Word embeddings

Chloé Clavel,
chloe.clavel@telecom-paristech.fr,

Telecom ParisTech, France

Objectives of the Lecture

At the end of the lecture and the lab

you will master :

- the underlying principles of word embeddings
- the algorithm of the most popular word embeddings tool word2vec

you will be able :

- to go further knowing the existing variants and tools

Outline of the course

Plan du cours

Main idea of word embeddings

Learning distributed [1] representations :
word (or sentence) $\rightarrow$ vector (dense)
Example :

$$\mathit{linguistics} = \begin{bmatrix} 15 \\ 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

1. Each entity corresponds to a pattern of activity distributed over computing elements. Each computing element is involved in representing different entities (`https://web.stanford.edu/~jlmcc/papers/PDP/Chapter3.pdf`

2

Main idea of word embeddings

A vector which embeds the semantic link between words

(Mikolov et al., NAACL HLT, 2013)

Underlying principle : distributional semantics

- **Hypothesis :** 2 words occurring in a same context have a semantic proximity
- Example : $\overbrace{\textbf{My favorite fruit is an}}^{\textit{context}}$ *APPLE*.
  $\overbrace{\textbf{My favorite fruit is an}}^{\textit{context}}$ *ORANGE*.
  $\Rightarrow$ High semantic similarity between ORANGE and APPLE (both are fruits) because they tend to appear in a similar context in texts. $\Rightarrow$ $vect_{ORANGE} \sim vect_{APPLE}$

Underlying principle : distributional semantics

Contextual information is sufficient to obtain a viable representation of words

- "For a large class of cases [...] the meaning of a word is its use in the language." Wittgenstein (Philosophical Investigations, 43 - 1953)
- "You shall know a word by the company it keeps", Firth ("A synopsis of linguistic theory 1930-1955." - 1957)

Matrix of co-occurences

In order to make neighbors represent words

NxN matrix where N is the size of the vocabulary

$$cooccurrenceMatrix(i, j) = \text{number of occurences of } word_i \text{ beside } word_j \text{ (co-occurrence)}$$

PRACTICE : build the co-occurrence matrix of the following example corpus

- I like deep learning.
- I like NLP.
- I enjoy flying.

Matrix of co-occurences

1) I like deep learning. 2) I like NLP. 3) I enjoy flying.

$$cooccurenceMatrix = \begin{bmatrix} counts & I & like & enjoy & deep & learning & NLP & flying & . \\ I & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ like & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ enjoy & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ deep & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ learning & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ NLP & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ flying & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ . & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Matrix of co-occurences

Remarks

- The matrix is symetric
- You can augment the window for counting the co-occurrences $cooccurrenceMatrix(i, j)$ = number of cooccurences of $word_i$ and $word_j$ in a window of size n (more common 5-10)
  $\Rightarrow$ captures both syntactic and semantic information
- Other option : window = whole document $\Rightarrow$ give general topics (e.g. all sport terms will have similar entries)
- Increase in size with vocabulary $\Rightarrow$ require a lot of storage
- sparsity for the subsequent classification models $\Rightarrow$ low robustness

Solution to the dimensionality issue

How to reduce the dimensionality ?

- Store most of the important information in a fixed, small number of dimensions : **a dense vector** (usually 25-1000 dimensions)
- use SVD – Singular Value Decomposition of cooccurrence matrix

SVD – Singular Value Decomposition of cooccurrence matrix

$$X_{NxN} = U_{NxR} S_{RxR} V^T_{RxN}$$

- S_{RxR} is the diagonal matrix storing the singular values in its diagonal.
- We can reduce its dimension by keeping the first k singular values (according to the desired percentage variance captured).
- word vector = the rows of U_{Nxk} (a dense vector)

Problem with SVD

- Computational cost
- Bad for millions of words or documents
- Hard to incorporate new words or documents

Main Idea of word2vec

- Idea : Directly learn low-dimensional word vectors
- Instead of computing cooccurrence counts $\rightarrow$ predict surrounding words of every word
- Faster and can easily incorporate a new sentence/ document or add a word to the vocabulary

Neural Language models

Inspiration : Neural Probabilistic language models of Bengio et al. 2003

- Language model : predicting upcoming words from prior word context
- Neural (Feed forward [a]) probabilistic models :
  - maximize the probability of the next word w_t given the previous words h (for history) (maximum likelihood (ML) principle)
  - in terms of a softmax function

a. the information moves in only one direction, forward, from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network such as in RNN.

Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3 :1137-1155, 2003.

2

Overview of word embeddings computation

What do you need in order to train word vectors ? a big dataset of texts

What will you obtain ? if V is the vocabulary of the dataset, for each word $w_i \in V$, a dense vector v is computed

$$\text{Example : } \textit{linguistics} = \begin{bmatrix} 15 \\ 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

How ? using neural networks

Overview of word embeddings computation

These representations are very good at encoding dimensions of similarity !
Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

- syntactically : $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
- semantically : $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Plan du cours

Two types of NN architecture :

- CBOW (Continuous Bag Of Words)
- Skip-gram

CBOW (Continuous Bag Of Words)

- Input of the network : context
  $INPUT = \{$**April, 1959, recorded, what**$\}$
- Output of the network : target $OUTPUT =$ **Davis**
- learn how to predict a word according to its context

Definition - Context of a word

Context of a word :

- the context of a word is the set of the C surrounding words.
- Example : "In March and $\overbrace{\text{April 1959,}}^{\text{left context}}$ **Davis** $\overbrace{\text{recorded what}}^{\text{right context}}$ many critics consider his greatest album, Kind of Blue"
- The $C = 2$ left-right context of word **Davis** is represented as follows :

$$\{April, 1959, recorded, what\}$$

- Observation : the order of the words in the context has no influence in word embeddings computation

Definition - target word

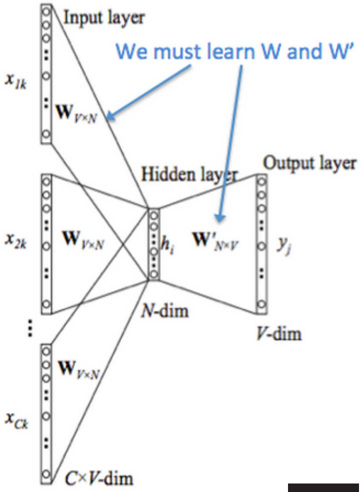
"In March and $\overbrace{\text{April 1959}}^{\text{left context}}$, **Davis** $\overbrace{\text{recorded what}}^{\text{right context}}$ many critics consider his greatest album, Kind of Blue"

Target :

DAVIS the word from which we want to compute the vector

CBOW (Continuous Bag Of Words)

- use a two-layer-NN (hidden layer, output layer)
- learn W et W' so that if $INPUT =$ {**April, 1959, recorded, what**}, $OUTPUT =$ **Davis**



Inputs and outputs of NN architecture

- words are represented by one-hot vectors.
- "one-hot" comes from digital circuit design
- a **One-hot** vector is an $\mathbb{R}^{|V|}$ vector ($|V|$, size of the vocabulary)
- each word is indexed in the sorted English language (alphabetic order).

- 0 at each vector component except at the index of that word (one 1)

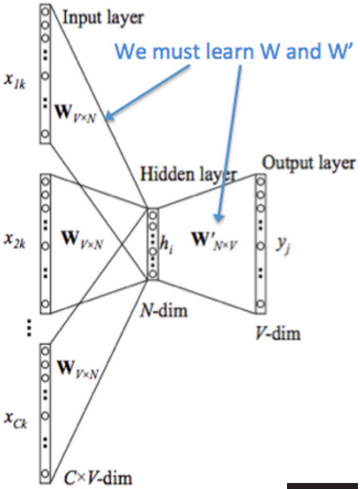
$$at \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

Questions

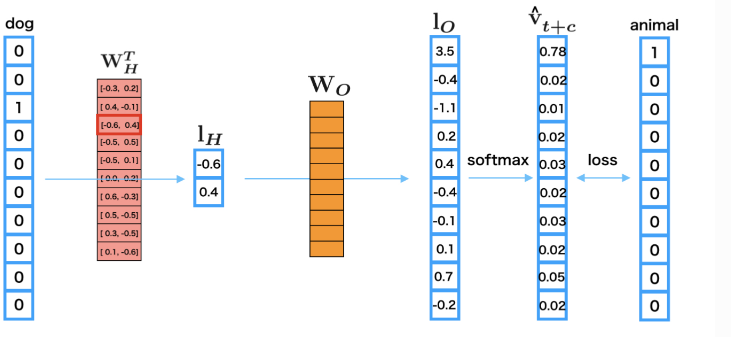
What do we need to start learning the word vectors ?
What are the inputs/outputs of the NN in the CBOW model ?
How are we representing the words in these inputs/outputs ?

Training the NN for CBOW

- Input of the network : one hot vectors of the *context* words : $(x^{c-m}, ..., x^{c-1}, x^{c+1}, ..., x^{c+m}) \in \mathbb{R}^{|V|}$ (size m)
- Wanted outputs of the network : y one hot vector of the target
- The model learns W et W' so that the output of the NN, $\hat{y}$, matches y



Overview



from https://docs.chainer.org/en/stable/examples/word2vec.html#main-algorithm

Training the NN for CBOW

- The model learns W et W' so that the output of the NN, $\hat{y}$, matches y
- Choice of the loss measure : cross-entropy

$$H(\hat{y}, y) = -\sum_{j=1}^{\|V\|} y_j log(\hat{y}_j)$$

- PRACTICE : what is vector y ? Simplify the expression of the loss

Training the NN for CBOW

$$H(\hat{y}, y) = -\sum_{j=1}^{\|V\|} y_j log(\hat{y}_j)$$

As $y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$, $H(\hat{y}, y) = -y_c log(\hat{y}_c) = -log(\hat{y}_c)$

Training the NN for CBOW

$$H(\hat{y}, y) = -log(\hat{y}_c)$$

PRACTICE

- Compute the loss when the prediction is correct
- Compute the loss when the the prediction is bad *e.g.* $\hat{y}_c = 0.01$

Training the NN for CBOW

PRACTICE

- If the prediction is correct $\hat{y}_c = 1$ and $H(\hat{y}, y) = -log(\hat{y}_c) = 0$
- If the prediction is bad *e.g.* $\hat{y}_c = 0.01$,
  $H(\hat{y}, y) = -log(\hat{y}_c) = -log(0.01) = 4.605$

Training the NN for CBOW

- Learning objective : update W et W' in order to minimize the loss

$$H(\hat{y}, y) = -log(\hat{y}_c)$$

- $W \in \mathbb{R}^{Nx|V|}$ and $W' \in \mathbb{R}^{|V|xN}$,
- N is the size of the embedding space

Training the NN for CBOW

The learnt W and W' correspond to the word embeddings

- The j^{th} column of W is the embedded vector for $word_j$, when it is an input of the model
- The i^{th} row of W' is the embedded vector for $word_i$, when it is an output of the model
- Note that we do in fact learn two vectors for every word

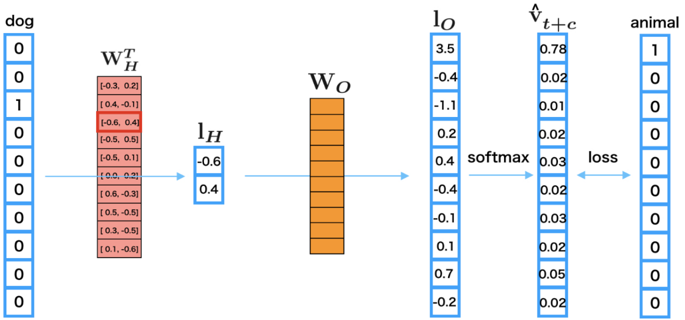
Note : in the SVD of cooccurrence matrix, word vector = the rows of $U_{|V|xN}$ (a dense vector)

Remark :

Once the embeddings are learned, we'll have two embeddings for each word wi (we have learnt W and W')

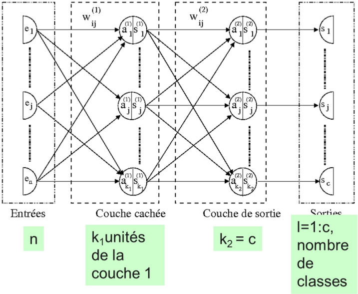
- We can choose to throw away the W' matrix and just keep W
- Alternatively we can add the two embeddings together, using the summed embedding as the new N-dimensional embedding,
- or we can concatenate them into an embedding of dimensionality 2N.

Overview



from `https://docs.chainer.org/en/stable/examples/word2vec.html#main-algorithm`

Reminder : ML NN Layers

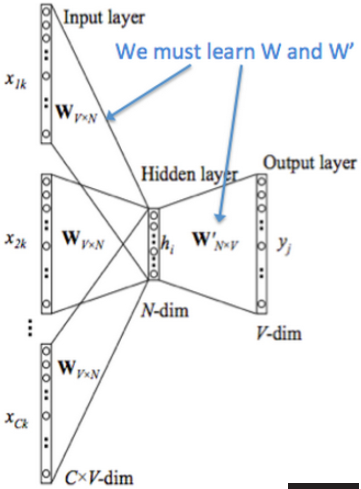


FORWARD : we need to compute all the outputs of the $m-1$ layer to compute the outputs of the m layer.

FORWARD : How to express $\hat{y}$?

$\hat{y}$ (that will be compared to $y = x_c$) is computed according to

- $(x^{c-m}, ..., x^{c-1}, x^{c+1}, ..., x^{c+m}) \in \mathbb{R}^{|V|}$
- $W \in \mathbb{R}^{Nx|V|}$ and $W' \in \mathbb{R}^{|V|xN}$,
- N is the size of the embedding space



2

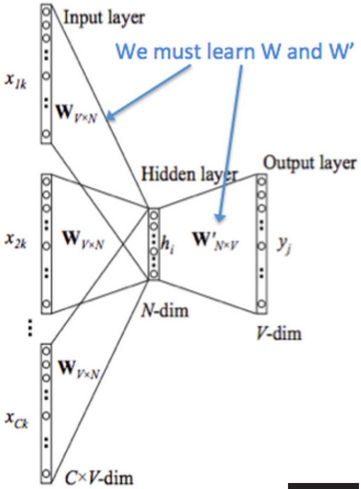
FORWARD : How to express $\hat{y}$?

1) Hidden layer (activation function : identity) :

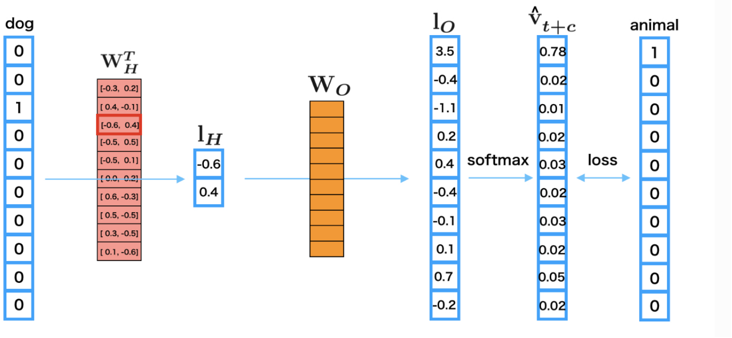
$$v_{c-m} = W * x^{c-m}, \ldots, v_{c-1} = W * x^{c-1},$$

$$v_{c+1} = W * x^{c+1}, \ldots, v_{c+m} = W * x^{c+m}$$

(Express the word embeddings vectors of the inputs)
PRACTICE : what is the dimension of v_{c-k} ?



Overview

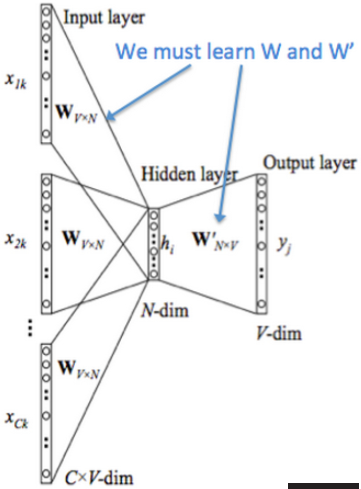


from https://docs.chainer.org/en/stable/examples/word2vec.html#main-algorithm

FORWARD : How to express $\hat{y}$?

2) Hidden layer : Average these vectors to get the vector $h = \hat{v} \in \mathbb{R}^N$

$$\hat{v} = mean(v^{c-m}, ..., v^{c-1}, v^{c+1}, ..., v^{c+m})$$



FORWARD : How to express $\hat{y}$?

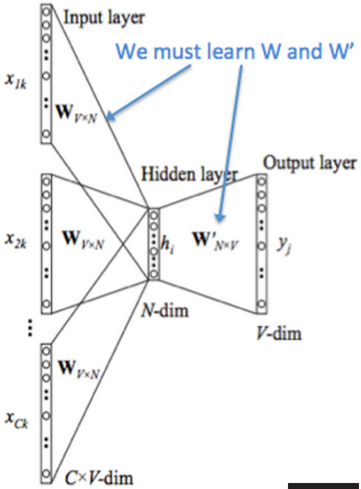
3) OUTPUT LAYER (activation function is softmax)

$$\hat{y} = softmax(W' * \hat{v}),\ \hat{y} \in \mathbb{R}^{|V|}$$

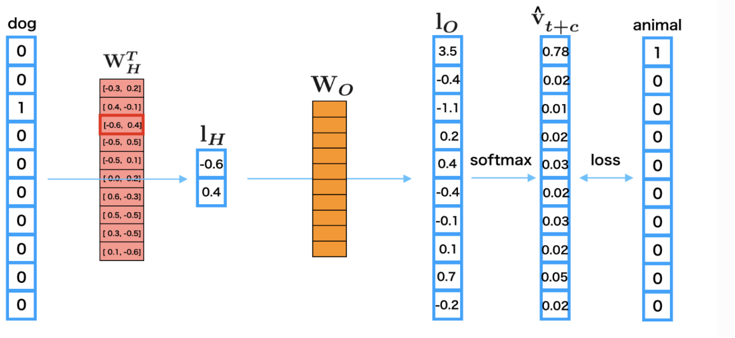
Reminder : $softmax(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

Then, $\hat{y}_j = \frac{e^{W'_j * \hat{v}}}{\sum_{k=1}^{|V|} e^{W'_k * \hat{v}}}$, where W'_j is the j^{th} line of W'

can be viewed as the probability of jth word to appear with the context words



Overview



from https://docs.chainer.org/en/stable/examples/word2vec.html#main-algorithm

How to compute the loss according to W and W'

- $H(\hat{y}, y) = -log(\hat{y}_c)$ (the probability of target word to appear with the context words)
- $\hat{y}_c = softmax(W'_c * \hat{v})$,
- $H(\hat{y}, y) = -log(softmax(W'_c * \hat{v})) = -log(\frac{exp(W'_c * \hat{v})}{\sum_{j=1}^{|V|} e^{W'_j * \hat{v}}})$

  $H(\hat{y}, y) = -W'_c * \hat{v} + log(\sum_{j=1}^{|V|} e^{W'_j * \hat{v}})$
- where :
  $\hat{v} = mean(W * x^{c-m}, ..., W * x^{c-1}, W * x^{c+1}, ..., W * x^{c+m})$
  and W'_j is the j^{th} line of W'

Reminder : training and backpropagation algorithm

1. define the loss
2. compute partial derivatives
3. Iteration on all *context*, *target* pairs of the corpus and apply gradient descent algorithm from output layers to input layers

Word2vec learns embeddings by starting with an initial set of embedding vectors and then iteratively shifting the embedding of each word w

Training the NN for CBOW

CBOW Model : how to find W and W' that minimize the loss

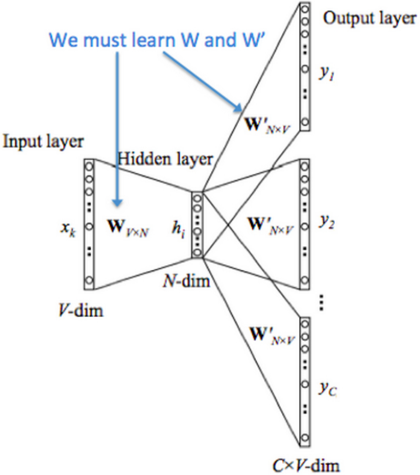
- $J(W, W') = H(\hat{y}, y) = -W'_c * \hat{v} + log(\sum_{j=1}^{|V|} e^{W'_j * \hat{v}})$
- where $\hat{v}$ depends on W
- use gradient descent to update W and W'

Skip-gram

- Input of the network : target
- Output of the network : context words
- Learn how to predict a context given a word/Predict surrounding words in a window of length m of every word.

Skip-gram

- Learn W et W' so that if $INPUT =$ **Davis**, $OUTPUT =$ {**April, 1959, recorded, what**}



2

Reminder

What do we need to start learning the word vectors ?

Skip-gram in practice

Dataset : "The quick brown fox jumped over the lazy dog"
For **Skip-gram** , the task is to predict :

- 'the' and 'brown' from 'quick'
- 'quick' and 'fox' from 'brown'
- *etc.*

Skip-gram

1. one hot input vector of the center word : x
2. compute the embedded word vector for the center word $v_c = Wx$
3. no averaging (because one input), just set $\hat{v} = v_c$
4. Generate $2m$ score vectors, $u_{c-m}, \ldots, u_{c-1}, u_{c+1}, \ldots, u_{c+m}$ using $u = W' v_c$
5. Turn each of the scores into probabilities, $\hat{y} = softmax(u)$
6. We desire our probability vector generated to match the true probabilities which is $y(c-m), \ldots, y(c-1), y(c+1), \ldots, y(c+m)$, the one hot vectors of the actual outputs.

Plan du cours

Underlying principles of word embeddings

NN architecture for the computation of word vectors

References

References for this lecture

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.
Lecture from Stanford
`http://cs224d.stanford.edu/lecture_notes/notes1.pdf`
Tools : word2vec from Google
`https://code.google.com/p/word2vec/` tutorial from tensorflow `https://www.tensorflow.org/tutorials/word2vec`
Other representation : Glove
`http://nlp.stanford.edu/projects/glove/`

To go further : Fasttext

P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information

- skipgram model, where each word is represented as a bag of character n-grams
- compute word representations for words that did not appear in the training data.
- integrate morphological information

To go further : contextualized word embeddings

State of the art model : BERT : Bi-directional Encoder Representations from Transformers
Devlin, Jacob, et al. "Bert : Pre-training of deep bidirectional transformers for language understanding."

To go further : Document/sentence level representations

DOC2VEC https://arxiv.org/pdf/1607.05368.pdf and gensim tool : https://radimrehurek.com/gensim/lrec2010_final.pdf
SENTENCE EMBEDDINGS Article de Felix Hill : "Learning Distributed Representations of Sentences from Unlabelled Data"