An overview of word2vec

Benjamin Wilson

Berlin ML Meetup, July 8 2014

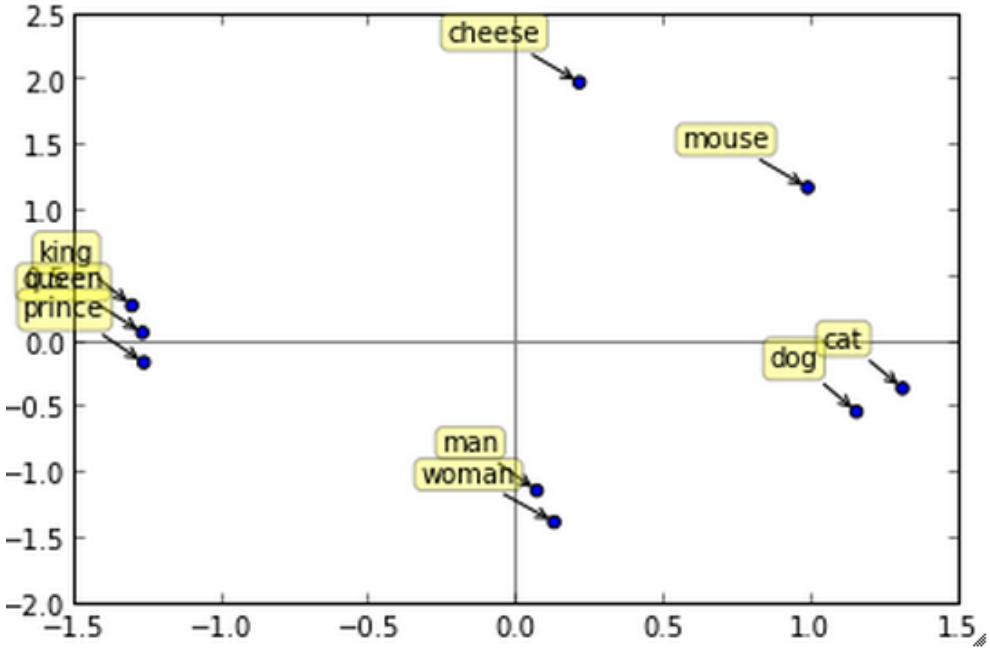
Outline

word2vec associates words to points in space

- word2vec associates words with points in space
- word meaning and relationships between words are encoded spatially
- learns from input texts
- developed by Mikolov, Sutskever, Chen, Corrado and Dean in 2013 at Google Research

Similar words are closer together

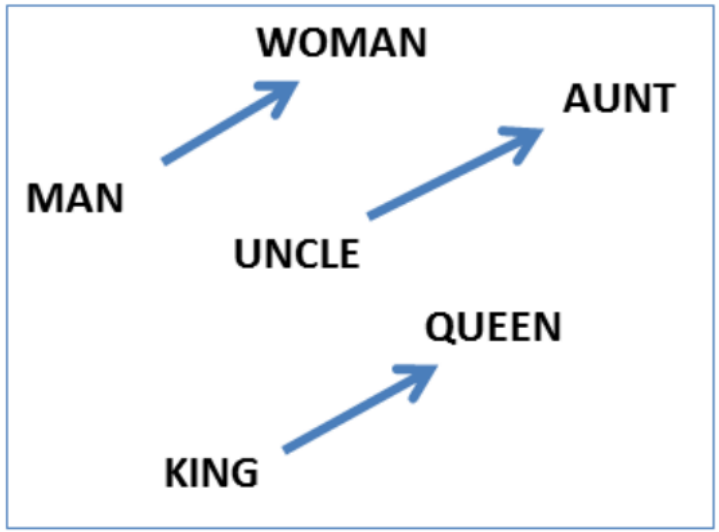
- spatial distance corresponds to word similarity
- words are close together $\Leftrightarrow$ their "meanings" are similar
- notation: word $w \mapsto$ `vec[w]` its point in space, as a position vector.



e.g. `vec[woman]` $= (0.1, -1.3)$.

Word relationships are displacements

- the displacement (vector) between the points of two words represents the word relationship.
- same word relationship $\Rightarrow$ same vector



Source: *Linguistic Regularities in Continuous Space Word Representations*, Mikolov et al, 2013

e.g.

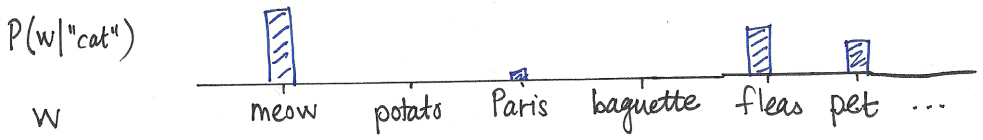
$$\texttt{vec[queen]} - \texttt{vec[king]} = \texttt{vec[woman]} - \texttt{vec[man]}$$

What's in a name?

- How can a machine learn the meaning of a word? Machines only understand symbols!
- Assume the Distributional Hypothesis (D.H.) (Harris, 1954):

  "words are characterised by the company that they keep"

- Suppose we read the word "cat". What is the probability $P(w|\texttt{cat})$ that we'll read the word w nearby?



- D.H. : the meaning of "cat" is captured by the probability distribution $P(\cdot|\texttt{cat})$.

word2vec as shallow learning

- word2vec is a successful example of “shallow” learning
- word2vec can be trained as a very simple neural network
  - single hidden layer with no non-linearities
  - no unsupervised pre-training of layers (i.e. no deep learning)
- word2vec demonstrates that, for vectorial representations of words, shallow learning can give great results.

word2vec focuses on vectorization

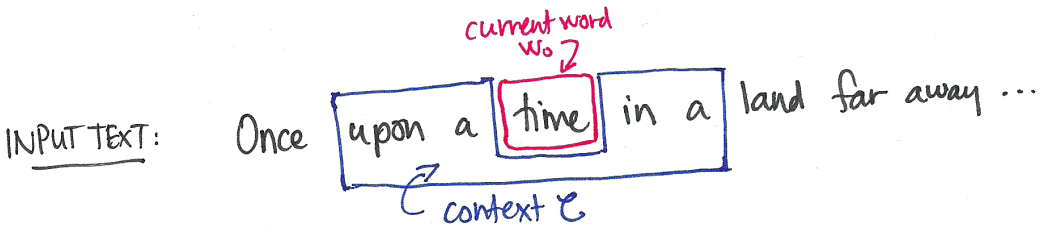
- word2vec builds on existing research
- architecture is essentially that of Minh and Hinton's log-bilinear model
- change of focus: vectorization, not language modelling.

word2vec scales

- word2vec scales very well, allowing models to be trained using *more data*.
- training speeded up by employing one of:
  - hierarchical softmax (more on this later)
  - negative sampling (for another day)
- runs on a single machine - can train a model at home
- implementation is published

Learning from text

- word2vec learns from input text
- considers each word w_0 in turn, along with its context $\mathcal{C}$
- context = neighbouring words (here, for simplicity, 2 words forward and back)



sample #	w_0	context $\mathcal{C}$
1	once	{upon, a}
	...	
4	time	{upon, a, in, a}
	...	

Two approaches: CBOW and Skip-gram

word2vec can learn the word vectors via two distinct learning tasks, CBOW and Skip-gram.

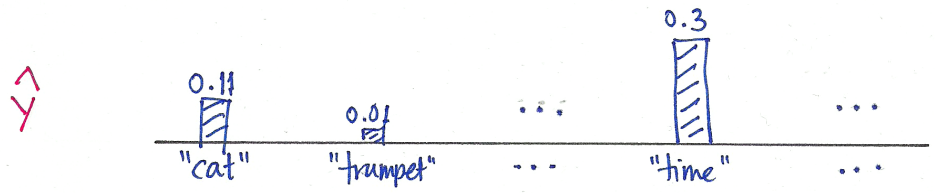
- CBOW: predict the current word w_0 given only $\mathcal{C}$
- Skip-gram: predict words from $\mathcal{C}$ given w_0
- Skip-gram produces better word vectors for infrequent words
- CBOW is faster by a factor of window size – more appropriate for larger corpora
- We will speak only of CBOW (life is short).

CBOW learning task

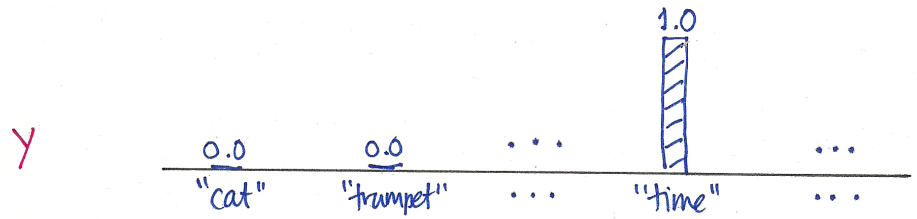
- Given only the current context $\mathcal{C}$, e.g.

$$\mathcal{C} = \{\texttt{upon}, \texttt{a}, \texttt{in}, \texttt{a}\}$$

  predict which of all possible words is the current word w_0, e.g. $w_0 = \texttt{time}$.
- multiclass classification on the vocabulary W
- output is $\hat{y} = \hat{y}(\mathcal{C}) = P(\cdot|\mathcal{C})$ is a probability distribution on W, e.g.



- train so that $\hat{y}$ approximates target distribution y – "one-hot" on the current word, e.g.

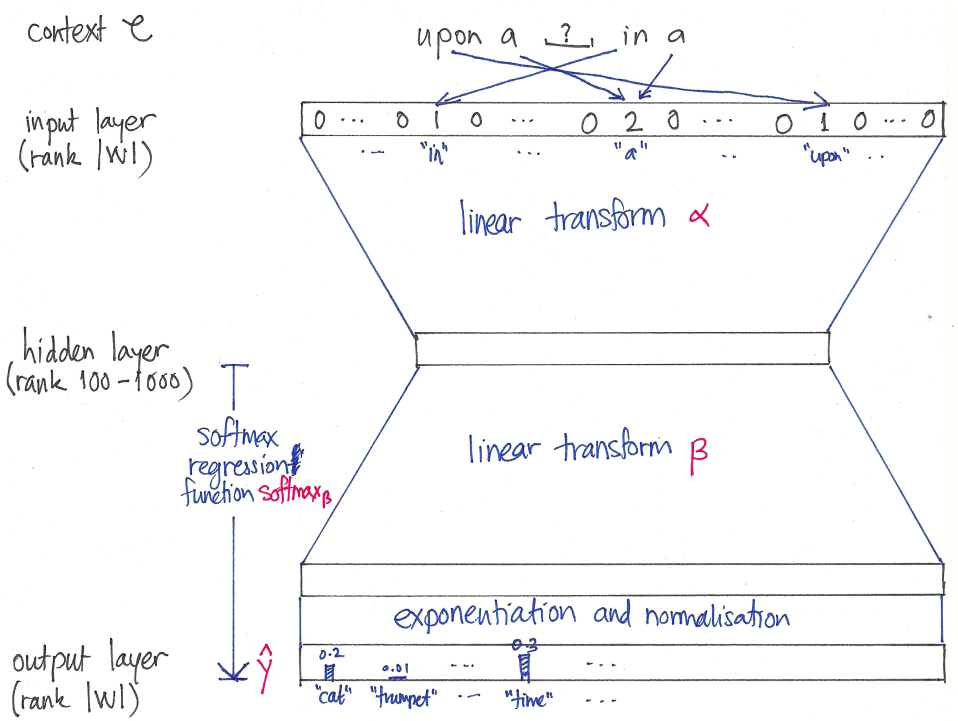


training CBOW with softmax regression

Model:

$$\hat{y} = P(\cdot|\mathcal{C}; \alpha, \beta) = \text{softmax}_\beta(\sum_{w \in \mathcal{C}} \alpha_w),$$

where α, β are families of parameter vectors. Pictorially:



stochastic gradient descent

- learn the model parameters (here, the linear transforms)
- minimize the difference between output distribution $\hat{y}$ and target distribution y, measured using the cross-entropy H:

$$H(y, \hat{y}) = -\sum_{w \in W} y_w \log \hat{y}_w$$

- given y is one-hot, same as maximizing the probability of the correct outcome

$$\hat{y}_{w_0} = P(w_0 | \mathcal{C}; \alpha, \beta).$$

- use stochastic gradient descent: for each (current word, context) pair, update all the parameters once.

word2vec word representation

Post-training, associate every word $w \in W$ with a vector `vec[w]`:

- `vec[w]` is the vector of synaptic strengthes connecting the input layer unit w to the hidden layer
- more meaningfully, `vec[w]` is the hidden-layer representation of the single-word context $\mathcal{C} = \{w\}$.
- vectors are (artifically) normed to unit length (Euclidean norm), post-training.

word vectors encode meaning

Consider words $w, w' \in W$:

$$
\begin{aligned}
w \approx w' \quad &\Leftrightarrow \quad P(\cdot|w) \approx P(\cdot|w') \\
&\qquad \text{(by the Distributional Hypothesis)} \\
&\Leftrightarrow \quad \text{softmax}_\beta(\texttt{vec[w]}) \approx \text{softmax}_\beta(\texttt{vec[w']}) \\
&\qquad \text{(if model is well-trained)} \\
&\Leftrightarrow \quad \texttt{vec[w]} \approx \texttt{vec[w']}
\end{aligned}
$$

The last equivalence is tricky to show ...

word vectors encode meaning (cont.)

We compare output distributions using the cross-entropy:

$$H(\text{softmax}_\beta(u), \text{softmax}_\beta(v))$$

- $\Leftarrow$ follows from continunity in u, v
- $\Rightarrow$ can be argued for from the convexity in v when u is fixed.

word relationship encoding

Given two examples of a single word relationship e.g.

queen is to *king* as *aunt* is to *uncle*

Find the closest point to

$$\texttt{vec[queen]} + (\texttt{vec[uncle]} - \texttt{vec[aunt]}).$$

It should be `vec[king]`.

- Perform this test for many word relationship examples.
- CBOW & Skip-gram give correct answer in 58% - 69% of cases.
- Cosine distance is used (justified empirically!).
- What is the natural metric?

Source: *Efficient estimation of word representations in vector space*, Mikolov et al., 2013

softmax implementations are slow

Updates *all* second-layer parameters for every (current word, context) pair $(w_0, \mathcal{C})$ – very costly.

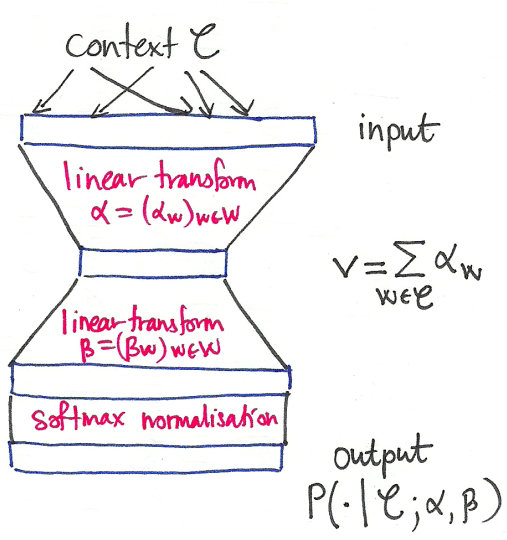
Softmax models

$$P(w_0|\mathcal{C};\alpha,\beta) = \frac{\exp^{\beta_w^\top v}}{\sum_{w' \in W} \exp^{\beta_{w'}^\top v}}$$

where $v = \sum_{w \in \mathcal{C}} \alpha_w$, and

$$(\alpha_w)_{w \in W} \quad (\beta_w)_{w \in W}$$

are the model parameters.



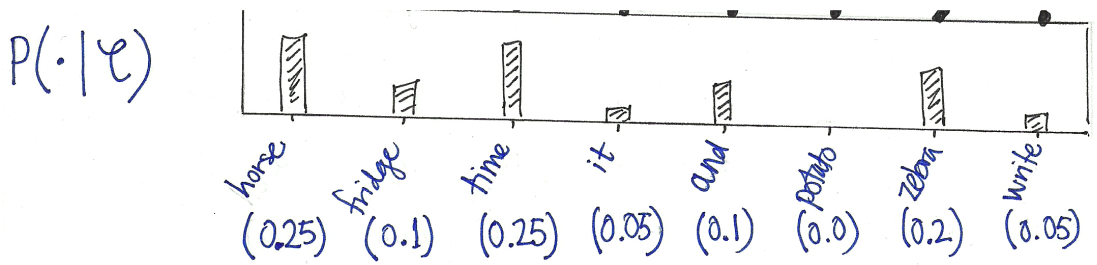
For each $(w_0, \mathcal{C})$ pair, must update $O(|W|) \approx 100k$ parameters.

alternative models with fewer parameter updates

word2vec offers two alternatives to replace softmax.

- “hierarchical softmax” (H.S.) (Morin & Bengio, 2005)
- “negative sampling”, an adaptation of “noise contrastive estimation” (Gutmann & Hyvärinen, 2012) (skipped today)
- negative sampling scales better in vocabulary size
- quality of word vectors comparable
- both make significantly fewer parameter updates in the second-layer (no less parameters).

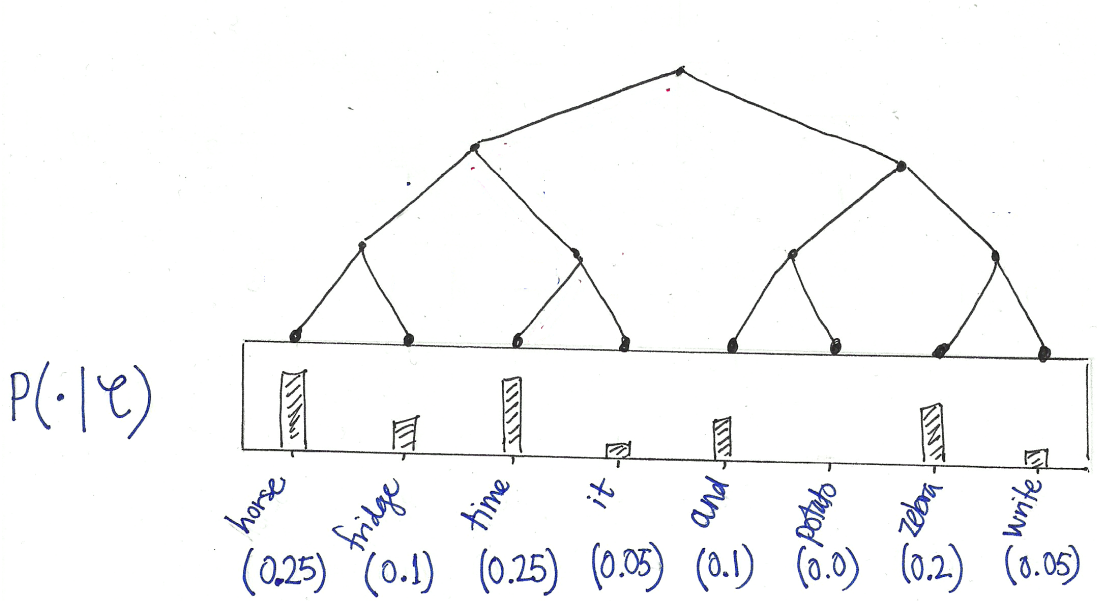
hierarchical softmax



- choose an arbitrary binary tree (# leaves = vocabulary size)
- then $P(\cdot|\mathcal{C})$ induces a weighting of the edges
- think of each parent node n as a Bernoulli distribution P_n on its children. Then e.g.

$$P(\texttt{time}|\mathcal{C}) = P_{n_0}(\text{left}|\mathcal{C})P_{n_1}(\text{right}|\mathcal{C})P_{n_2}(\text{left}|\mathcal{C})$$

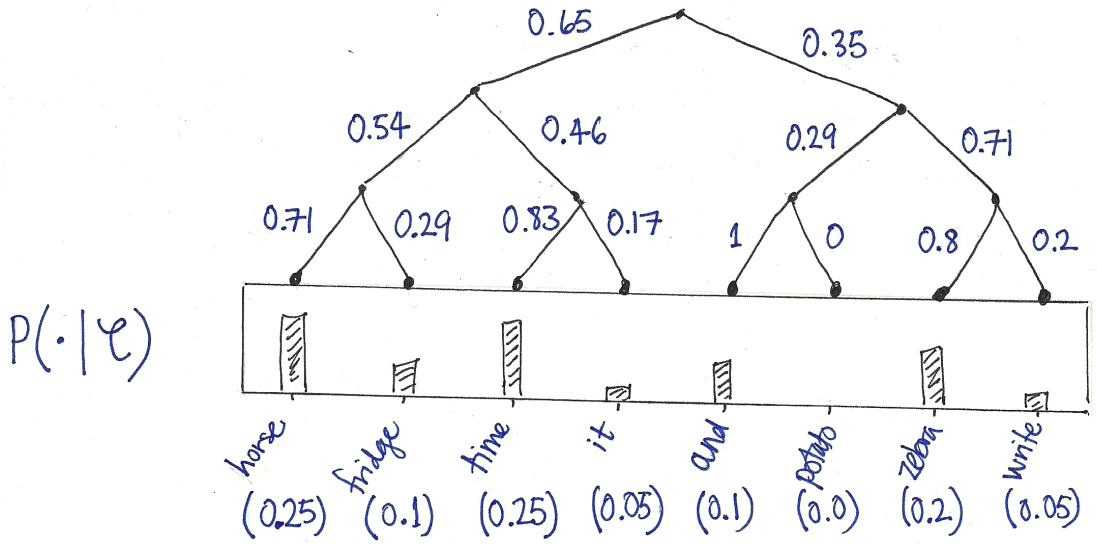
hierarchical softmax



- choose an arbitrary binary tree (# leaves = vocabulary size)
- then $P(\cdot|\mathcal{C})$ induces a weighting of the edges
- think of each parent node n as a Bernoulli distribution P_n on its children. Then e.g.

$$P(\texttt{time}|\mathcal{C}) = P_{n_0}(\text{left}|\mathcal{C})P_{n_1}(\text{right}|\mathcal{C})P_{n_2}(\text{left}|\mathcal{C})$$

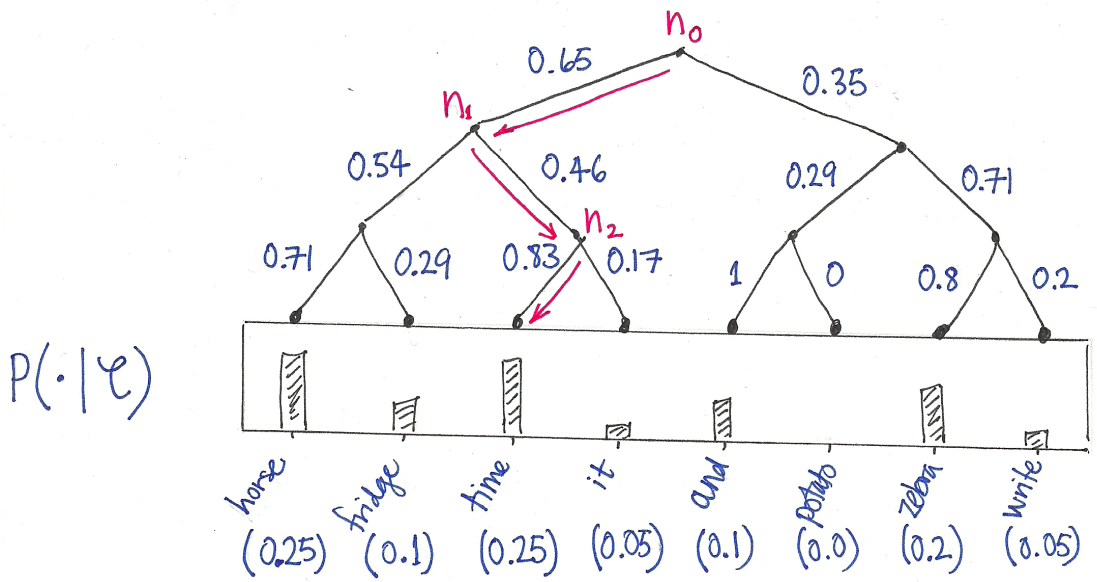
hierarchical softmax



- choose an arbitrary binary tree (# leaves = vocabulary size)
- then $P(\cdot|\mathcal{C})$ induces a weighting of the edges
- think of each parent node n as a Bernoulli distribution P_n on its children. Then e.g.

$$P(\texttt{time}|\mathcal{C}) = P_{n_0}(\text{left}|\mathcal{C})P_{n_1}(\text{right}|\mathcal{C})P_{n_2}(\text{left}|\mathcal{C})$$

hierarchical softmax



- choose an arbitrary binary tree (# leaves = vocabulary size)
- then $P(\cdot|\mathcal{C})$ induces a weighting of the edges
- think of each parent node n as a Bernoulli distribution P_n on its children. Then e.g.

$$P(\texttt{time}|\mathcal{C}) = P_{n_0}(\text{left}|\mathcal{C})P_{n_1}(\text{right}|\mathcal{C})P_{n_2}(\text{left}|\mathcal{C})$$

hierarchical softmax modelling

In H.S., the probability of any outcome depends on only $O(\log|W|)$ parameters.

For each parent node n, assume that its Bernoulli distribution is modelled by: $P_n(\mathit{left}|\mathcal{C}) = \sigma(\theta_n^\top v)$ where:

- θ_n is a vector of parameters
- $v = \sum_{w \in \mathcal{C}} \alpha w$ is the context vector
- σ is the sigmoid function $\sigma(z) = \frac{1}{1+\exp^{-z}}$

Then, e.g.

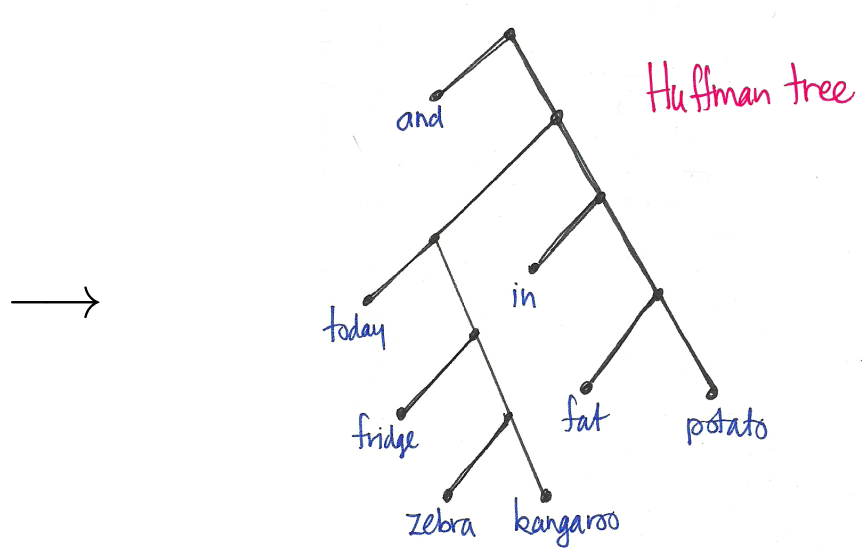
$$P(\texttt{time}|\mathcal{C}) = \sigma(\theta_{n_0}^\top v) \cdot (1 - \sigma(\theta_{n_1}^\top v)) \cdot \sigma(\theta_{n_2}^\top v)$$

depends on only 3 = $\log_2|W|$ of the parameter vectors θ_n.

word2vec H.S. uses Huffman tree

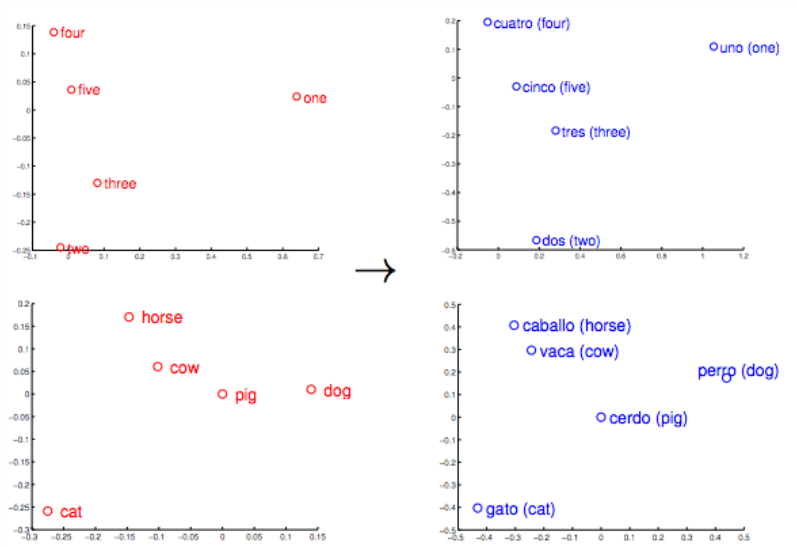
- could use any binary tree (# leaves = vocabulary size)
- word2vec uses a Huffman tree
  - frequent words have shorter paths in the tree
  - results in an even faster implementation

word	count
fat	3
fridge	2
zebra	1
potato	3
and	14
in	7
today	4
kangaroo	2



application to machine translation

- train word representations for e.g. English and Spanish separately
- the word vectors are similarly arranged!
- learn a linear transform that (approximately) maps the word vectors of English to the word vectors of their translations in Spanish
- same transform for all vectors



Source: *Exploiting Similarities among Languages for Machine Translation*, Mikolov, Quoc, Sutskever, 2013

applications to machine translation - results

English - Spanish: can guess the correct translation in 33% - 35% percent of the cases.

Translation	Edit Distance		Word Co-occurrence		Translation Matrix	
	P@1	P@5	P@1	P@5	P@1	P@5
En → Sp	13%	24%	19%	30%	33%	51%
Sp → En	18%	27%	20%	30%	35%	52%
En → Cz	5%	9%	9%	17%	27%	47%
Cz → En	7%	11%	11%	20%	23%	42%

Source: *Exploiting Similarities among Languages for Machine Translation*, Mikolov, Quoc, Sutskever, 2013