Word embeddings

Herman Kamper

Motivation

Word embeddings are continuous vector representations of words.

If we could represent words as vectors that capture "meaning" then we could feed them as input features to standard machine learning models (SVMs, logistic regression, neural networks).

A first approach: One-hot vectors

Each word is represented as a length-V vector. The vector is all zeros except for a one at the index representing that word type.[1]

Example:

$$\texttt{cat} = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^\top$$
$$\texttt{feline} = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}^\top$$

But one-hot vectors on their own still cannot capture similarity:

- Any two one-hot vectors are orthogonal
- I.e. cosine similarity between two one-hot vectors is always 0

Cosine similarity and distance

We often use cosine similarity (or distance) to compare word embeddings. (Why not Euclidean?) The cosine similarity between two vectors is the cosine of the angle between them:

$$\cos\theta = \frac{(\mathbf{w}^{(a)})^\top \mathbf{w}^{(b)}}{\|\mathbf{w}^{(a)}\|\,\|\mathbf{w}^{(b)}\|} \in [-1, 1]$$

Cosine distance is defined as

$$d_{\cos}(\mathbf{w}^{(a)}, \mathbf{w}^{(b)}) \triangleq 1 - \frac{(\mathbf{w}^{(a)})^\top \mathbf{w}^{(b)}}{\|\mathbf{w}^{(a)}\|\,\|\mathbf{w}^{(b)}\|} \in [0, 2]$$

[1] In this note I use $V = |\mathcal{V}|$ to denote the vocabulary size.

Word2vec

Word2vec (Mikolov et al., 2013a) is a framework for learning word embeddings.

It relies on an idea that is the basis of many modern NLP approaches:

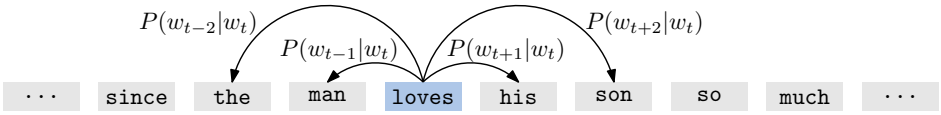
> A word's meaning is given by the words that frequently appear close-by.

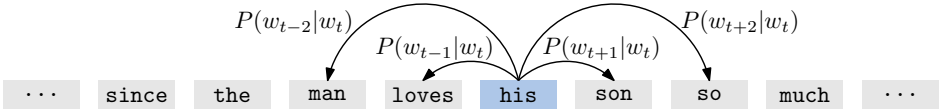
Two model variants:

1. Skip-gram: Predict context words given centre word
2. Continuous bag-of-words (CBOW): Predict centre word from context words

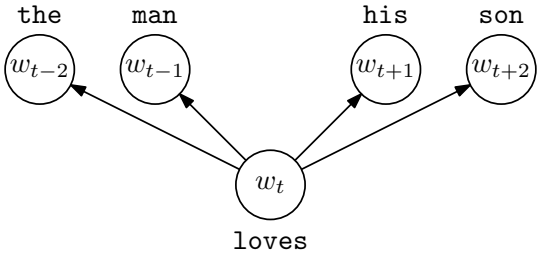
Word2vec: Skip-gram

Example windows in skip-gram:





Skip-gram predicts context words given a centre word:



Skip-gram loss function

Assumptions:

1. Each window is an i.i.d. sample
2. Within each window, each context word is conditionally independent given the centre word, e.g.

$$P(\texttt{the}, \texttt{man}, \texttt{his}, \texttt{son} | \texttt{loves}) = P(\texttt{the}|\texttt{loves})\, P(\texttt{man}|\texttt{loves})\, P(\texttt{his}|\texttt{loves})\, P(\texttt{son}|\texttt{loves})$$

A dataset gives a large number of (w_t, w_{t+j}) input-output word pairs.

For now, let's pretend that our training set consists of a single long sequence $w_{1:T}$.

Skip-gram loss function

We minimise the negative log likelihood (NLL) of the parameters:[2]

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\log \prod_{t=1}^{T} P_{\boldsymbol{\theta}}(w_{t-M}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+M} | w_t) \\ &= -\log \prod_{t=1}^{T} \prod_{\substack{-M \le j \le M \\ j \neq 0}} P_{\boldsymbol{\theta}}(w_{t+j} | w_t) \\ &= -\sum_{t=1}^{T} \sum_{\substack{-M \le j \le M \\ j \neq 0}} \log P_{\boldsymbol{\theta}}(w_{t+j} | w_t) \end{aligned}$$

In practice we optimise the average NLL, i.e. we divide by the number of pairs. (This number of terms is not equal to T. Why not?)

[2] This is not the probability of the training sequence $w_{1:T}$ like in a language model, but the probability of all the windows. With the two assumptions, this then becomes the product of the individual word-pair probabilities.

Skip-gram model structure

How do we model $P_{\boldsymbol{\theta}}(w_{t+j}|w_t)$? What structure do we use?

Assumption 3: Each context word can be predicted from the centre word in the same way, irrespective of its position. E.g. the prediction of w_{t-2} and w_{t-1} is done in the same way from w_t.

For each word type we have two vectors:

- $\mathbf{v}_w$ when w is a centre word
- $\mathbf{u}_w$ when w is a context word

For centre word c and context word o we use the following model:

$$P_{\boldsymbol{\theta}}(w_{t+j} = o|w_t = c) = P(o|c) = \frac{e^{\mathbf{u}_o^\top \mathbf{v}_c}}{\sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c}}$$

where V is the vocabulary size.

This can be written as a softmax function. For input $w_t = c$, the model outputs a probability distribution over the vocabulary:

$$\boldsymbol{f}_{\boldsymbol{\theta}}(w_t = c) = \begin{bmatrix} P(1|c) \\ P(2|c) \\ \vdots \\ P(V|c) \end{bmatrix} = \frac{1}{\sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c}} \begin{bmatrix} e^{\mathbf{u}_1^\top \mathbf{v}_c} \\ e^{\mathbf{u}_2^\top \mathbf{v}_c} \\ \vdots \\ e^{\mathbf{u}_V^\top \mathbf{v}_c} \end{bmatrix} = \text{softmax}(\mathbf{U}\mathbf{v}_c)$$

Skip-gram word embeddings

- We get two vectors for a single word type: $\mathbf{v}$ and $\mathbf{u}$
- Normally in skip-gram we use the $\mathbf{v}$ vectors as word embeddings
- These are represented together in matrix $\mathbf{V}$
- For skip-gram we don't use the context vectors $\mathbf{U}$ at test time

Skip-gram optimisation

Parameters: $\boldsymbol{\theta} = \{\mathbf{V}, \mathbf{U}\}$

Perform gradient descent on each parameter vector:

$$\mathbf{v}_c \leftarrow \mathbf{v}_c - \eta \frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{v}_c}$$
$$\mathbf{u}_o \leftarrow \mathbf{u}_o - \eta \frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{u}_o}$$

We need the gradients with respect to the loss function:

$$J(\boldsymbol{\theta}) = -\sum_{t=1}^{T} \sum_{\substack{-M \le j \le M \\ j \neq 0}} \log P_{\boldsymbol{\theta}}(w_{t+j} | w_t)$$

Consider inside term for a single training pair $(w_t = c, w_{t+j} = o)$:

$$\begin{aligned} J_{c,o}(\boldsymbol{\theta}) = -\log P(o|c) &= -\log \frac{e^{\mathbf{u}_o^\top \mathbf{v}_c}}{\sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c}} \\ &= -\left[\log e^{\mathbf{u}_o^\top \mathbf{v}_c} - \log \sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c} \right] \\ &= -\left(\mathbf{u}_o^\top \mathbf{v}_c - \log \sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c} \right) \end{aligned}$$

You can show that

$$\begin{aligned} \frac{\partial J_{c,o}(\boldsymbol{\theta})}{\partial \mathbf{v}_c} &= -\mathbf{u}_o + \sum_{j=1}^{V} \frac{e^{\mathbf{u}_j^\top \mathbf{v}_c}}{\sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c}} \mathbf{u}_j \\ &= -\mathbf{u}_o + \sum_{j=1}^{V} P(j|c)\, \mathbf{u}_j \end{aligned}$$

and similar for the derivatives w.r.t. the $\mathbf{u}$ vectors.

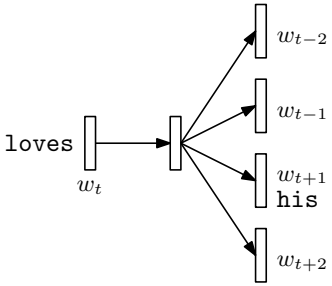
Example of skip-gram embeddings

For a skip-gram trained on the the AG News dataset, we print the closest word embeddings (cosine distance) to a query word:

```
Query: referendum
    mandate (0.5513) vote (0.5551) ballots (0.5872)
    vowing (0.5916) constitutional (0.6109)

Query: venezuela
    chavez (0.5229) venezuelas (0.5840)
    venezuelan (0.6057) hugo (0.6171) counties (0.6229)

Query: war
    terrorism (0.6230) raging (0.6280) resumed (0.6296)
    independence (0.6422) deportation (0.6444)

Query: pope
    ii (0.5573) democrat (0.6149) canaveral (0.6323)
    edwards (0.6377) sen (0.6379)

Query: schumacher
    johan (0.5250) ferrari (0.5493) trulli (0.5651)
    poland (0.5885) owen (0.5921)

Query: ferrari
    rubens (0.5049) austria (0.5281) barrichello (0.5416)
    schumacher (0.5493) seasonopening (0.6042)

Query: soccer
    football (0.4817) basketball (0.5429)
    mens (0.5510) fc (0.5530) ncaa (0.5547)

Query: cricket
    kolkata (0.4818) test (0.4908) oval (0.5444)
    bangladesh (0.5585) tendulkar (0.5756)
```

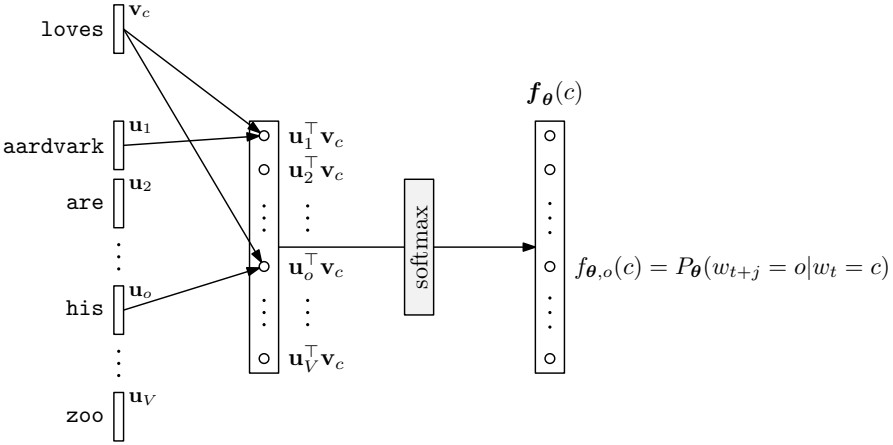
Skipgram as a neural network

In the paper:



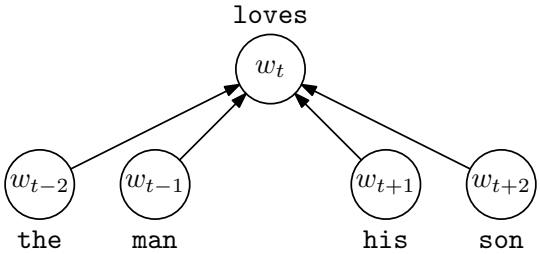
But this makes it look like we are predicting the context words given the window word, which is a bit misleading.

More accurately, as a neural network vector diagram:



Word2vec: Continuous bag-of-words (CBOW)

The continuous bag-of-words (CBOW) model predicts a centre word given the context words:



$$P(\texttt{loves}|\texttt{the}, \texttt{man}, \texttt{his}, \texttt{son})$$

Assumption: Each window is an i.i.d. sample

Loss function

We again minimise the NLL of the parameters:

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\log \prod_{t=1}^{T} P_{\boldsymbol{\theta}}(w_t|w_{t-M}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+M}) \\ &= -\sum_{t=1}^{T} \log P_{\boldsymbol{\theta}}(w_t|w_{t-M}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+M}) \end{aligned}$$

Model structure

We now have multiple context words in a single training sample, so we calculate an average context embedding $\bar{\mathbf{v}}_o$. This gives the model:

$$
\begin{aligned}
&P_{\boldsymbol{\theta}}(w_t = c | w_{t-M} = o_1, w_{t-M+1} = o_2, \ldots, w_{t+M} = o_{2M}) \\
&\quad = \frac{\exp\left\{\mathbf{u}_c^\top \bar{\mathbf{v}}_o\right\}}{\sum_{k=1}^{V} \exp\left\{\mathbf{u}_k^\top \bar{\mathbf{v}}_o\right\}} \\
&\quad = \frac{\exp\left\{\frac{1}{2M}\mathbf{u}_c^\top \left(\mathbf{v}_{o_1} + \mathbf{v}_{o_2} + \ldots + \mathbf{v}_{o_{2M}}\right)\right\}}{\sum_{k=1}^{V} \exp\left\{\frac{1}{2M}\mathbf{u}_k^\top \left(\mathbf{v}_{o_1} + \mathbf{v}_{o_2} + \ldots + \mathbf{v}_{o_{2M}}\right)\right\}} \\
&\quad = \frac{e^{\mathbf{u}_c^\top \bar{\mathbf{v}}_o}}{\sum_{k=1}^{V} e^{\mathbf{u}_k^\top \bar{\mathbf{v}}_o}}
\end{aligned}
$$

Optimisation

As for skip-gram, we optimise the parameters using gradient descent. The gradients can be derived as for skip-gram.

Word embeddings

Unlike in skip-gram, for CBOW it is common to use the context vectors as word embeddings (denoted as $\mathbf{v}$ for CBOW). The centre embeddings $\mathbf{u}$ are thrown away.

Skip-gram with negative sampling

Mikolov (2013b) proposed an extension of the original skip-gram to overcome some of its shortcomings.

In standard skip-gram we have:

$$P_{\boldsymbol{\theta}}(w_{t+j} = o | w_t = c) = P(o|c) = \frac{e^{\mathbf{u}_o^\top \mathbf{v}_c}}{\sum_{k=1}^{V} e^{\mathbf{u}_k^\top \mathbf{v}_c}}$$

The normalisation is over V terms, which could (probably is) huge!

Negative sampling

Instead treat as binary logistic regression problem:

- $y = 1$ when a centre word $w_t = c$ is paired with a word $w_{t+j} = o$ occurring in its context window
- $y = 0$ when a centre word $w_t = c$ is paired with a randomly sampled word $w_{t+j} = k$ not occurring in its context

The model now becomes:

$$P_{\boldsymbol{\theta}}(y = 1 | w_t = c, w_{t+j} = o) = \sigma(\mathbf{u}_o^\top \mathbf{v}_c) = \frac{1}{1 + e^{-\mathbf{u}_o^\top \mathbf{v}_c}}$$

If we had just one positive pair $(w_t = c, w_{t+j} = o)$ in our training set, the NLL would be

$$\begin{aligned} J_{c,o}(\boldsymbol{\theta}) &= -\log P_{\boldsymbol{\theta}}(y = 1 | w_t = c, w_{t+j} = o) \\ &= -\log \sigma(\mathbf{u}_o^\top \mathbf{v}_c) \end{aligned}$$

If we only had this single positive example $(y = 1)$, we could easily hack the loss by just making $\mathbf{u}_o$ and $\mathbf{v}_c$ really big. So we need some negative examples $(y = 0)$.

For each positive pair $(w_t = c, w_{t+j} = o)$ we sample K words not occurring in the context window of c. The loss now becomes:

$$\begin{aligned}
J_{c,o}(\boldsymbol{\theta}) &= -\log \Big[P_{\boldsymbol{\theta}}(y=1|w_t=c, w_{t+j}=o) \\
&\qquad \prod_{k=1}^{K} P_{\boldsymbol{\theta}}(y=0|w_t=c, w_{t+j}=k)\Big] \\
&= -\log P_{\boldsymbol{\theta}}(y=1|w_t=c, w_{t+j}=o) \\
&\qquad - \sum_{k=1}^{K} \log P_{\boldsymbol{\theta}}(y=0|w_t=c, w_{t+j}=k) \\
&= -\log \sigma(\mathbf{u}_o^\top \mathbf{v}_c) - \sum_{k=1}^{K} \log\Big(1 - \sigma(\mathbf{u}_{w_k}^\top \mathbf{v}_c)\Big) \\
&= -\log \sigma(\mathbf{u}_o^\top \mathbf{v}_c) - \sum_{k=1}^{K} \log \sigma(-\mathbf{u}_{w_k}^\top \mathbf{v}_c)
\end{aligned}$$

General ML idea: Contrastive learning

This idea of contrasting observations in the vicinity of a sample with negative samples from elsewhere forms the basis for the more general idea of contrastive learning in machine learning.

Global vector (GloVe) word embeddings

Old-school word embeddings

Before methods like the (neural-like) word2vec framework, word embeddings were based on co-occurrence counts.

A co-occurrence count matrix was typically decomposed using some matrix factorization approach to get lower-dimensional word embeddings.

Advantages over word2vec:

- Very fast
- Capture global statistics that word2vec misses: Word2vec considers windows within a batch; just counting can easily tell you about how words co-occur over an entire corpus

Despite these advantages (especially the speed one), word2vec just works better (normally).

But could we get the best of both worlds? The global vector (GloVe) word embedding method tries to do this.

The GloVe model

$C_{c,o}$ is the total number of times that centre word c occurs with context word o in the same context window. (For old-school approaches, we would have started by collecting these counts.)

GloVe tries to minimise the squared loss between the model output $f_{\boldsymbol{\theta}}(c, o)$ and the log of these counts:

$$J(\boldsymbol{\theta}) = \sum_{c=1}^{V} \sum_{o=1}^{V} \left(f_{\boldsymbol{\theta}}(c, o) - \log C_{c,o}\right)^2$$

We could use something fancy for $f_{\boldsymbol{\theta}}$, but let's go with something simple:

$$f_{\boldsymbol{\theta}}(c, o) = \mathbf{u}_c^\top \mathbf{v}_o + b_c + c_o$$

where b_c and c_o are bias terms for the centre and context words, respectively.

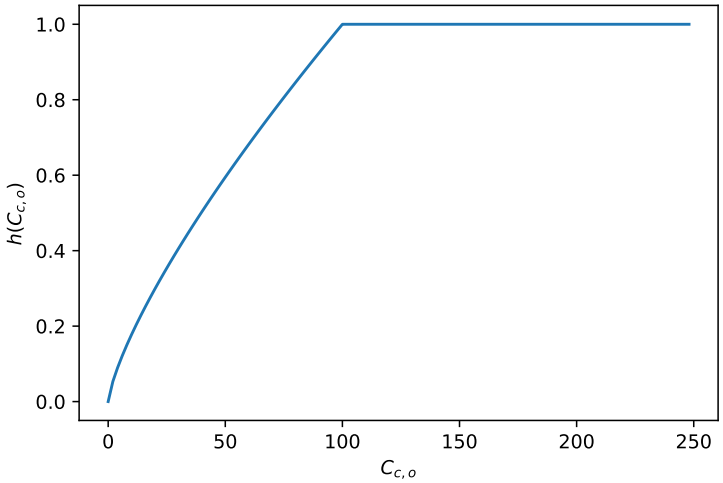
We might also not care that much about word pairs with very few counts, so we can weigh these:

$$\boxed{J(\boldsymbol{\theta}) = \sum_{c=1}^{V} \sum_{o=1}^{V} h(C_{c,o}) \; \left(f_{\boldsymbol{\theta}}(c, o) - \log C_{c,o}\right)^2}$$

where $h(x)$ is a weight function.

A suggested choice is

$$h(x) = \begin{cases} \left(\frac{x}{100}\right)^{0.75} & \text{if } x < 100 \\ 1 & \text{otherwise} \end{cases}$$



Since $h(0) = 0$, the squared loss is only calculated over word pairs where $C_{c,o} > 0$. This makes training way faster compared to considering all possible word pairs.

The original paper (Pennington et al., 2014) notes that either the centre word embedding vector $\mathbf{v}$ or context vectors $\mathbf{u}$ can be used as word embeddings. In the paper they actually sum the two to get the final embedding.

Further reading

There are some other interpretations of the GloVe loss. E.g. you can make some (loose, intuitive) connections with the skip-gram loss itself.

Evaluating word embeddings

Qualitative evaluation

- Consider the closest word embeddings to some query words (as we did for the skipgram model
- Visualise the embeddings in two or three dimensions using dimensionality reduction:
  - PCA
  - t-SNE
  - UMAP

Skipgram embeddings visualised with UMAP using https://projector.tensorflow.org/:

Extrinsic evaluation

- Build and evaluate a downstream system for a real task
- E.g. text classification or named entity recognition
- Slow: Need to build and evaluate system
- Sometimes unclear whether there are interactions between subsystems, making a single subsystem difficult to evaluate
- But on the other hand also the best real world test setting

Intrinsic evaluation

- Word analogy tasks (weird)
- Correlation with human word similarity judgments

Intrinsic: Word analogy

A word analogy is specified as

$$a : b \quad :: \quad c : d$$

Task: Given a, b, c find d

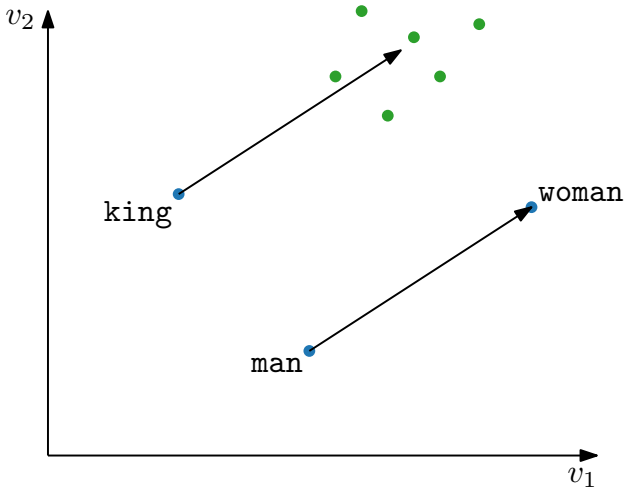
Example:

$$\texttt{man} : \texttt{women} \quad :: \quad \texttt{king} : ?$$

To solve the task using some word embedding approach, we find the word with vector closest to

$$\mathbf{v}_c + (\mathbf{v}_b - \mathbf{v}_a)$$

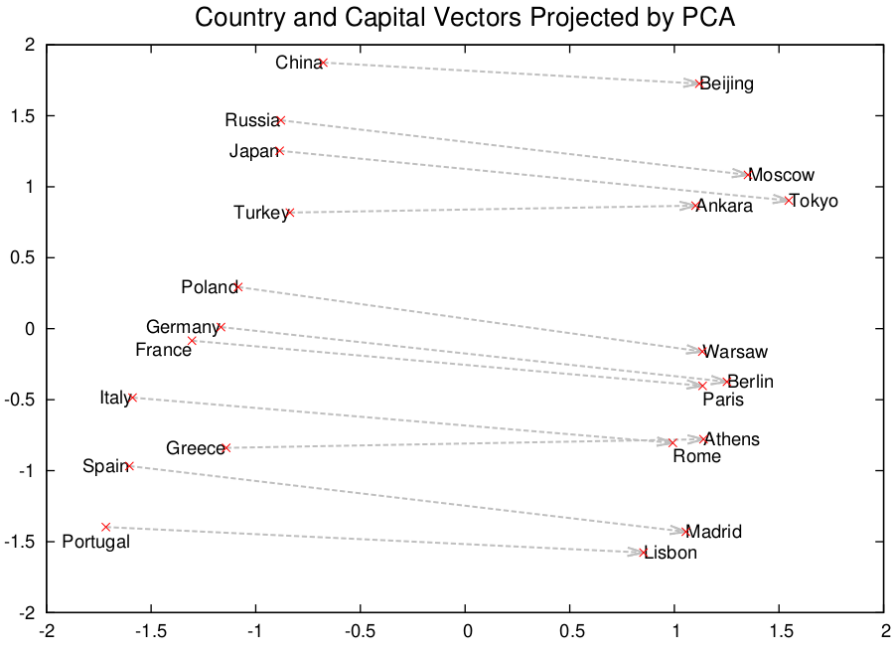
and check if this matches a ground truth labelling.



Cosine distance are often used to determine the closest word embedding.

One problem is that information might not be encoded linearly in the word embeddings.

Visualisation of skipgram embeddings with negative sampling (Mikolov et al., 2013b):



Intrin.: Correlation with human judgments

Have humans scale how similar word pairs are.

Examples from WordSim353:

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10.00
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Compare this to the similarity assigned by a word embedding approach. Normally Spearman's rank correlation coefficient is used as metric.

A comparison of different word embedding approaches on WordSim353 (Pennington et al., 2014):

Model	Size	Correlation (%)
SVD	6B	35.3
SVD-S	6B	56.5
SVD-L	6B	65.7
CBOW	6B	57.2
Skip-gram	6B	62.8
GloVe	6B	65.8
SVD-L	42B	74.0
GloVe	42B	**75.9**
CBOW	100B	68.4

Exercises

Exercise 1: The skip-gram loss function and cross-entropy

For a centre word c, the skip-gram model outputs a vector $\boldsymbol{f}_{\boldsymbol{\theta}}(w_t = c) \in [0,1]^V$. Let's represent the target context word o as a one-hot vector $\mathbf{y}$. Show that the loss for the skip-gram model (without negative sampling) can be written as the cross-entropy between $\mathbf{y}$ and $\boldsymbol{f}_{\boldsymbol{\theta}}(w_t = c)$, if you treat these as discrete distributions over the V words in the vocabulary.

Videos covered in this note

- Why word embeddings? (9 min)
- One-hot word embeddings (6 min)
- Skip-gram introduction (7 min)
- Skip-gram loss function (8 min)
- Skip-gram model structure (8 min)
- Skip-gram optimisation (10 min)
- Skip-gram as a neural network (10 min)
- Skip-gram example (2 min)
- Continuous bag-of-words (CBOW) (6 min)
- Skip-gram with negative sampling (16 min)
- GloVe word embeddings (12 min)
- Evaluating word embeddings (21 min)

References

Y. Goldberg and O. Levy, "word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method," *arXiv*, 2014.

C. Manning, "CS224N: Introduction and word vectors," *Stanford University*, 2022.

C. Manning, "CS224N: Word vectors, word senses, and neural classifiers," *Stanford University*, 2022.

T. Mikoliv, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NeurIPS*, 2013b.

J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *EMNLP*, 2014.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2021.