Word2vec very successfully implemented word embeddings using this context-meaning idea.

- We start with a very large corpus of text (e.g. all of Wikipedia)
- Every word is represented by a vector in $\mathbb{R}^n$ space (n~200 dimensions)
- You have a model (e.g. a NN) that tries to predict the vector of a word (i.e. the central word) given the vectors of the words around it (i.e. its context). In probability terms, the NN models the probability $P( w_c | w_{c-3}, w_{c-2}, w_{c-1}, w_{c+1}, w_{c+2}, w_{c+3} )$
- Go through each central word - context pair in the corpus
- In each iteration, modify the NN and vectors a little bit for words with similar contexts to have similar vectors
- Repeat last 2 steps many times
There are two versions of word2vec:

- CBOW = A NN learn to model the central word $P(w_c | w_{c-3} \ldots)$
- Skip-gram = A NN learn to model the context $P(w_{c+i} | w_c)$
Word2vec: Skip-gram

- The NN uses a single hidden layer, a single weight matrix ($W_{VD}$), the transpose of this weight matrix ($W^T_{VD}$), and a single activation function (softmax)
Word2vec: Skip-gram

\[
\begin{bmatrix}
0 & 0 & 1 & 0 \\
23 & 5 & 7 & 0 \\
4 & 6 & 13 & 0 \\
10 & 12 & 19 & 0 \\
11 & 18 & 25 & 0 \\
\end{bmatrix} \times \begin{bmatrix}
17 & 24 & 1 \\
23 & 5 & 7 \\
4 & 6 & 13 \\
10 & 12 & 19 \\
11 & 18 & 25 \\
\end{bmatrix} = \begin{bmatrix}
10 & 12 & 19 \\
\end{bmatrix}
\]
Word2vec: learning

- OK, awesome, then we use a NN to implement the model... but how do we learn the model? How do we find the values of $W_{VD}$? This is the matrix that contains all our vector embeddings.

- We apply stochastic gradient descent (SGD) on a very very large corpus:
  - Go through each central word - context pair in the corpus
  - In each iteration, modify the NN and vectors a little bit for words with similar contexts to have similar vectors
  - Repeat last 2 steps many times

- But there is a problem: The denominator of

$$\frac{\exp(\sum_{v} [x] [W_{VD}] [W_{VD}]^T)}{\sum_v \exp(\sum_{v} [x] [W_{VD}] [W_{VD}]^T)} = [y] \quad \text{..., or:} \quad \frac{\sum_{v} \exp(\sum_{v} [x] [W_{VD}] [W_{VD}]^T)}{\sum_i \exp(\sum_i [w] [w]^T)} = y(v)$$
The normalization factor is computationally too expensive:

\[
\frac{\exp\left(\frac{[x] \times [W] \times [W^T]}{v_d} \right)}{\sum_{v} \exp\left(\frac{[x] \times [W] \times [W^T]}{v_d} \right)} = y \quad \sum_{i} \exp\left(\frac{\tilde{w}_i \times \tilde{w}_a}{v_d} \right) = y(v)
\]

To bypass this problem, rather than calculating the denominator exactly, we calculate and approximation of it by using “negative sampling”

Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word).

We take N negative samples (using word probabilities).

Maximize probability that real outside word appears, minimize probability that random words appear around central word.

“Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
The main end result of word2vec (and other embedding algorithms) is that words of similar meaning end up being placed nearby to each other in $\mathbb{R}^D$ space.
A surprising side result is that specific directions in $\mathbb{R}^D$ space also encode meaning. Both properties are very useful for building further ML on top of the embeddings.
Co-occurrence matrices

- But if all what word2vec is doing is extracting the meaning of words from their context, why don't we simply count how often words appear together?
- This is called a “co-occurrence matrix”, and it also gets quite far capturing the meaning of words.
- As the NN in word2vec, this very simple matrix of co-occurrence counts can be calculated:
  - Go through each central word - context pair in the corpus (context window length is commonly anything between 1 and 5)
  - In each iteration, update in the row of the count matrix corresponding to the central word by adding +1 in the columns corresponding to the context words
  - Repeat last 2 steps many times
Co-occurrence matrices

Example corpus:
- ... after a few days the fur of the dog was unkept and dirty, and this spread ...
- ... you will soon realise that walks in the park are dog’s priority, and they will ...
- ... he was worried that his neighbour’s dog kept barking during all night ...
- ... the warden had a labrador of brown fur who kept chasing squirrels in ...
- ... breeders recommend to daily take your labrador to the park to tempter ....
- ... at the end of the day, a labrador barks less than other breeds, but he also ...
## Co-occurrence matrices

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>dog</th>
<th>labrador</th>
<th>fur</th>
<th>park</th>
<th>bark</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>23</td>
<td>4</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>4</td>
<td>28</td>
<td>23</td>
<td>13</td>
<td>22</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>labrador</td>
<td>0</td>
<td>23</td>
<td>25</td>
<td>16</td>
<td>23</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>fur</td>
<td>12</td>
<td>13</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>park</td>
<td>0</td>
<td>22</td>
<td>23</td>
<td>0</td>
<td>21</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>bark</td>
<td>0</td>
<td>28</td>
<td>22</td>
<td>0</td>
<td>3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
While word2vec was able to encode the meaning of the words in a matrix $W_{vD}$ of size $V \times D$, the co-occurrence $C_{vv}$ matrix has size $V \times V$

- But there are some problems with this basic approach:
- The co-occurrence matrix increase very fast (as $V^2$) with the size of the vocabulary
- This requires a lot of storage
- This requires a lot of computation (e.g. to calculate distance between words)
- Rare words will have very few counts
- ML algorithms built on top of the co-occurrence matrix have sparsity problems
Solution: Store the most important information in a fixed small number of dimensions (usually 25-1000, as in word2vec), rather than in V dimensions.

There are many methods to reduce dimensionality while preserving the most important information: Principal Component Analysis, Independent Component Analysis...

Une commonly used to reduce $C_{vv}$ is Singular Value Decomposition (SVD):

$$[M] = [U][\Sigma][V]$$

$$[C] = [U][\Sigma][V]$$

Why do we use this very complicated method?
Co-occurrence matrices

- If $M_{mn}$ were a linear transformation “$M: \mathbb{R}^v \rightarrow \mathbb{R}^v$” then:
  - $U = \text{rotates the basis of } M: \mathbb{R}^v \rightarrow \mathbb{R}^v$
  - $\Sigma = \text{rescales the basis}$
  - $V = \text{rotates the basis again}$

$$M = U \cdot \Sigma \cdot V^*$$
And the magic comes from the fact that, if we decompose $M_{mn}$ in its singular values

$$C_{vv} \rightarrow U_{vv} \times \Sigma_{vv} \times V_{vv}$$

keep only the ‘d’ largest of such singular values

$$C_{vv} \rightarrow U_{vv} \times \Sigma_{vv} \times V_{vv} \rightarrow U_{vv} \times \Sigma_{vd} \times V_{dd}$$

and recalculate $M_{mm}$

$$U_{vv} \times \Sigma_{vd} \times V_{dd} \rightarrow C_{vd}$$

We obtain the closest approximation of $M_{mm}$ according to mean square error
Co-occurrence matrices

“An improved model of semantic similarity based on lexical co-occurrence” Rohde, 2005
Co-occurrence matrices

➢ Count based models:
- LSA, HAL (Lund & Burges)
- COALS, Hellinger-PCA (Rohde, Lebert & Collobert)
- Fast training
- Efficient use of statistics
- Primarily used to capture word similarity
- Disproportionate important given to large counts

➢ NN based models:
- Skip-gram, CBOW (Mikolov)
- NNLM, HLBL, RNN (Bengio, Collobert & Weston, Huang, Mnih & Hinton)
- Scales with corpus size
- Inefficient use of statistics
- Gives improved performance on other tasks
- Can capture complex patterns beyond word similarity
Glove

- Rather than word counts $C_w$, people often used the probabilities of one word appearing in the context of another: $P(v_1|v_2)$

- Both word counts in $C_w$ and probabilities $P(v_1|v_2)$, depend very strongly on the frequency of words: frequent words will have much larger counts and probabilities

- The authors of Glove suggest that ratios of probabilities between words are much better suited to create good embeddings

- The authors of Glove introduce 2 further heuristic arguments:
  - The distance between words $d(v_1, v_2)$ should be a linear function
  - The Distance between words should be symmetric between context and central words. Namely $d(v_1, v_2)$ when $v_1$ is a central word and $v_2$ a context word should be the same than $d(v_1, v_2)$ when $v_1$ is a context word and $v_2$ a central word
The embeddings $w_i$ that best fulfill those three rules, are those whose scalar product approximates $\log(C_{vv})$ minus two constant values that depend only on the 2 words being multiplied:

$$w_i^T w_k + b_i + b_k \sim \log(C_{vv}[i,k])$$

You can obtain these embeddings $w_i$ for all words by minimising the error function:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

The function $f(\ldots)$ is also defined heuristically →
Glove

- You can obtain these embeddings \( w_i \) for all words by minimising the error function:

\[
\sum_{i,j=1}^{V} f(C_{ij}) \left( \begin{bmatrix} w_i^T \\ v_1 \end{bmatrix} \begin{bmatrix} w_j \\ 1 \end{bmatrix} + b_i + b_j - \log(C_{ij}) \right)^2
\]

\[
J = \sum_{i,j=1}^{V} f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + b_j - \log X_{ij} \right)^2
\]

- The advantages of Glove:
  - Fast training
  - Scalable to huge corpora
  - Good performance even with small corpus and small vectors
Evaluating quality of embeddings

- **Intrinsic methods**: Measure some statistical property of the embeddings that should correlate with quality (e.g. similar words should be close to each other)
  - Fast to compute
  - Helps to understand the system
  - The method does not fully ensure that the embeddings are going to perform well when sent to another real world task

- **Extrinsic methods**: Use the embeddings in a real NN and on a real task to evaluate embeddings (e.g. part of speech tagging)
  - Slow to compute
  - Unclear what part of the performance on the real task comes from the embedding and which part comes from the rest of the NN... and which from the embeddings - NN interaction
Evaluating quality of embeddings

Intrinsic methods:

➢ Word Vector Analogies:
  - \( a \) is to \( b \) as \( c \) is to \( x \)
  - \textit{man} is to \textit{woman} as \textit{king} is to \( x \)

➢ Evaluate word vector by how well their cosine distance after addition captures intuitive semantic and syntactic analogies

➢ Discarding the input words from the search

➢ Problem: What if the information is there but it is nonlinear
Evaluating quality of embeddings
Evaluating quality of embeddings
Evaluating quality of embeddings

Intrinsic methods:

- There are datasets available to run intrinsic evaluation:
  - [https://github.com/nicholas-leonard/word2vec/questions-words.txt](https://github.com/nicholas-leonard/word2vec/questions-words.txt) → Word relationships
  - [https://github.com/nicholas-leonard/word2vec/blob/master/questions-phrase.txt](https://github.com/nicholas-leonard/word2vec/blob/master/questions-phrase.txt) → phrase relationships

Athens Greece Bangkok Thailand  cup - coffee → 6.6
Fresno California Anchorage Alaska cup - article → 2.4
free freely usual usually  Noon - string → 0.5
clear unclear certain uncertain  Midday - noon → 0.3
Evaluating quality of embeddings

Figure 4: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus (Wikipedia 2014 + Gigaword 5) with the same 400,000 word vocabulary, and use a symmetric context window of size 10.
Evaluating quality of embeddings

Glove: Global vectors for word representation. Pennington et al. EMNLP. 2014
Evaluating quality of embeddings

**Extrinsic methods:**

- Typical benchmark tasks:
  - Named Entity Recognition
  - Parts Of Speech tagging
  - Sentiment analysis
  - Translation
  - ... basically, anything meaningful

---

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
<th>ACE</th>
<th>MUC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete</td>
<td>91.0</td>
<td>85.4</td>
<td>77.4</td>
<td>73.4</td>
</tr>
<tr>
<td>SVD</td>
<td>90.8</td>
<td>85.7</td>
<td>77.3</td>
<td>73.7</td>
</tr>
<tr>
<td>SVD-S</td>
<td>91.0</td>
<td>85.5</td>
<td>77.6</td>
<td>74.3</td>
</tr>
<tr>
<td>SVD-L</td>
<td>90.5</td>
<td>84.8</td>
<td>73.6</td>
<td>71.5</td>
</tr>
<tr>
<td>HPCA</td>
<td>92.6</td>
<td><strong>88.7</strong></td>
<td>81.7</td>
<td>80.7</td>
</tr>
<tr>
<td>HSMN</td>
<td>90.5</td>
<td>85.7</td>
<td>78.7</td>
<td>74.7</td>
</tr>
<tr>
<td>CW</td>
<td>92.2</td>
<td>87.4</td>
<td>81.7</td>
<td>80.2</td>
</tr>
<tr>
<td>CBOV</td>
<td>93.1</td>
<td>88.2</td>
<td>82.2</td>
<td>81.1</td>
</tr>
<tr>
<td>GloVe</td>
<td><strong>93.2</strong></td>
<td>88.3</td>
<td><strong>82.9</strong></td>
<td><strong>82.2</strong></td>
</tr>
</tbody>
</table>

Table 4: F1 score on NER task with 50d vectors. Discrete is the baseline without word vectors. We use publicly-available vectors for HPCA, HSMN, and CW. See text for details.

We have seen:
- Word2vec: Neural networks approach
- Co-occurrence matrices: Counting approach
- Glove: Combining word2vec and co-occurrences
- Evaluation: Intrinsic and extrinsic
Papers =

- A latent variable model approach to PMI-based word embeddings, Arora et al., 2016. [http://aclweb.org/anthology/Q16-1028](http://aclweb.org/anthology/Q16-1028)