t}$$

- where N is the corpus size in number of documents, and df_t is the number of documents in which term t occurs;

  $\rightarrow$ thus terms which occur in all documents have a weight of 1, those that don't have a weight greater than 1

- Also usually logged:

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

  $\rightarrow$ ever-present words have an idf weight of zero $(log_{10}(1))$, so they will be ignored in document representations, since tf-idf for word t in doc d:

$$w_{t,d} = tf_{t,d} \times idf_t$$

Document similarity

- How similar are 2 documents?
- *Document representation* as centroid (multi-dim mean) of word vectors in document, for words $1..k$:

$$d = \frac{w_1 + w_2 + \cdots + w_k}{k}$$

- Can then compare documents d_1 and d_2 by $cos(d_1, d_2)$
- Applications in: information retrieval, plagiarism detection, recommender systems, etc (often with tf-idf weighting)

PMI

- *Pointwise mutual information*: information-theoretic measurement (Fano 1961) – do events x and y co-occur more than if they were independent?

$$PMI(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

- PMI between target word w and context word c (Church & Hanks 1989):

$$PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

$\rightarrow$ thus a ratio of how much more 2 words co-occur than we expect by chance (numerator: how often we observe 2 words together; denominator: P of 2 independent events)

Positive PMI

- PMI from negative to positive infinity: negative PMI meaning that 2 words co-occur *less often* than expected by chance;
- But it's not clear people are good at *unrelatedness* judgements in order to evaluate such measures;
- So we replace negative PMI values by 0, hence *Positive PMI* (PPMI)

$$PPMI(x,y) = \max\left(\log\frac{P(x,y)}{P(x)P(y)}, 0\right)$$

- See J&M 3rd edn §6.6 for worked example, bias towards infrequent events, and ways to mitigate (weight by $^{\alpha}$ where $\alpha < 1$, or smoothing with constant k)

Part 3: Prediction-Based Approaches

Word embeddings

- From long, sparse count vectors to short, dense machine-learnt vectors: *word embeddings*
- Width tends to be 50..300..1000 (rather than vocabulary size)
- Dense, real-valued, possibly negative, vectors (rather than non-negative, mostly-zero sparse vectors)
- Dimensions do not have a clear interpretation but...
- They work better: *Don't count, predict!* (Baroni et al, ACL 2014)
- Because fewer weights to learn, less danger of over-fitting to the training data, at least...
- 'embedding' from mathematics: mapping from one space or mathematical structure (e.g. counts, observed) to another (e.g. latent, learned) $\rightarrow$ shifted to mean the resulting dense vector in the latent space
- Default meaning of (word) embedding: dense representation of a word token in a corpus of texts (but check this assumption)

Word embeddings

- Intuition: train a classifier on binary prediction task 'does word w co-occur with word v?'
- $\rightarrow$ learned classifier weights as word embeddings
- Unannotated corpora as training data, for 'self-supervised' learning (authors as the supervisors)
- **Q**: 'Does word w co-occur with word v?' **A**: Check in the training data
- Background: neural language models trained to predict the next word based on prior words (Bengio et al, 2003; Collobert et al, 2011)
- word2vec simpler than an NN: binary classification rather than word prediction (language modelling)
- & logistic regression rather than multi-layer NN (Mikolov et al, 2013)

word2vec

Two algorithms: target words & context windows

- Skip-grams (SG)

  Predict context words given target (position independent)
- Continuous Bag of Words (CBOW)

  Predict target word from bag-of-words context

$\rightarrow$ usually ‘word2vec’ = *skip-gram with negative sampling* (SGNS)

word2vec: SGNS

Skip-gram with negative sampling: given string x and context-window $\pm N$

1. Treat target word and context words as positive examples of word association
2. Randomly sample other words from V to get negative samples
3. Use logistic regression to learn how to distinguish the positive from negative instances
4. At end: learned weights are your word embeddings

word2vec: SGNS

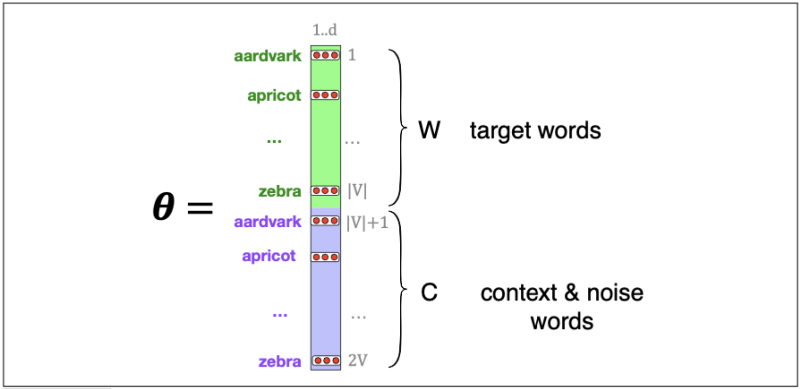


Figure 6.13 The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter θ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension d, formed by concatenating two matrices, the target embeddings **W** and the context+noise embeddings **C**.

(source: J&M 3rd edn, chapter 6)

word2vec: SGNS

Skip-gram with negative sampling: given string x and context-window $\pm N$
e.g. target word = *apricot* & window size = 2
... take a { tablespoon of apricot jam and } a teaspoon of...

1. Treat target word and context words as positive examples of word association: (apricot, tablespoon), (apricot, of), etc
2. Randomly sample other words from V to get negative samples: n.negative.examples $>$ n.positive.examples (ratio a parameter); sampling by weighted unigram frequency $P_\alpha(w)$ ($\alpha = \frac{3}{4}$ found to work well, boosts rare words)

   e.g. (apricot, aardvark), (apricot, axolotl), (apricot, matrix), (apricot, where), etc

word2vec: SGNS

Skip-gram with negative sampling: given string x and context-window $\pm N$
e.g. target word = *apricot* & window size = 2
... take a { tablespoon of apricot jam and } a teaspoon of ...

1. Treat target word and context words as positive examples of word association: (apricot, tablespoon), (apricot, of), etc
2. Randomly sample other words from V to get negative samples: e.g. (apricot, aardvark), (apricot, axolotl), (apricot, matrix), (apricot, where), etc
3. Use logistic regression to learn how to distinguish the positive from negative instances, cross-entropy loss function:

$$L_{CE} = -log\left[P(+|w, c_{pos})\prod_{i=1}^{k} P(-|w, c_{neg_i})\right]$$

For one true word-context pair, vs k noise words $c_{neg_1}, ..., c_{neg_k}$

word2vec: SGNS

- Logistic regression to learn how to distinguish the positive from negative instances, cross-entropy loss function:

$$L_{CE} = -log\left[P(+|w, c_{pos})\prod_{i=1}^{k} P(-|w, c_{neg_i})\right]$$

  For one true word-context pair, vs k noise words $c_{neg_1}, ..., c_{neg_k}$
- Minimize the loss function
- First term: maximise similarity of word-context pairs from positive examples
- Second term: minimise similarity of word-context pairs from negative examples (multiplied because independence assumption)
- Probabilities from embedding similarity: word-context co-occurrence if vectors are similar, i.e. a high dot-product (not normalized as cosine) from $-\infty$ to ∞, transformed to 0..1 with sigmoid function

word2vec: SGNS

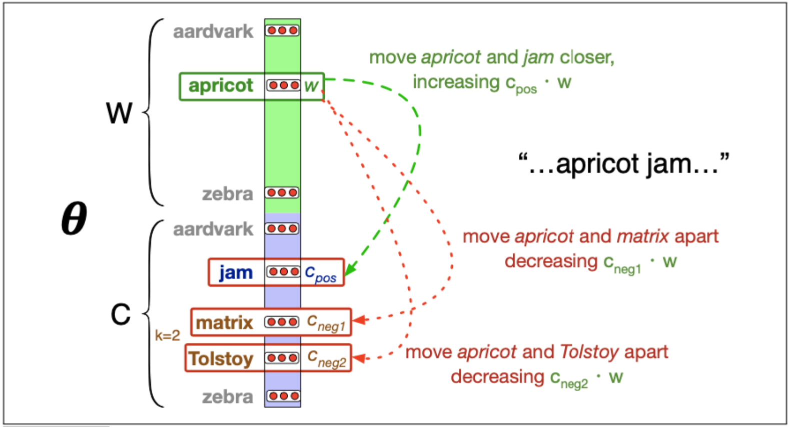
$$L_{CE} = -log\left[P(+|w, c_{pos})\prod_{i=1}^{k} P(-|w, c_{neg_i})\right]$$

- Probabilities from embedding similarity: word-context co-occurrence if vectors are similar, i.e. a high dot-product (cosine being a normalized dot product) from $-\infty$ to ∞, transformed to 0..1 with sigmoid function
- Thus: maximize dot product of true word-context pair, minimize dot product of word and k negative samples

$$L_{CE} = -\left[\log\sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]$$

- Minimize the loss function with stochastic gradient descent (prev lectures, J&M 3rd edn ch5)

word2vec: SGNS



aardvark
apricot
w
W
zebra
θ
aardvark
jam
c_{pos}
C
matrix
c_{neg1}
k=2
Tolstoy
c_{neg2}
zebra
move *apricot* and *jam* closer, increasing $c_{pos} \cdot w$
"…apricot jam…"
move *apricot* and *matrix* apart decreasing $c_{neg1} \cdot w$
move *apricot* and *Tolstoy* apart decreasing $c_{neg2} \cdot w$

Figure 6.14 Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

(source: J&M 3rd edn, chapter 6)

word2vec: SGNS

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w) \right]$$

- Minimize the loss function with stochastic gradient descent
- Randomly initialize weight matrices **W** and **C**
- Work thru training corpus adjusting **W** and **C** according to value of gradient & learning rate η
- "At end: learned weights are your word embeddings":
- Two embeddings per word: target embedding $\mathbf{w}_i$ and context embedding $\mathbf{c}_i$, stored in **W** and **C**
- Might add them together, or discard **C** and keep **W** only

word2vec: SGNS

- Context window size $\pm N$ affects performance: can be tuned on a devset
- Tokenization / phrases among pre-processing choices: e.g. so you can measure similarity of ‘san_francisco’, ‘los_angeles’, ‘golden_gate’, ‘san_diego’
- Limit on vocabulary size $|V|$ you aim to learn reps for (i.e. 10K most frequent words)
- Can sub-sample the frequent words, skip-gram window usually around 10
- Code repository made available: `https://code.google.com/archive/p/word2vec/`
- Plus pre-trained English Google News vectors (300w)

Other prediction-based approaches

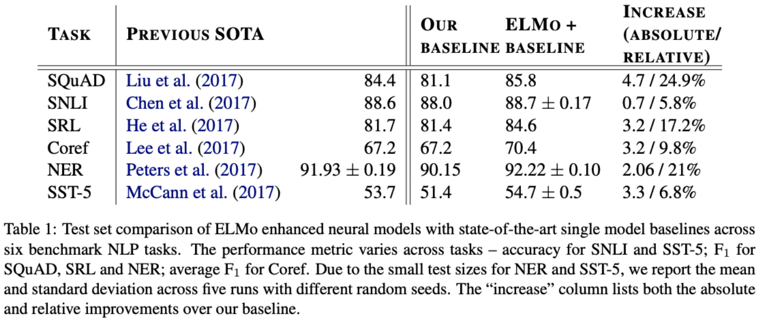
- fasttext: extension of word2vec by Facebook researchers (Bojanowski et al, 2017), deals with unknown words (‘OOVs’) at test time through sub-word models: character n-grams and special word boundary symbols $<$ and $>$. e.g. please + [<pl, ple, lea, eas, ase, se>]
- Good for morphologically-rich languages where vocab limits mean you either lose word forms or lemmatise and lose morph-information
- GloVe: ‘Global Vectors’, makes use of global corpus statistics (Pennington et al, 2014)
- Problem of word senses: *sense embeddings* (Camacho-Collados & Pilehvar, JAIR 2018)
- *Contextual embeddings* (vs ‘static’ ones): began with ELMo (Peters et al, NAACL 2018) word reps obtained from internal states of a bi-directional LSTM (next lecture)
- Moved on to the BERT family of large language models, pre-trained with transformers on huge web corpora: can obtain dynamic word reps for a target word in a new string, introduced by Devlin et al, NAACL 2019 (made accessible by `Hugging Face`)

Evaluation

- word2vec: Accuracy on curated word pairs for semantic and syntactic relations, vs randomly selected word pairs (later with phrases)
- e.g. Athens, Greece; brother, sister; apparent, apparently; mouse, mice; Switzerland, Swiss ($>$50% accuracy)
- And sentence completion from a list of 5 candidates;
- Benchmarking test suite by Baroni et al, ACL 2014: relatedness, synonyms, concept categorisation, analogy, selectional preferences in verb-noun pairs (e.g. people, eat)
- Word similarity datasets from human judgements through crowdsourcing: e.g. SimLex-999 (Hill et al, 2014)

Evaluation

- Later: downstream tasks, e.g. ELMo (Peters et al, 2018)

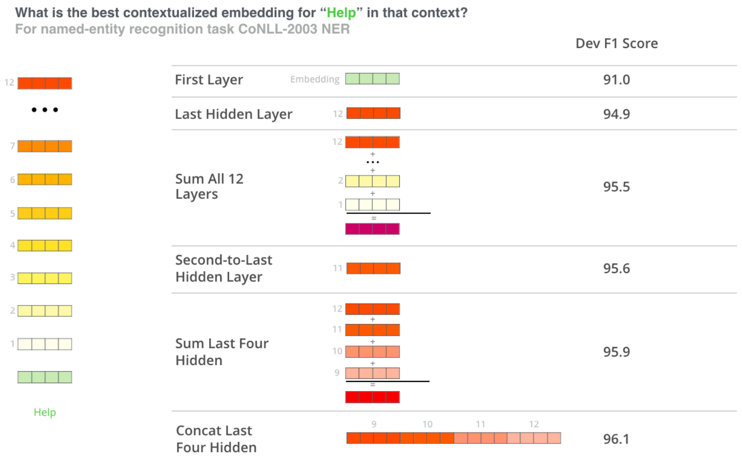


TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

Evaluation

- Features from BERT:



source: Jay Alammar, `https://jalammar.github.io/illustrated-bert/`

Part 4: Applications

Applications: what can we do with word embeddings?

- Analogy: a is to b as c is to what?
- e.g. ‘apple is to tree as grape is to ___?’
- Infamous: (king) $-$ (man) $+$ (woman) $=$ (queen); (Paris) $-$ (France) $+$ (Italy) $=$ (Rome)
- Hyped, but treat with caution!
- Ok if frequent words, small distances, certain relations (e.g. countries and capitals); not as good for other relations, also some constraints introduced (J&M 3rd edn §6.10)

Visualisation of semantic space

Example of embeddings learned for word sentiment (from 60D to 2D).

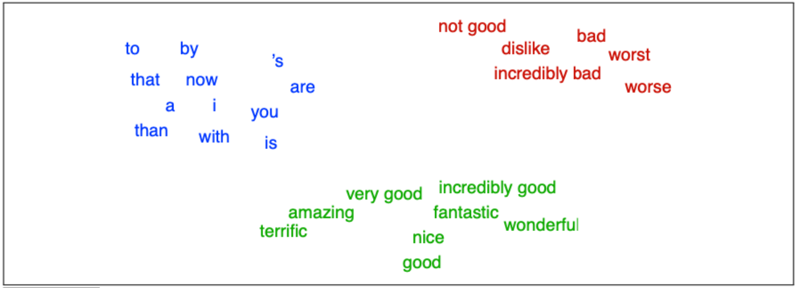


Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

From J&M 3rd edn chapter 6. t-SNE: 't-distributed stochastic neighbor embedding' projection method (van der Maaten & Hinton, 2008)

Cross-modal representations

Socher et al, NeurIPS 2013:

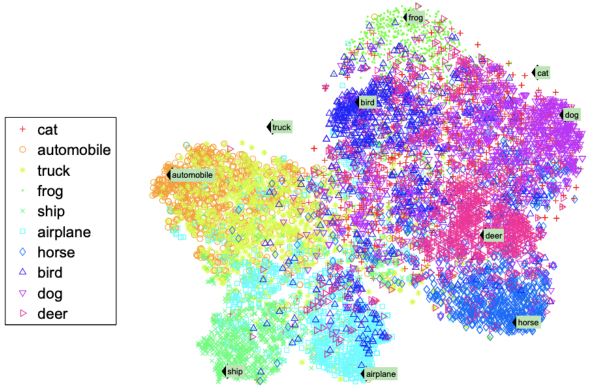


Figure 2: T-SNE visualization of the semantic word space. Word vector locations are highlighted and mapped image locations are shown both for images for which this mapping has been trained and unseen images. The unseen classes are cat and truck.

Diachronic analyses

Hamilton et al, ACL 2016:

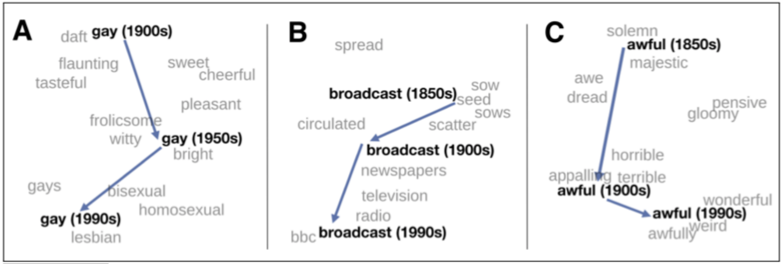


Figure 6.17 A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. The visualizations show the changes in the word *gay* from meanings related to "cheerful" or "frolicsome" to referring to homosexuality, the development of the modern "transmission" sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning "full of awe" to meaning "terrible or appalling" (Hamilton et al., 2016).

From J&M 3rd edn chapter 6.

Word embeddings in machine learning

- Inputs for machine learning: ubiquitous
- Because numeric, dense, self-supervised; advantages over hand-crafted, binary, domain-specific features
- For your input string x tokenized as $w_1, w_2, w_3, \ldots, w_n$
- e.g. Marek Rei & Helen Yannakoudakis, ACL 2016:

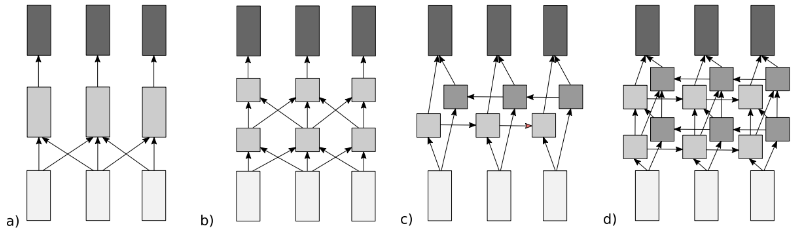


Figure 1: Alternative neural composition architectures for error detection. a) Convolutional network b) Deep convolutional network c) Recurrent bidirectional network d) Deep recurrent bidirectional network. The bottom layers are embeddings for individual tokens. The middle layers are context-dependent representations, built using different composition functions. The top layers are softmax output layers, predicting a label distribution for every input token.

Word embeddings in machine learning

- Inputs for machine learning: learning not just word representations
- For your input string x tokenized as $w_1, w_2, w_3, \ldots, w_n$

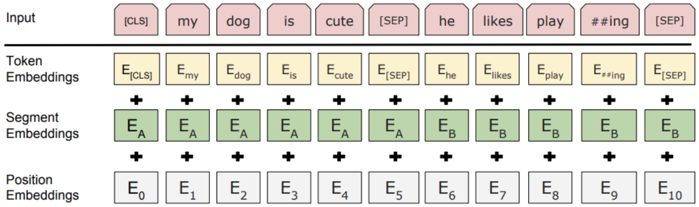

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Devlin et al, NAACL 2019: WordPiece tokenization (Hugging Face: SentencePiece / BPE)

Word embeddings in machine learning

- Cross-lingual word embeddings (Wu & Dredze, EMNLP 2019; Artetxe et al, ACL 2020; Conneau et al, ACL 2020)
- Non-parallel corpora in different languages: alignment methods
- $\rightarrow$ as a way to do 'zero-shot' cross-lingual tasks such as PoS-tagging, NER, NLI, initialise machine translation...

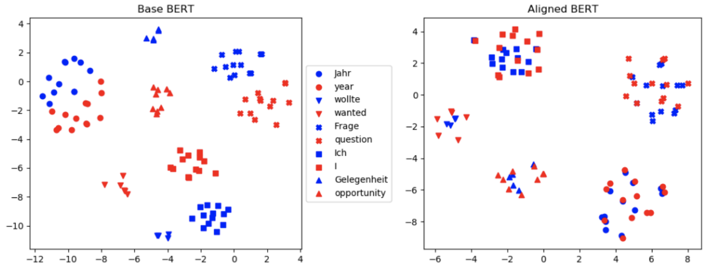


Figure 1: t-SNE (Maaten & Hinton, 2008) visualization of the embedding spaces of BERT pre- and post-alignment. Each point is a different instance of the word within a sentence in the Europarl corpus. This figure suggests that BERT begins already somewhat aligned out-of-the-box but becomes much more aligned after our proposed procedure.

Cao, Kitaev, Klein, ICLR 2020

Word embeddings in cognitive science

- Predicting eye-tracking (features), EEG (electrodes) and fMRI (voxels) vectors derived from published data
- Hollenstein et al, CoNLL 2019, aggregated error cp to random baseline:

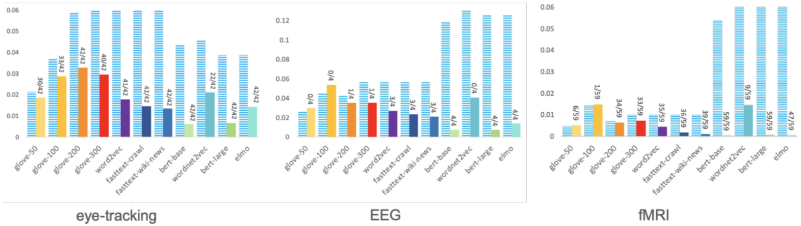


Figure 3: Results for each modality: Aggregated results for all embeddings predicting cognitive features for all datasets of a modality (sorted by dimension of embeddings in increasing order from left to right). The striped blued bars represent random baseline. The labels on the embedding bars show the ration of significant results under the Bonferroni correction to the total number of hypotheses.

- More in Formal Models of Language (Easter Term)

Reconsider the Distributional Hypothesis

Similarity = synonymy?

- Antonyms are basically as distributionally similar as synonyms:
- Distributional similarity is not referential similarity.
- Distinguishing synonyms from antonyms is a notoriously hard problem.

brief (A):
lengthy 0.256, hour-long 0.191, short 0.173, extended 0.163, frequent 0.162, recent 0.158, short-lived 0.155, prolonged 0.149, week-long 0.149, occasional 0.146

Encoding bias

- Gender: (man) − (computer programmer) + (woman) = (homemaker); Bolukbasi et al, 2016
- Problematic if e.g. word embeddings feature in recruitment systems: allocational harm
- Ethnicity: African-American names with higher GLoVe cosine to unpleasant words than European-American names; Caliskan et al, 2017
- Machine models demeaning specific social groups: representational harm
- Embeddings found to *amplify* biases found in the text (J&M 3rd edn §6.11)
- Ongoing research into 'debiasing' of word embeddings (and language models)

What is word meaning?

- For humans: meaning is not just word co-occurrence statistics
  - polysemy, dialect, slang, idiosyncrasy, pragmatics, politeness, affect, imageability, multimodal / multi-sensory, productive vs receptive, abstractness, implication, recency, salience, memories and associations, and much more
- Nonetheless much NLP progress with word embeddings

Summary

Count-based approaches

- Sparse vector representations
- Fast training
- Primarily used to capture word/document similarity

Prediction-based approaches

- Dense vector representations
- Improved performance on other tasks: easier to use as features in machine learning, may generalize better than storing explicit counts
- Can capture complex patterns beyond word similarity
- From word2vec (static; good baseline) to BERT (dynamic; usually SoTA)

Reading

- Ann's lecture notes.
  `https://www.cl.cam.ac.uk/teaching/2122/NLP/materials.html`
- Chapter 6 'Vector Semantics and Embeddings' by Jurafsky & Martin *Speech and Language Processing* 3rd edition (in prep).
  `https://web.stanford.edu/~jurafsky/slp3/6.pdf`
- More in Formal Models of Language and L101