Word Embeddings

Word Vectors

- At one level, it is simply a vector of weights.
- In a simple 1-of-N (or 'one-hot') encoding every element in the vector is associated with a word in the vocabulary.
- The encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero.
- One-hot representation:

Motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND

Hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Word Vectors - One-hot Encoding

- Suppose our vocabulary has only five words: King, Queen, Man, Woman, and Child.

- We could **encode the word 'Queen'** as:

King	Queen	Woman	Man	Child
0	1	0	0	0

1-of-N Encoding

Limitations of One-hot encoding

Word vectors are not comparable

- Using such an encoding, there is no meaningful comparison we can make between word vectors other than equality testing.

Word2Vec – A distributed representation

Distributional representation – word embedding?

Any word w_i in the corpus is given a distributional representation by an embedding

$$w_i \in R^d$$

i.e., a d ← dimensional vector, which is mostly learnt!

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Distributional Representation

- Take a vector with several hundred dimensions (say 1000).
- Each word is represented by a distribution of weights across those elements.
- So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and
- Each element in the vector contributes to the definition of many words.

Distributional Representation: Illustration

- If we label the dimensions in a hypothetical word vector (there are no such pre-assigned labels in the algorithm of course), it might look a bit like this:

		King	Green	Queen	Princess
Royalty ---		0.99	0.02	0.99	0.98
Masculine ---		0.99	0.05	0.01	0.02
Feminine ---		0.05	0.88	0.99	0.94
Age ---		0.7	0.6	0.5	0.1
...	.	.	.	.	.
	.	.	.	.	.

Such a vector comes to represent in some abstract way the ‘meaning’ of a word

Word Embeddings

- Embeddings are low dimensional representations of points in a higher dimensional vector space.
- Word embeddings are dense vector representations of words in a lower dimensional space, i.e., the number of dimensions is considerably less than the number of words.
- Word embeddings are capable of capturing both the context in which a word is likely to appear and some aspects of its meaning.
- Dimension d - typically in the range 50 to 1000
- Similar words should have similar embeddings
- SVD can also be thought of as an embedding method

- Work done by **Mikolov** and his colleagues at Google, who created a family of algorithms known as word2vec (Mikolov et al. 2013a,b) that construct embeddings via backpropagation learning.

Reasoning with Word Vectors

- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.
- Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.

Case of Singular-Plural Relations:

- If we denote the vector for word i as x_i, and focus on the singular/plural relation, we observe that:

x_{apple} - x_{apples} ≈ x_{car} – x_{cars} ≈ x_{family} - $x_{families}$ ≈ x_{car} – x_{cars} and so on.

Reasoning with Word Vectors

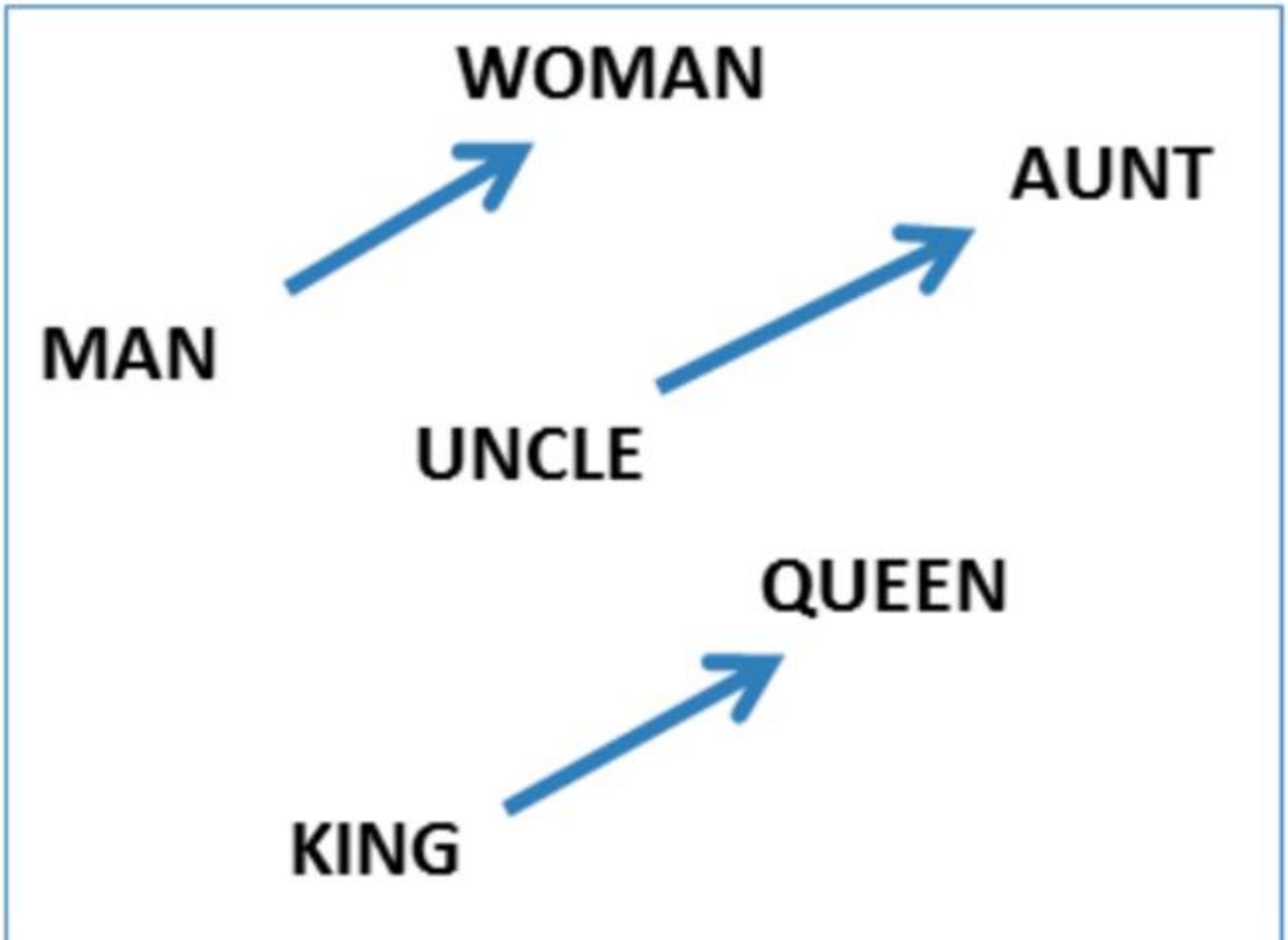
Perhaps more surprisingly, we find that this is also the case for a variety of semantic relations.

Good at answering analogy questions

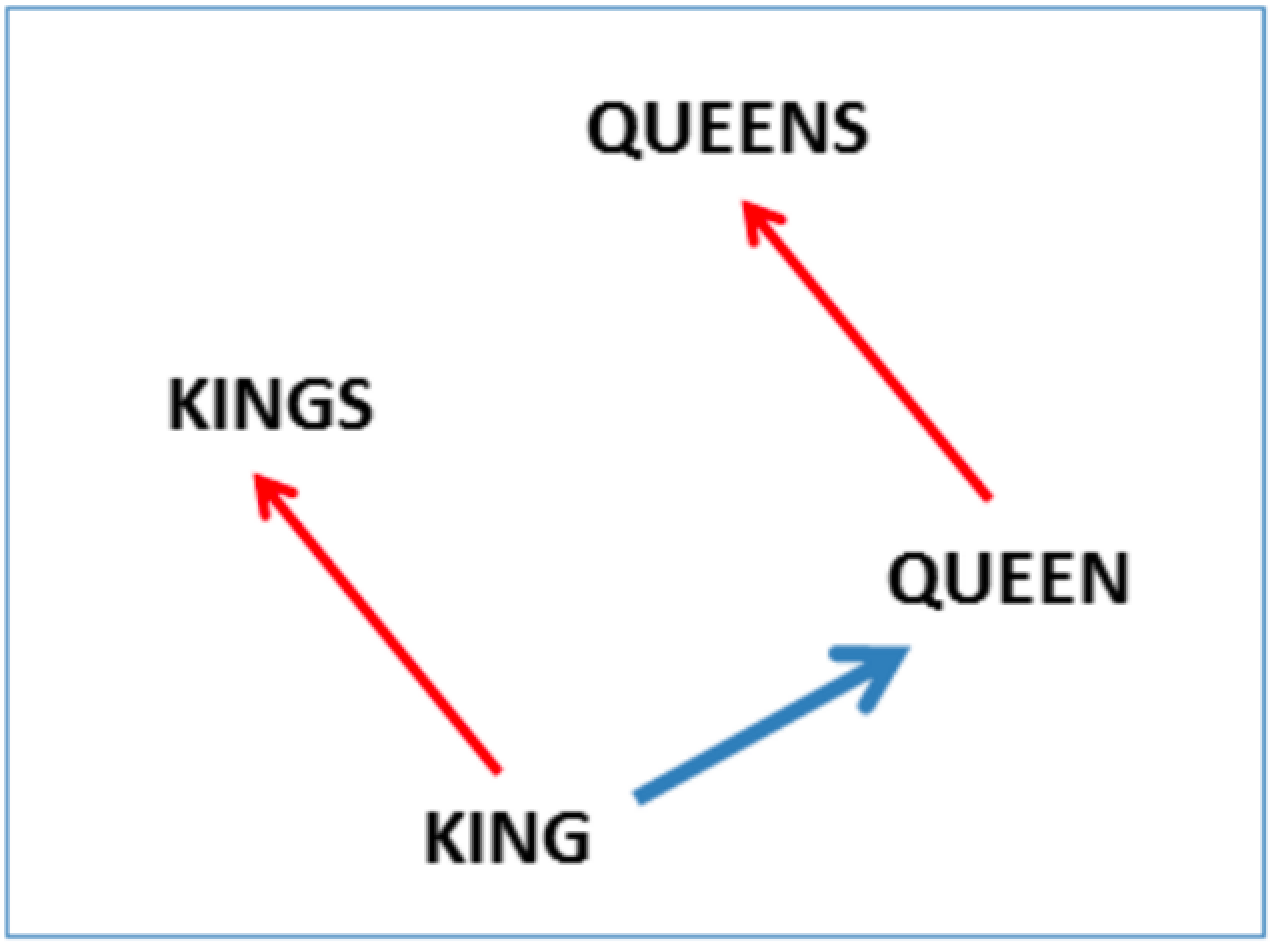
- a is to b, as c is to ?
- man is to woman as uncle is to ? (aunt)

A simple vector offset method based on cosine distance shows the relation.

Vector Offset for Gender Relation



Vector Offset for Singular-Plural Relation



Encoding Other Dimensions of Similarity

Analogy Testing:

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

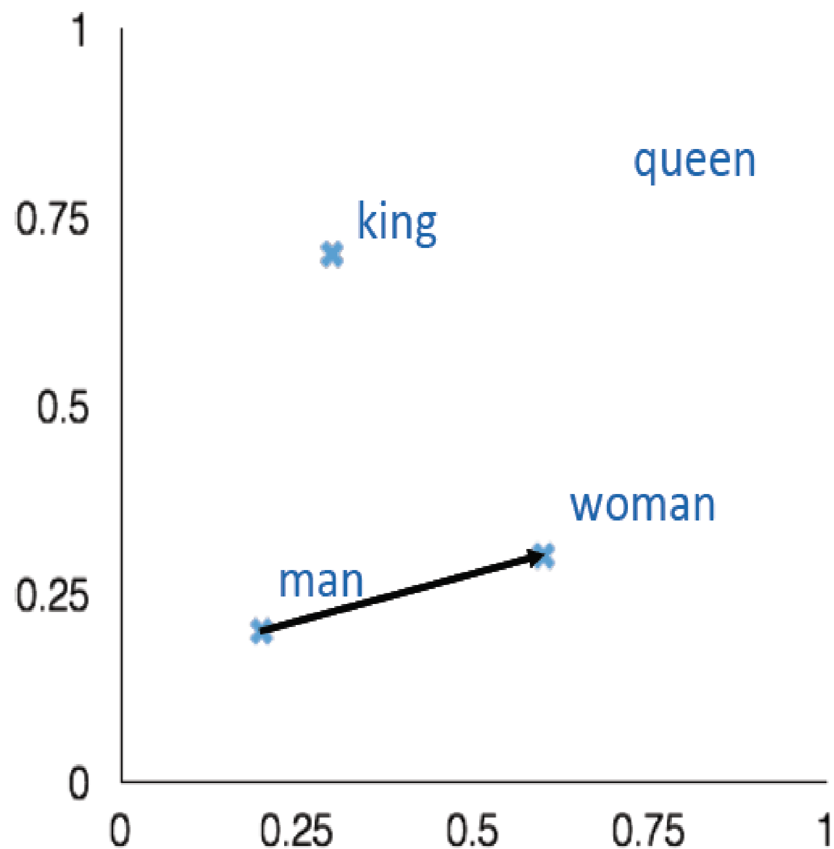
Analogy Testing

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Country-capital city relationships

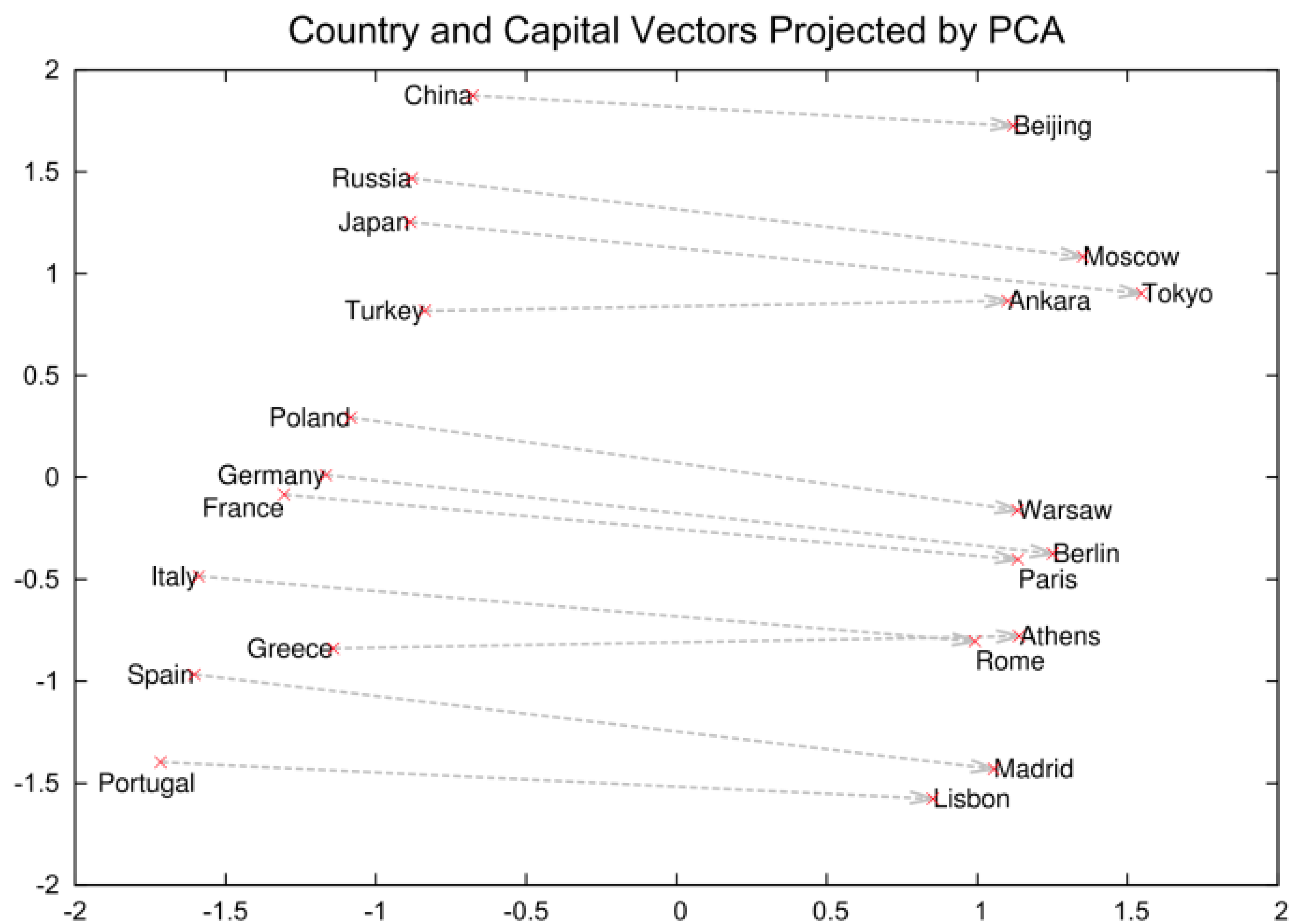


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

More Analogy Questions

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

Element Wise Addition

We can also use element-wise addition of vector elements to ask questions such as 'German + airlines' and by looking at the closest tokens to the composite vector come up with impressive answers:

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

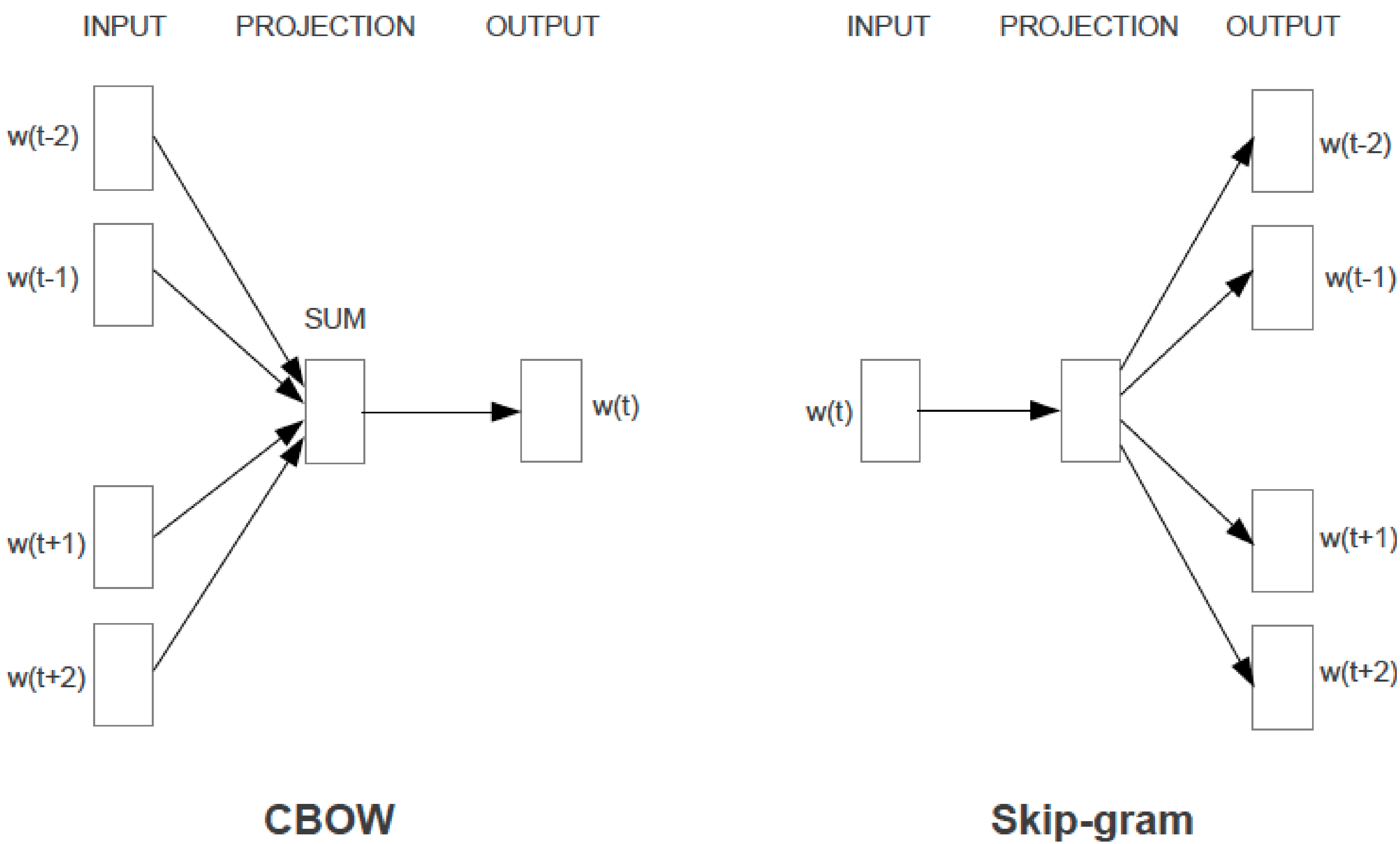
Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

Learning Word Vectors

Basic Idea:

- Instead of capturing co-occurrence counts directly, predict (using) surrounding words of every word.
- Code as well as word-vectors: https://code.google.com/p/word2vec/

Two Variations: CBOW and Skip-grams



CBOW

CBOW

- Consider a piece of prose such as:

  "The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntactic and semantic word relationships."

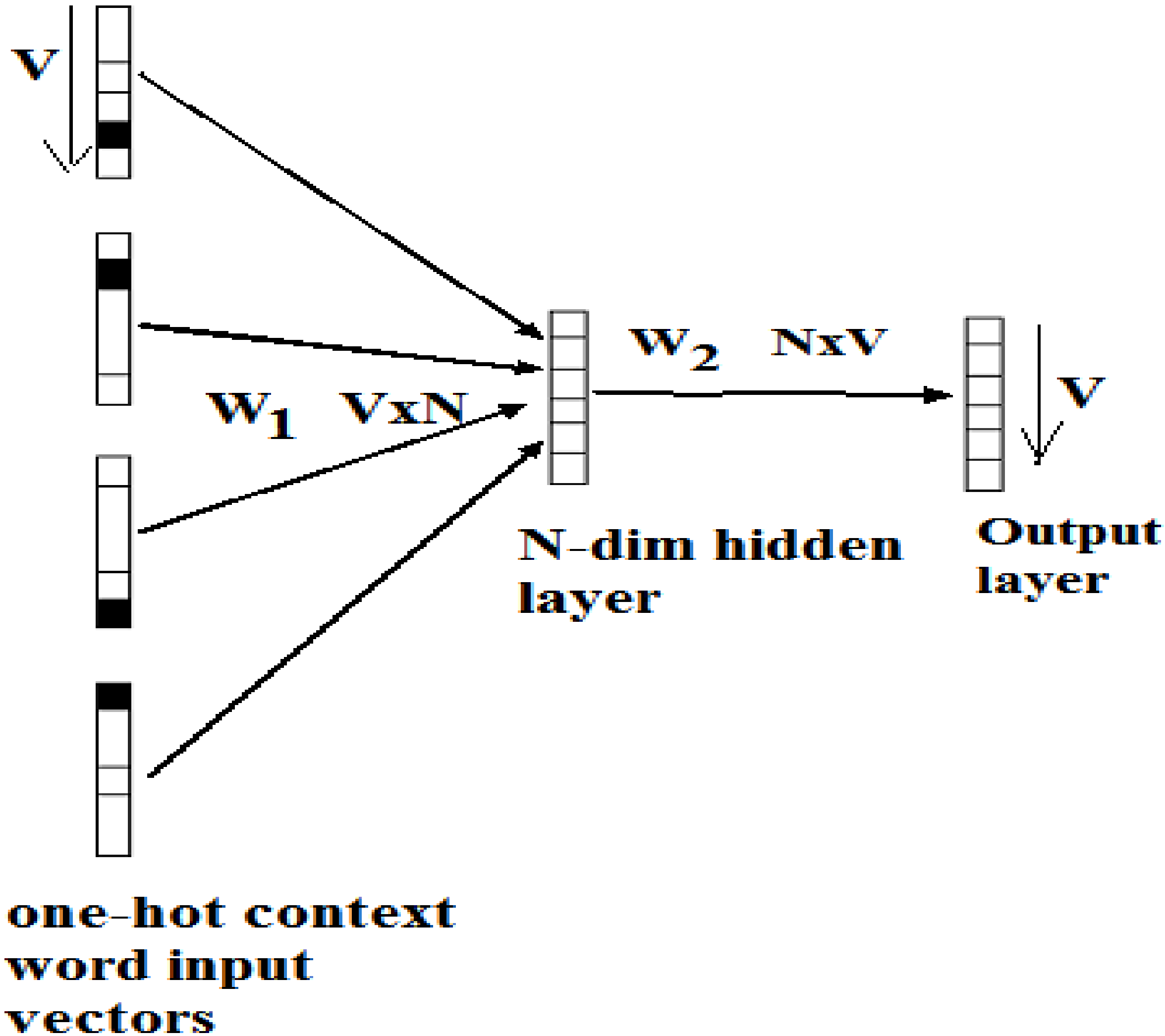
- Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words that precede it, and the four words that follow it:

…an efficient method for **learning** high quality distributed vector…

context **Focused word** **context**

CBOW

The context words form the input layer. Each word is encoded in one-hot form. A single hidden and output layer.



CBOW: Training Objective

- The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights.
- In our example, given the input (“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”), we want to maximize the probability of getting “learning” as the output.

CBOW: Input to Hidden Layer

- Since our input vectors are one-hot, multiplying an input vector by the weight matrix W1 amounts to simply selecting a row from W1.

$$
\begin{array}{ccc}
\text{Input} & W_1 & \text{hidden} \\
1\text{xV} & \text{V x N} & 1\text{ x N}
\end{array}
$$

$$
\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} = \begin{bmatrix} e & f & g & h \end{bmatrix}
$$

- Given C input word vectors, the activation function for the hidden layer h amounts to simply summing the corresponding ‘hot’ rows in W_1, and dividing by C to take their average.

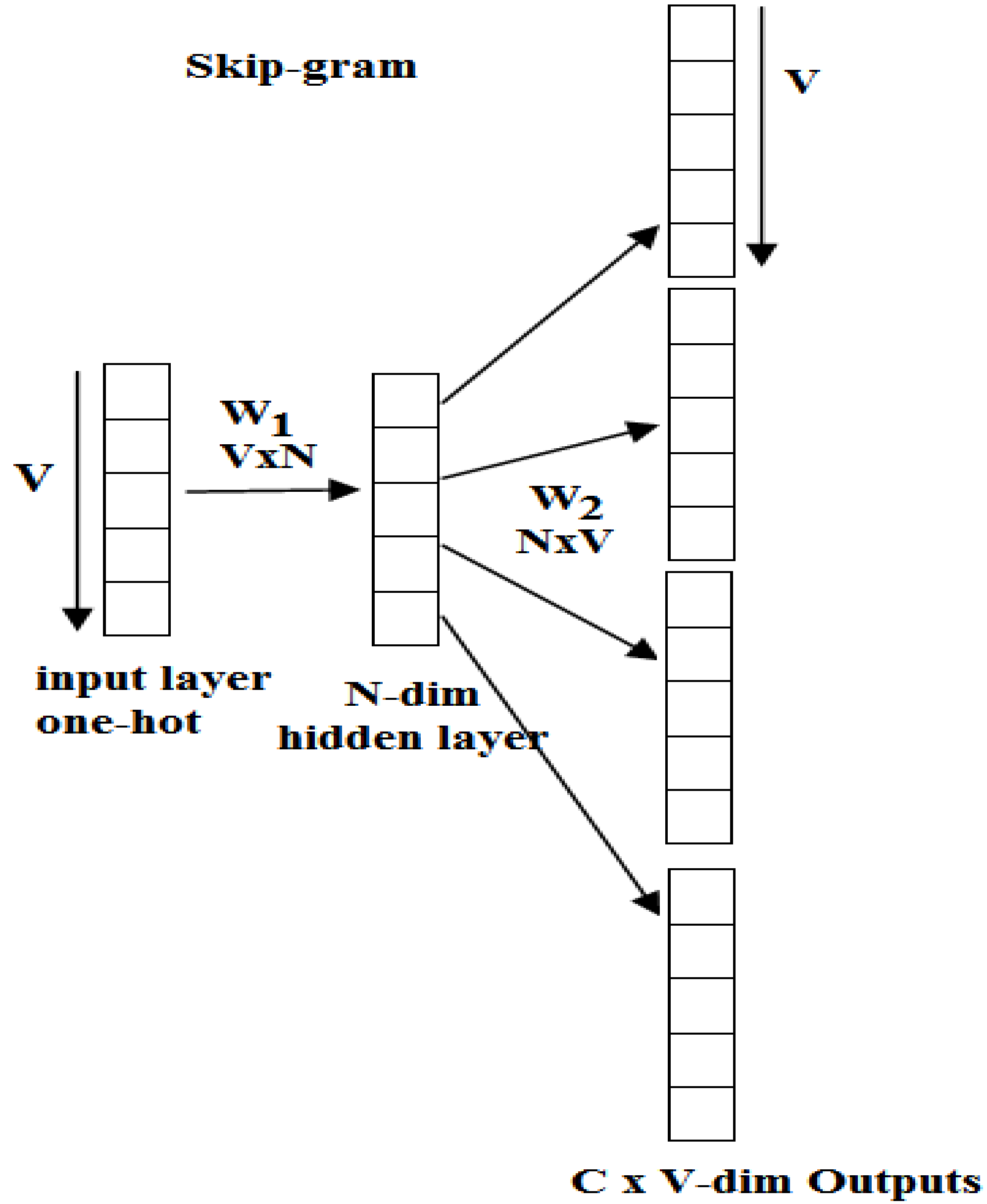
CBOW: Hidden to Output Layer

- From the hidden layer to the output layer, the second weight matrix W_2 can be used to compute a score for each word in the vocabulary, and softmax can be used to obtain the posterior distribution of words.

Skip-gram Model

Skip-gram Model

- The skip-gram model is the opposite of the CBOW model. It is constructed with the focus word as the single input vector, and the target context words are now at the output layer:



Skip-gram Model: Training

- The activation function for the hidden layer simply amounts to copying the corresponding row from the weights matrix W1 (linear) as we saw before.
- At the output layer, we now output-C multinomial distributions instead of just one.
- The training objective is to minimize the summed prediction error across all context words in the output layer. In our example, the input would be “learning”, and we hope to see (“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”) at the output layer.

Skip-gram Model

Details:

- Predict surrounding words in a window of length c of each word

Objective Function:

- Maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} log\, p(w_{t+j}|w_t)$$

Word Vectors

- For $p(w_{t+j}|w_t)$ the simplest first formulation is:

$$p(w_O|w_I) = \frac{exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W} exp({v'_w}^T v_{w_I})}$$

where v and v' are "input" and "output" vector representations of w (so every word has two vectors)

Parameters ϴ

- With d --dimensional words and V many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ v'_{aardvark} \\ v'_a \\ \vdots \\ v'_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Gradient Descent for Parameter Updates

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

Two sets of vectors

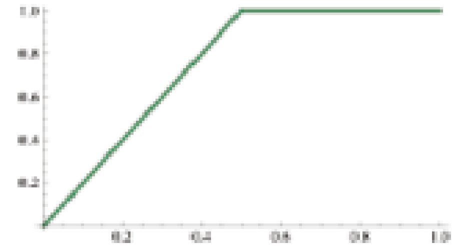
- Best solution is to sum these up:

  $L_{final} = L + L'$

- A good tutorial to understand parameter learning:
  - https://arxiv.org/pdf/1411.2738.pdf
- An interactive Demo
  - https://ronxin.github.io/wevi/

Glove

$$J = \frac{1}{2}\sum_{ij} f(P_{ij})\left(w_i \cdot \tilde{w}_j - \log P_{ij}\right)^2$$

$f \sim$ 

- Combine the best of both worlds – count based methods as well as direct prediction methods
  - Fast training
  - Scalable to huge corpora
  - Good performance even with small corpus, and small vectors
- Code and vectors: http://nlp.stanford.edu/projects/glove/