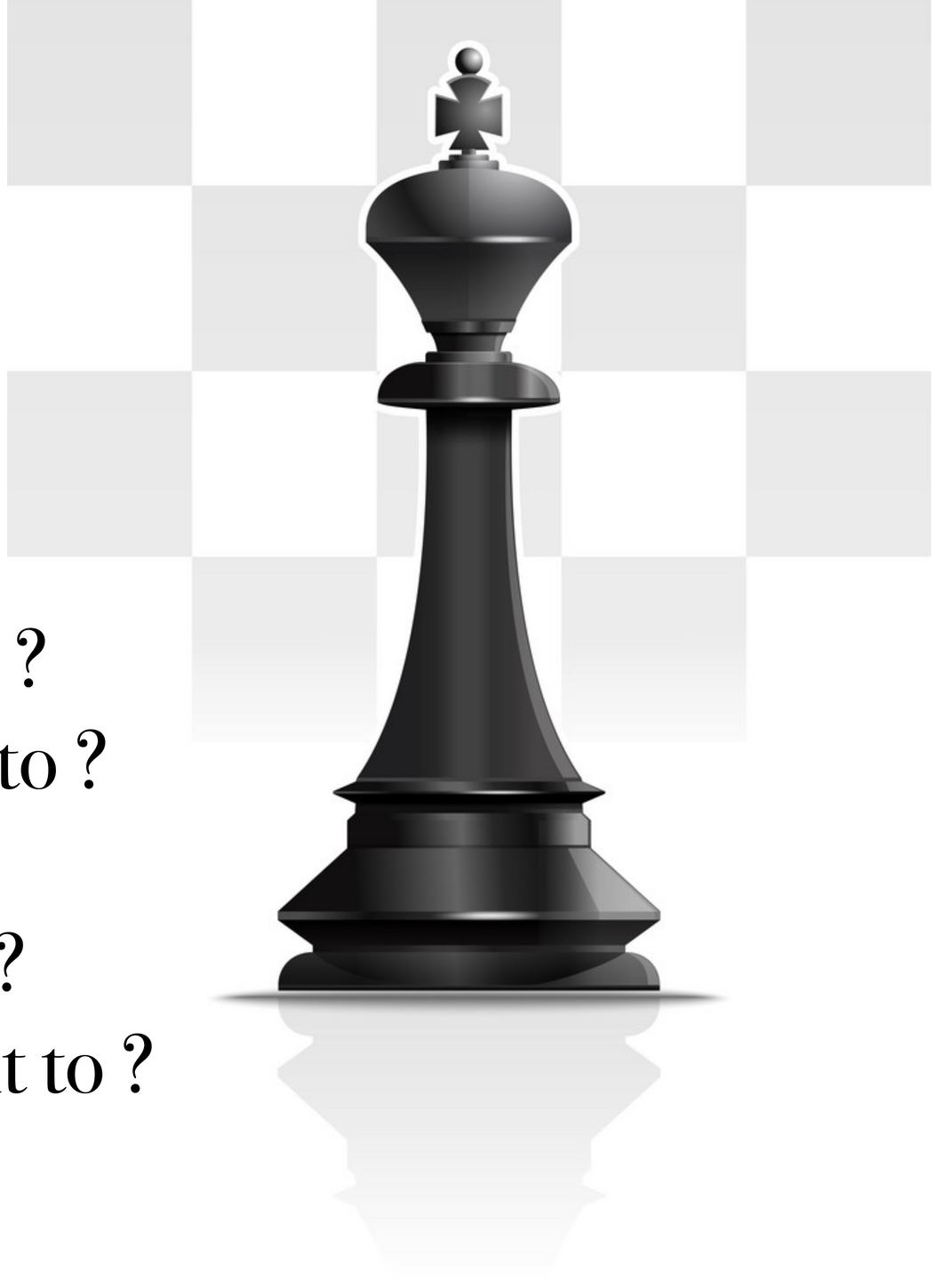
# Exploring Neural Word Embeddings with Python

Wolf Paulus

https://wolfpaulus.com

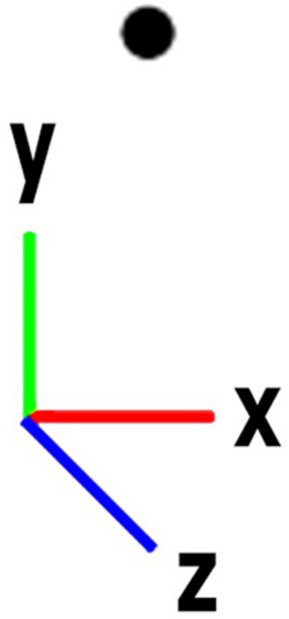
# Demo



man to king is like woman to ?
beer to Germany is like vine to ?
car to street is like bike to ?
hammer to tool is like car to ?
pupil to school is like student to ?

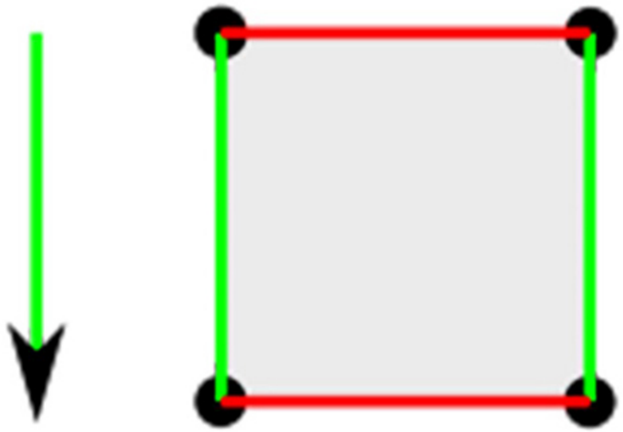

wolf@paulus.com

# 0D
# Point

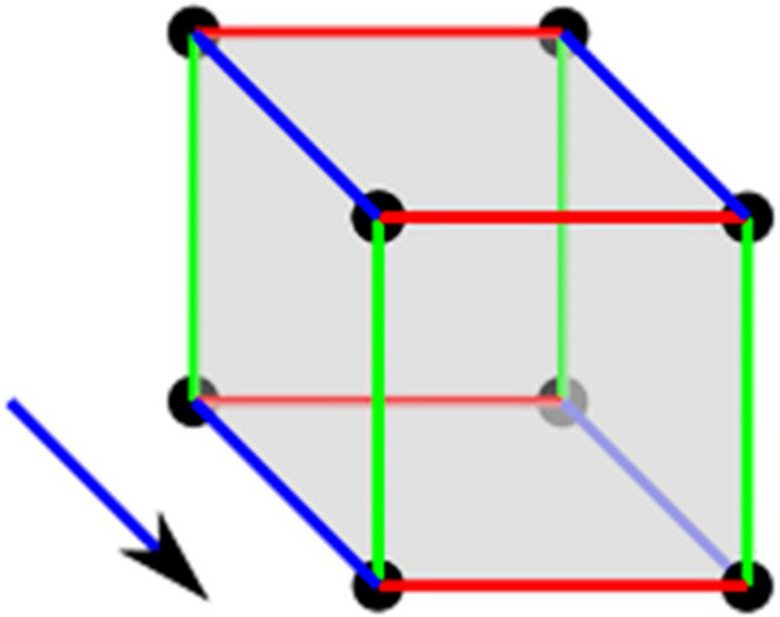



# 1D
# Line

# 2D
# Square



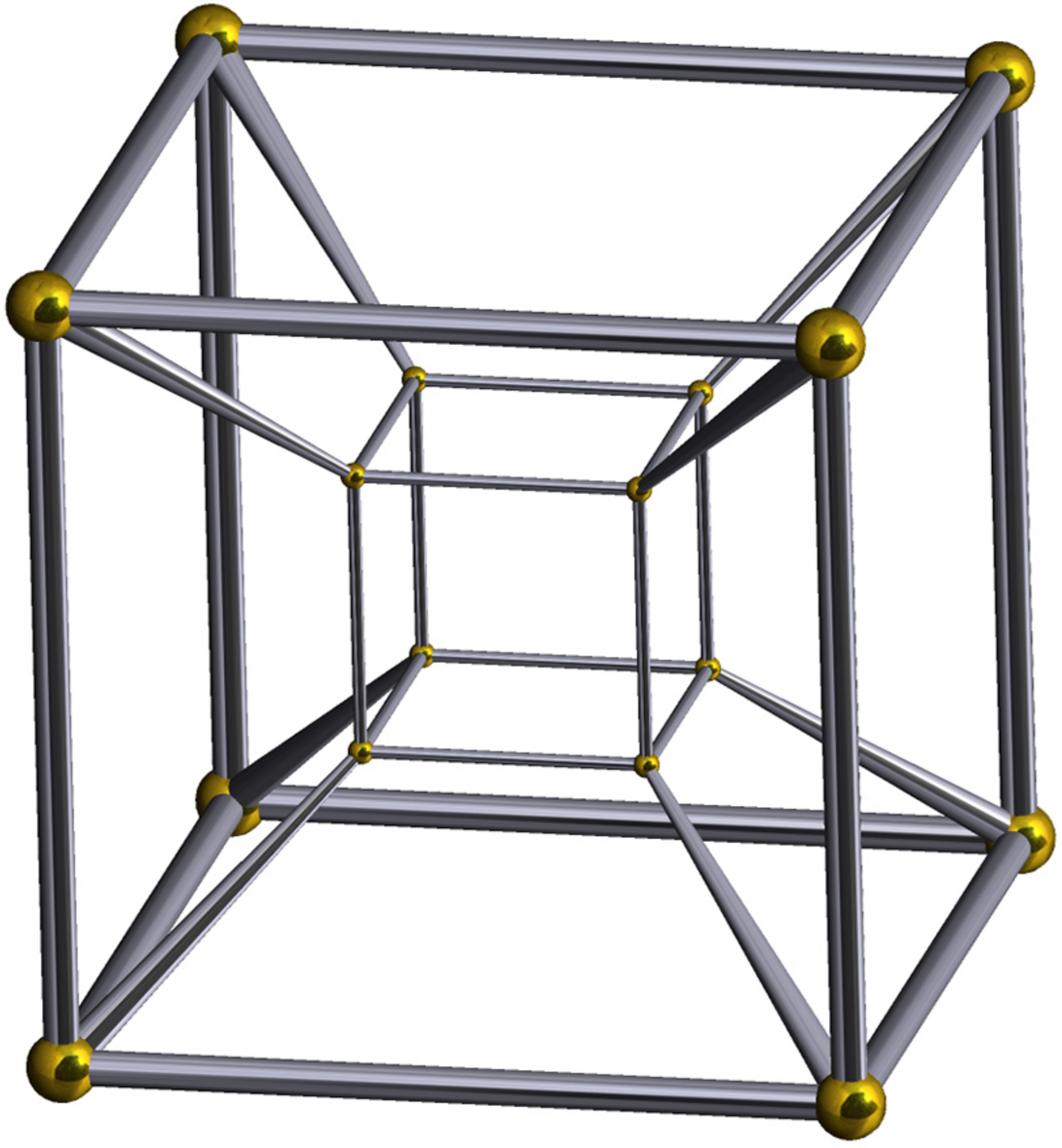
# 3D
# Cube

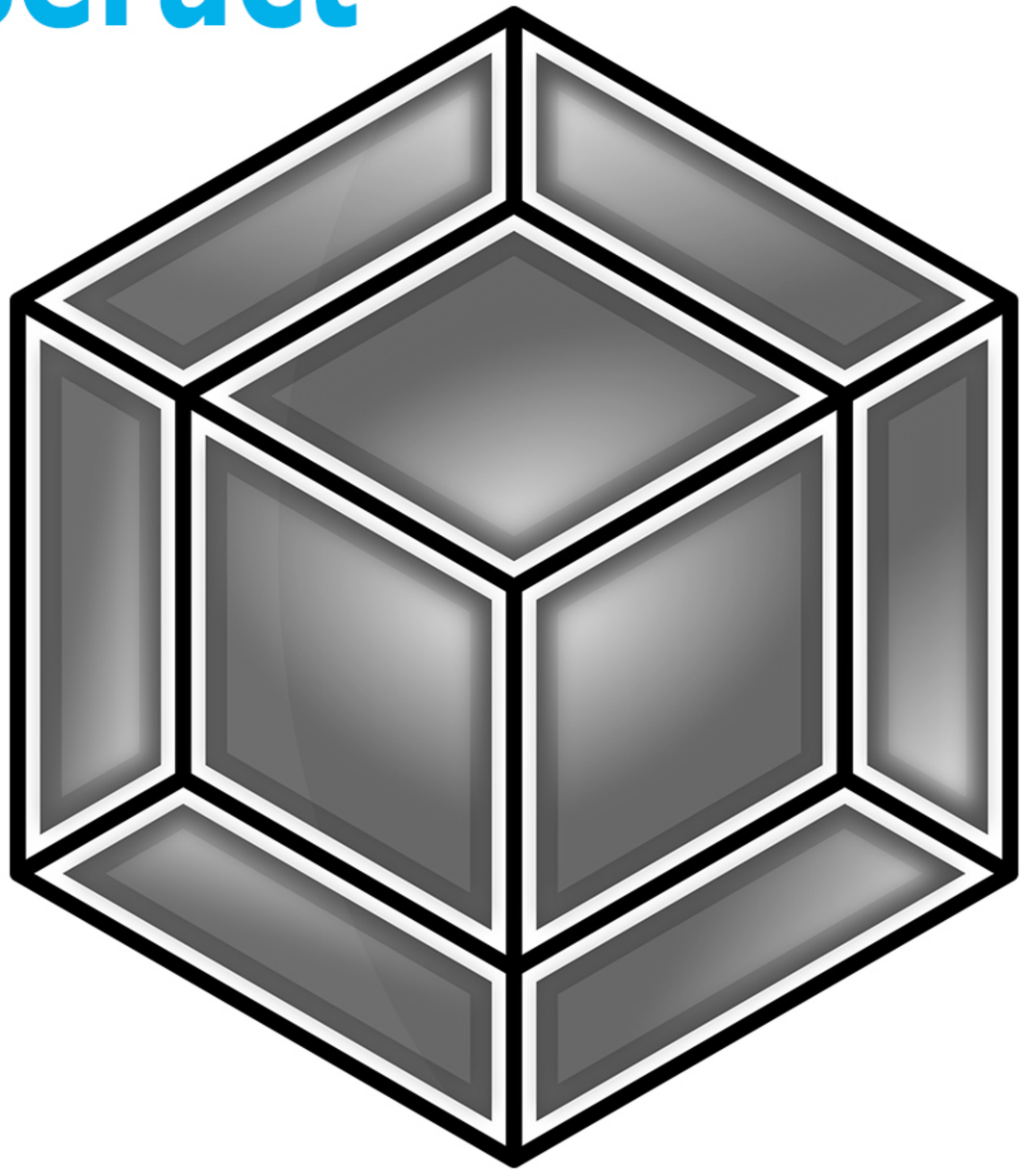


Wolf Paulus | https://wolfpaulus.com

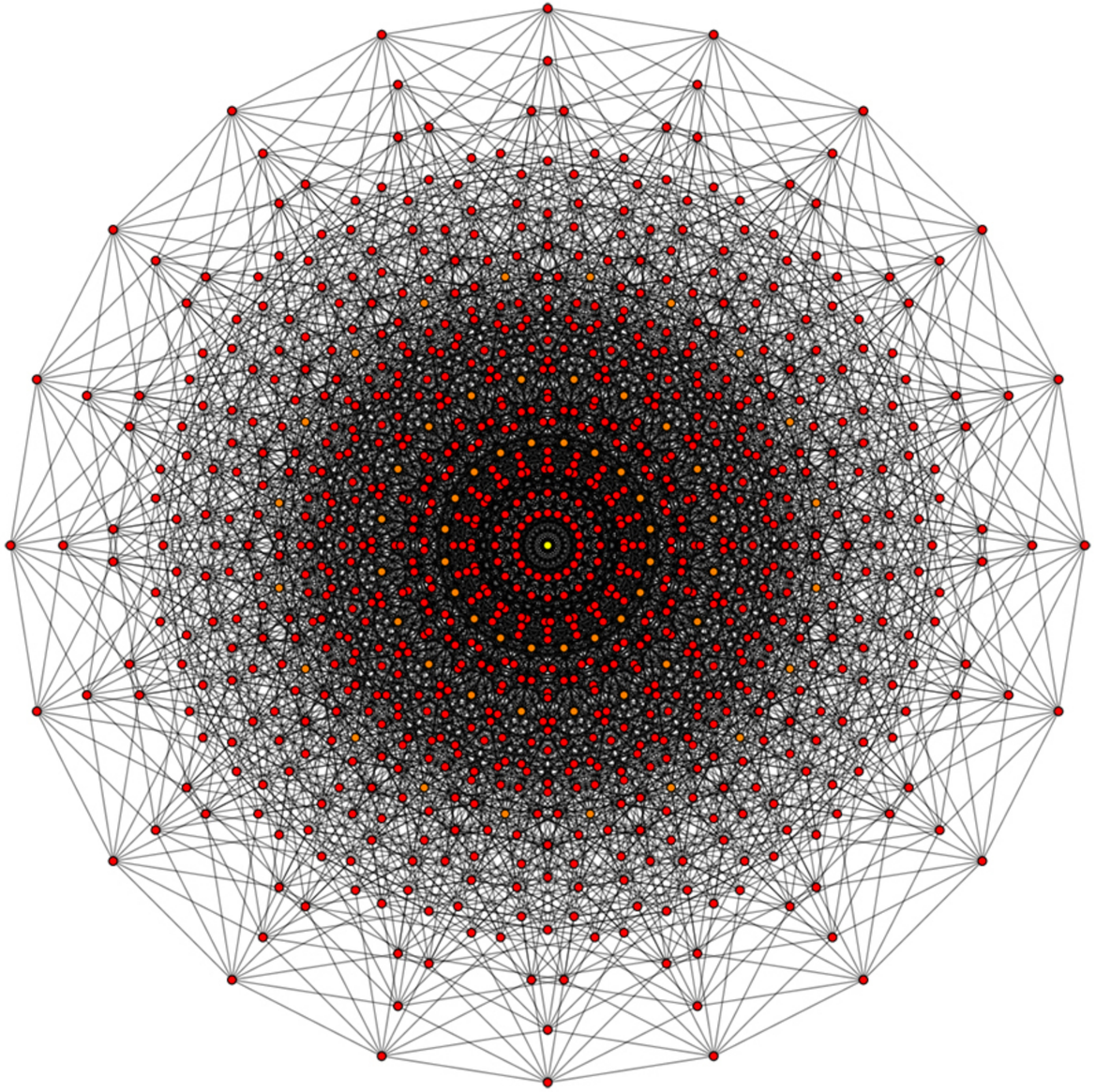
wolf@paulus.com

# 4D Tesseract





# 10D Dekeract



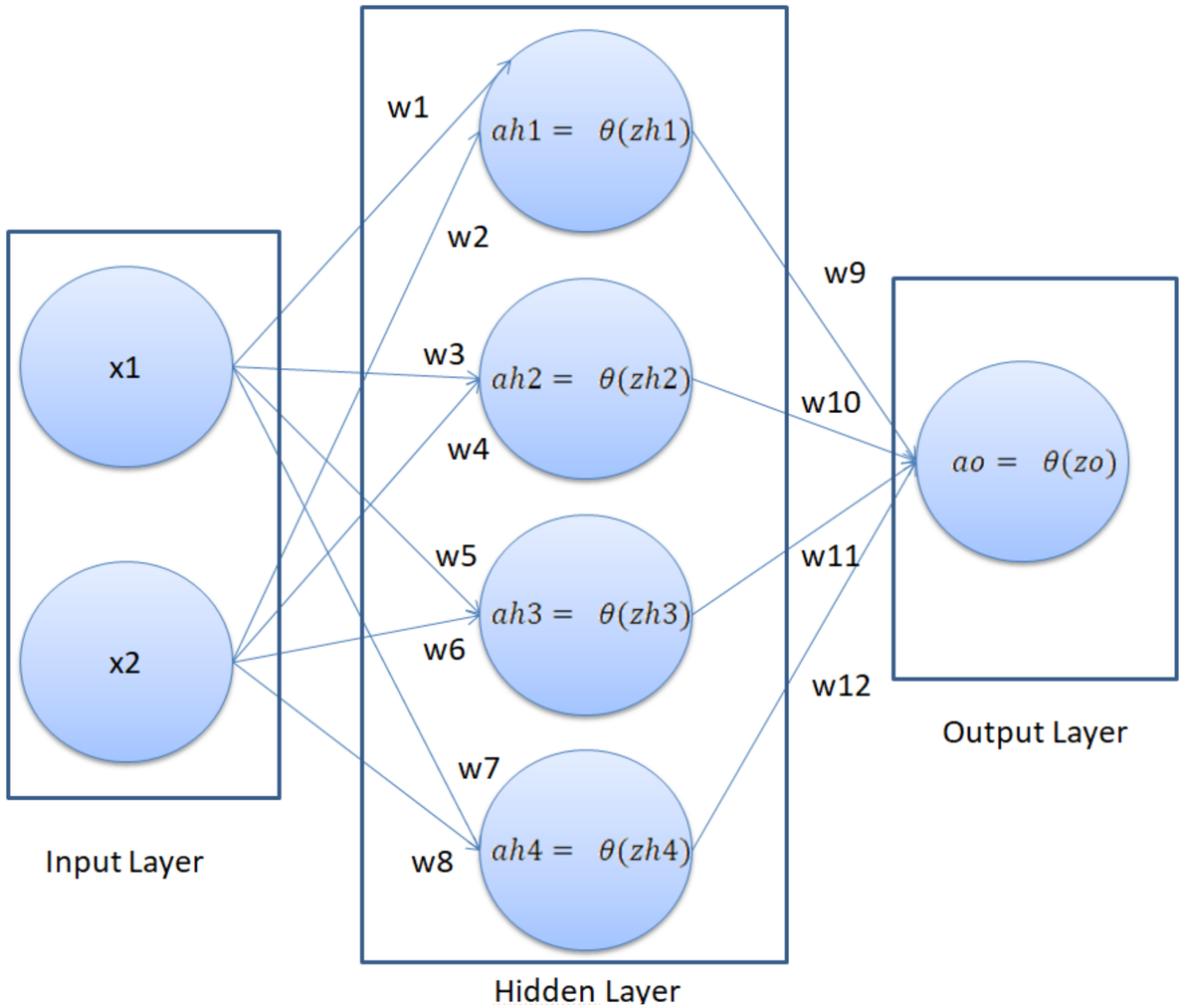
# Word Embeddings

A Word Embedding format generally tries to map a word using a dictionary to a vector.

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| man | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| woman | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| boy | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| girl | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| prince | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| princess | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| queen | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| king | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| monarch | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

A simple neural network with a single hidden layer is *trained* to perform a ***certain task***.
However, the goal is not to use the neural network once it's trained,
but to learn the weights of the hidden layer, i.e. “word vectors”.





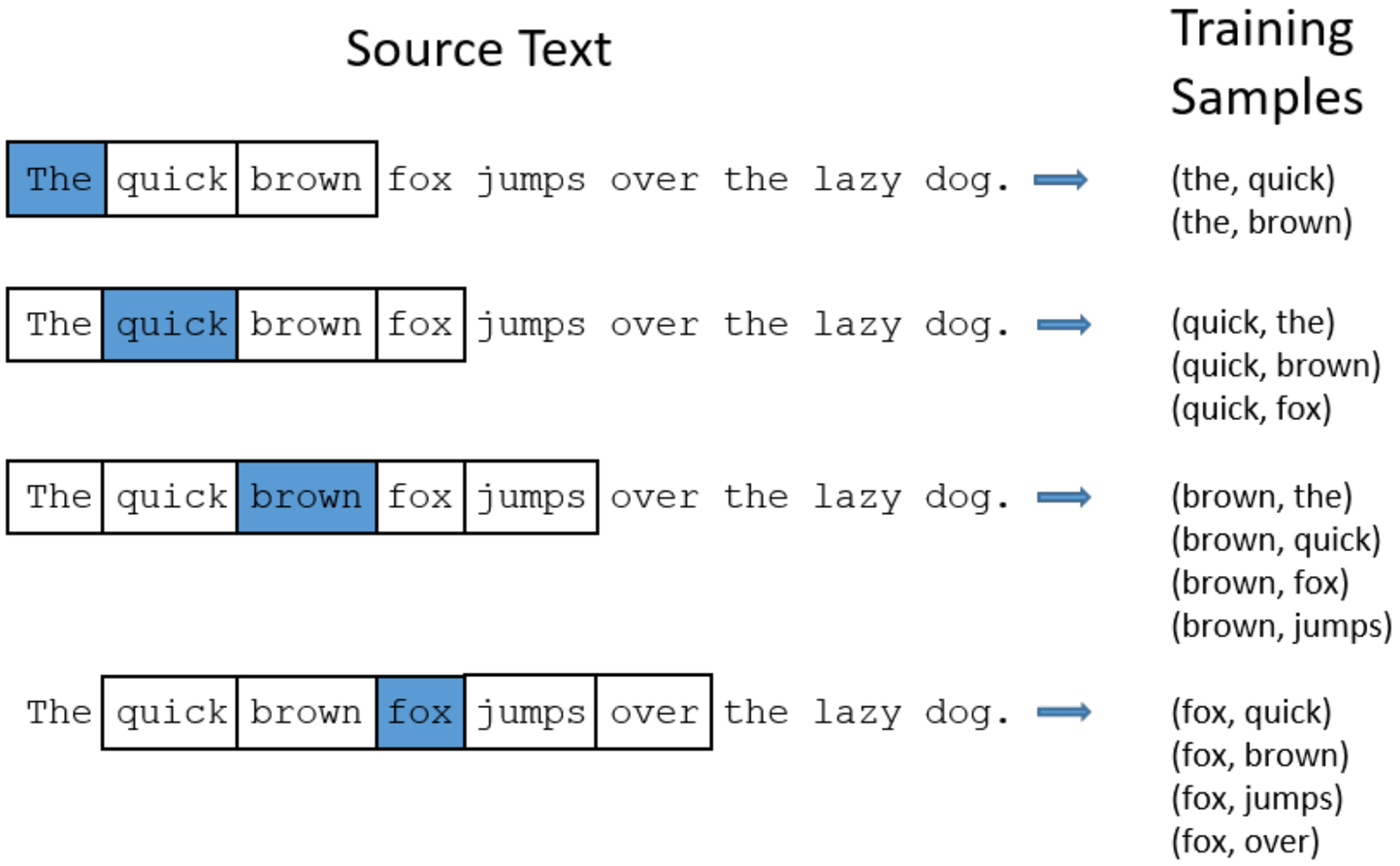
wolf@paulus.com

***Task***: Given a specific word in the middle of a sentence (the input word), look at the words nearby - a typical ***window-size*** might be 5, meaning 5 words behind and 5 words ahead (10 in total). The network is going to tell the probability for every word in the vocabulary of being a “nearby word”. The output probabilities are going to relate to how likely it is find each vocabulary word nearby the input word.

For example, if you gave the trained network the input word “Soviet”, the output probabilities are going to be much higher for words like “Union” and “Russia” than for unrelated words like “watermelon” and “kangaroo”.

The neural network is trained by feeding it word pairs found in the training documents (corpus).

The example shows some of the training samples (word pairs), taken form a training sentence, using a small window size of 2, with the word highlighted word being the input word.

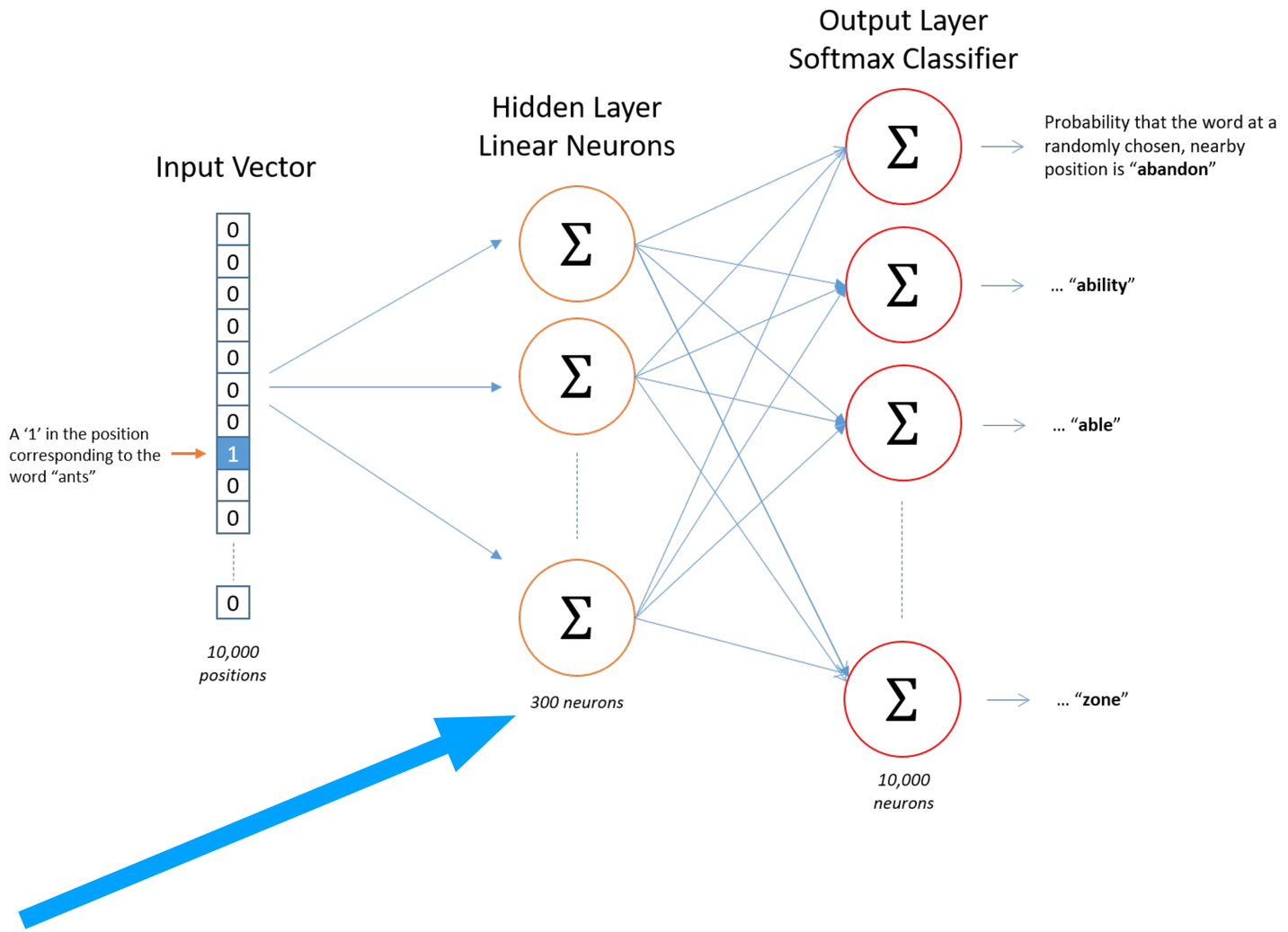


## Details

Strings can not be directly fed into a neural network.

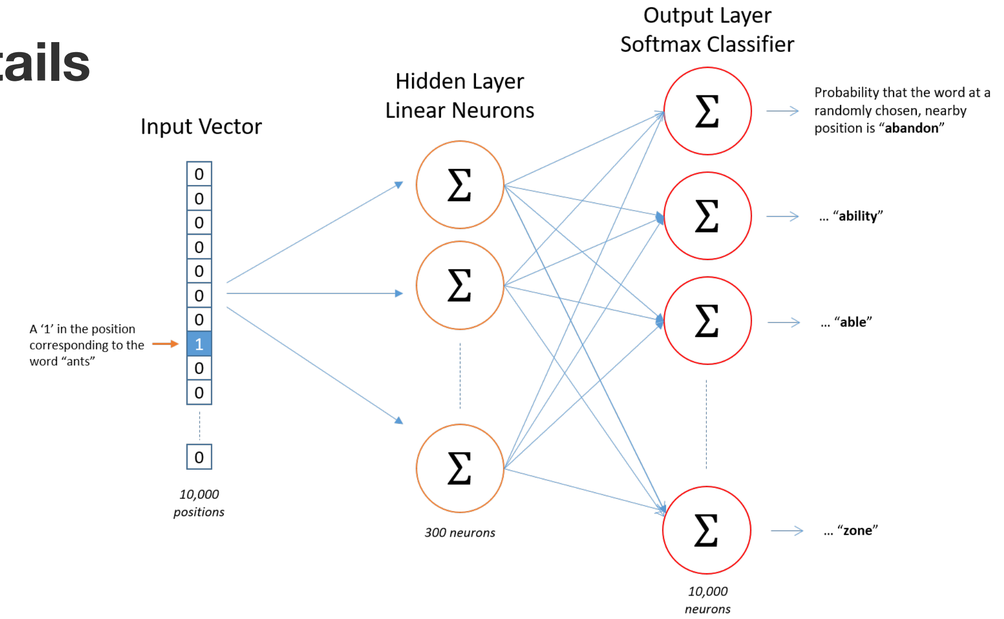
Therefore, in a  vocabulary of 10,000 unique words, each word would be represented by a vector will have 10,000 components.

The output of the network is a single vector (also with 10,000 components) containing, for every word in our vocabulary, the probability that a randomly selected nearby word is that vocabulary word.



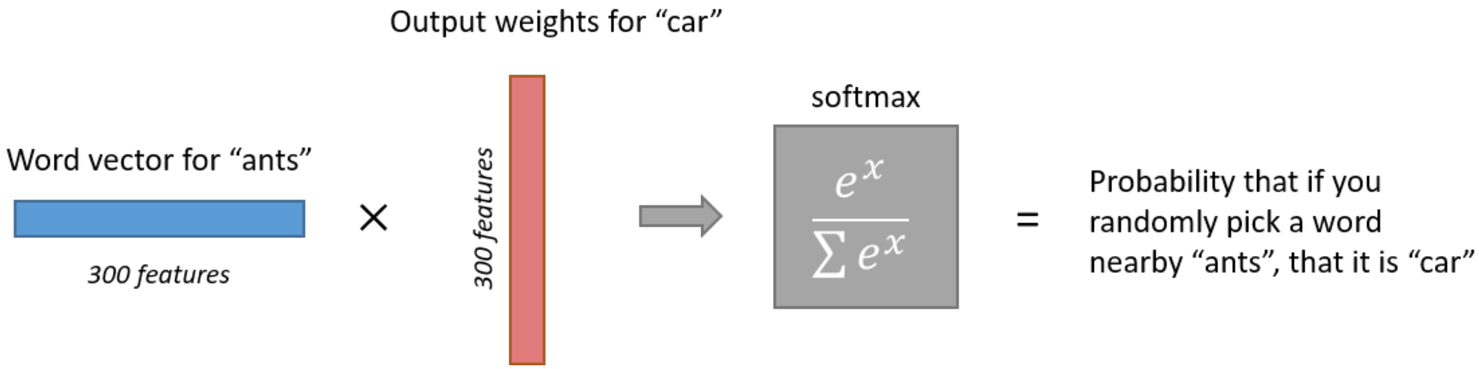

**300 features is what Google used in their published model trained on the Google news dataset**

# Details



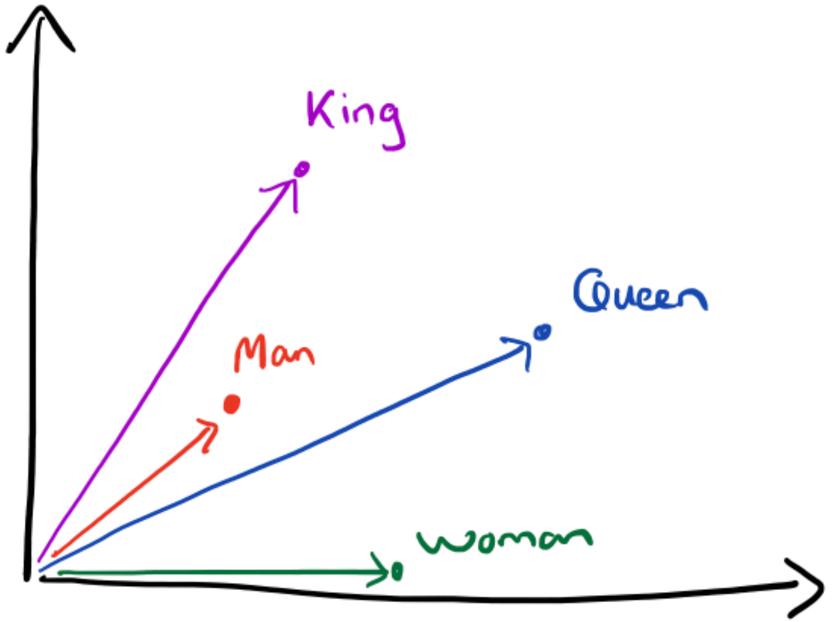

**Softmax Regression ..**
each output neuron (one per word in our vocabulary!) will produce an output between 0 and 1, and the sum of all these output values will add up to 1.

**Example:** calculating the probability that "car" appears nearby "ants".



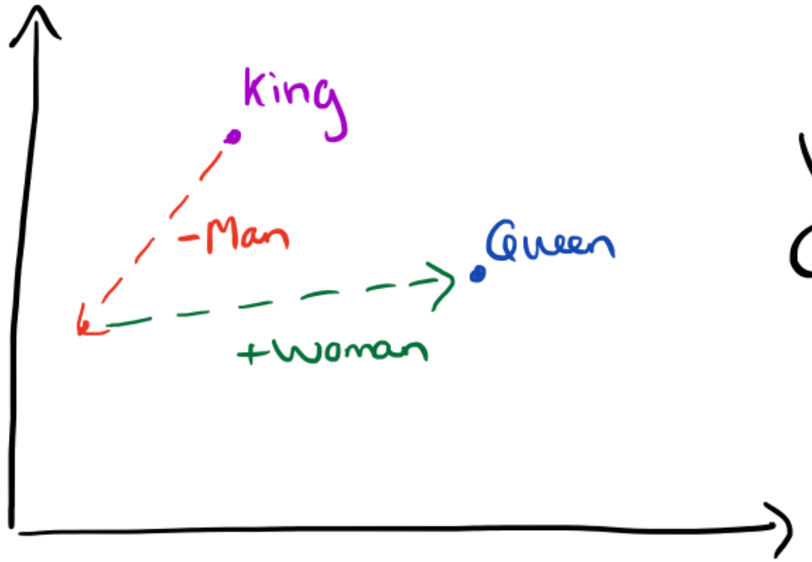


http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/




Word Vectors

Vectors for King, Man, Queen, Woman



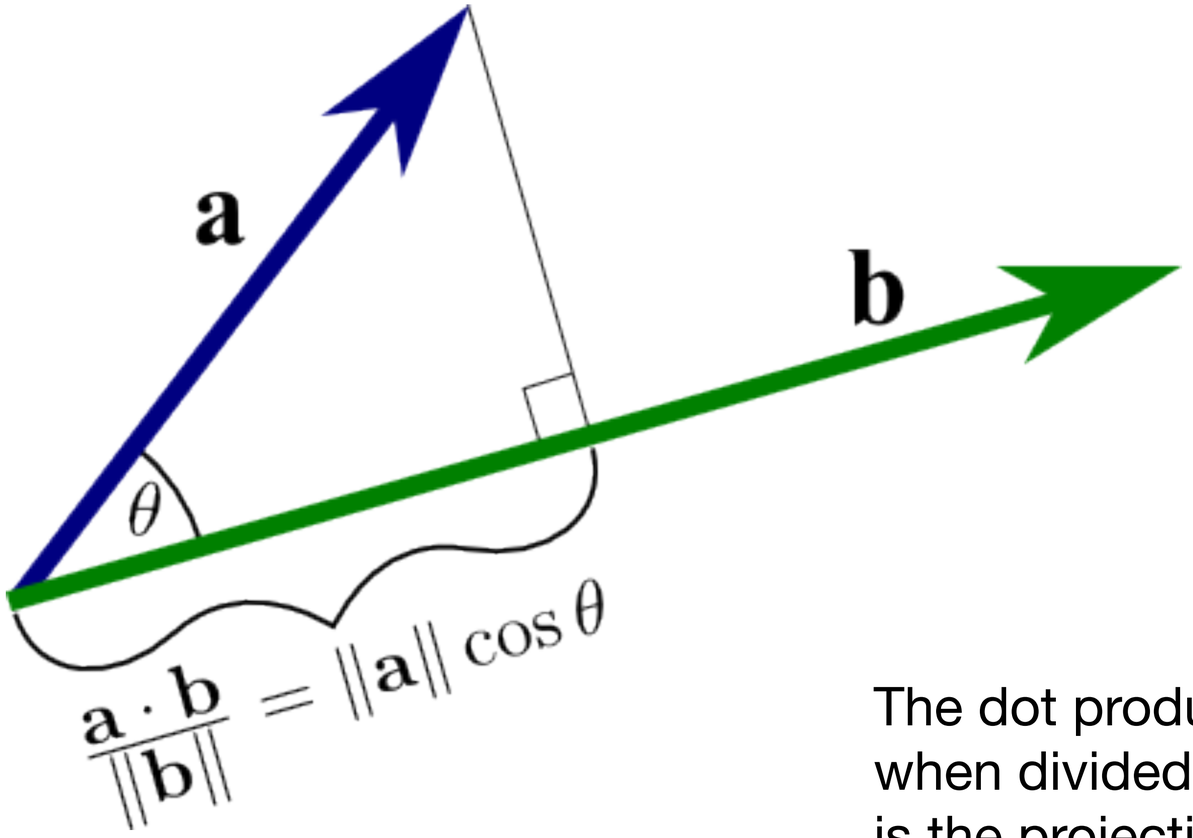

Vector Composition

The result of the vector composition King – Man + Woman = ?



wolf@paulus.com

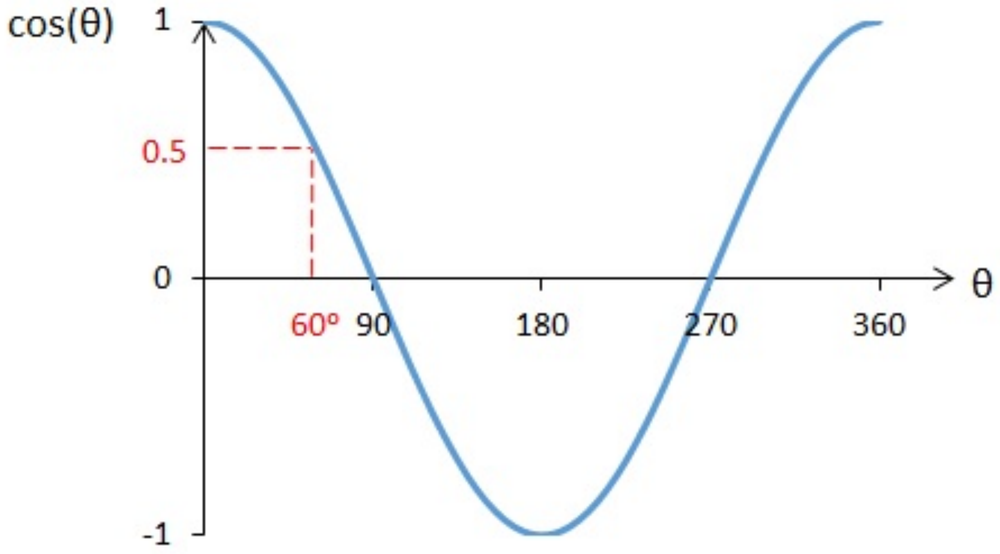
# Cosine Similarity




The dot product of vectors a and b,
when divided by the magnitude of b,
is the projection of a onto b

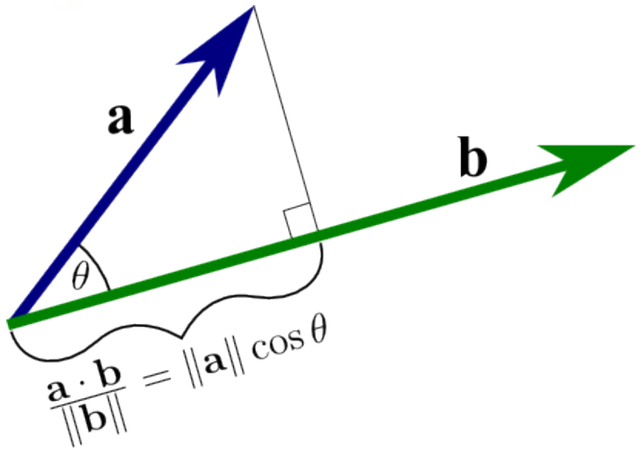

wolf@paulus.com

# Cosine Similarity




**Cosine similarity** is a measure of **similarity** between two vectors of an inner product space that measures the **cosine** of the angle between them.

The **cosine** of 0° is 1, and it is less than 1 for any angle in the interval (0,π] radians.




$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

# Cosine Similarity Example

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

```
[3,8,7,5,2,9], [10,8,6,6,4,5]
```

$$\left( similarity(A,B) = \frac{3 \cdot 10 + 8 \cdot 8 + 7 \cdot 6 + 5 \cdot 6 + 2 \cdot 4 + 9 \cdot 5}{\sqrt{3^2 + 8^2 + 7^2 + 5^2 + 2^2 + 9^2} \times \sqrt{10^2 + 8^2 + 6^2 + 6^2 + 4^2 + 5^2}} = \frac{219}{15.2315 \times 16.6433} = 0.8639 \right)$$

```
similarity = 0.8638935626791596
```

# Cosine Similarity

A list of words associated with “**Sweden**” using Word2vec, in order of proximity

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

| Word | Cosine distance |
|---|---|
| norway | 0.760124 |
| denmark | 0.715460 |
| finland | 0.620022 |
| switzerland | 0.588132 |
| belgium | 0.585835 |
| netherlands | 0.574631 |
| iceland | 0.562368 |
| estonia | 0.547621 |
| slovenia | 0.531408 |

Resources:

# Pre-trained word vectors

- English word vectors https://fasttext.cc/docs/en/english-vectors.html
  Don't use the whole 2GB file! The program would use too much memory. Instead, once you have downloaded the file, save the top n words in a separate file, **and remove the first line.** For example:
  $ cat wiki-news-300d-1M.vec | head -n 50001 | tail -n 50000 > vectors50k.vec

- GoogleNews-vectors-negative300.bin.gz
  https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz

# Raw model data

```
or 0.014047669 -0.08428601 0.09657773 0.13052626 -0.06028791 0.04916684 0.077847496 0.027510019 0.02604672 0.020193525 -0.09365112 -0.0749209 -0.06028791 -0.06350717 -0.002
their 0.037882384 0.101524785 0.00026833356 0.07424947 -0.10506048 -0.024118451 0.025002373 -0.0328314 -0.05909652 0.01382707 -0.0045774546 -0.018562367 -0.01969884 0.14445
who 0.050564233 0.06180073 -0.0069447793 0.016386557 -0.020756306 0.07022811 -0.072100855 -0.028403366 0.006866748 -0.0053061233 0.013265308 -0.01779112 -0.05087636 -0.0315
they 0.037074454 0.020784164 0.0221885 0.054488212 -0.10785296 -0.039040525 -0.010813382 -0.015447689 -0.032721013 0.05589255 -0.002475141 -0.07190197 -0.07414891 0.0379170
but -0.034717403 0.059464682 0.033293102 0.066942275 -0.010726788 0.08118532 0.047714178 -0.07050304 0.10112557 0.10468633 -0.057328228 -0.118929364 -0.066942275 -0.0519870
$ 0.04633706 -0.047928035 0.026449911 0.05409305 0.022373045 -0.040370915 0.02903524 -0.0054192487 0.11057256 -0.0342059 -0.053297564 0.15273333 0.017699564 -0.042558502 -0
had -0.030165998 0.030165998 0.00690776 -0.00016734684 0.054501593 -0.06438793 -0.021293646 -0.060331997 0.005988838 0.08314662 -0.049431678 -0.16527925 0.0066859513 0.0287
year 0.03167035 0.13218929 0.0018855124 0.07460683 -0.019152425 -0.061588187 0.096638374 -0.07911328 0.10965702 0.08862691 0.027789792 0.011203542 0.002738296 -0.052575283
were -0.045341507 -0.0111152725 0.041599732 -0.02201044 -0.016067622 -0.1047697 -0.015847517 0.021570232 0.018158613 0.055246208 -0.006988315 -0.11445429 -0.002173531 0.099
we -0.008248958 0.05387074 0.024241835 0.117617786 -0.038831826 -0.059257817 0.019752605 -0.023119528 -0.09382488 0.027384294 -0.069134116 -0.07541904 0.0078000347 -0.00259
more 0.032664012 -0.13065605 -0.04313644 0.07729652 -0.041640382 0.019822815 -0.018576099 -0.059842464 0.02144355 0.00081426266 -0.00094283046 -0.037900228 0.012965867 0.03
### 0.07172445 -0.0918823 0.021212623 0.05039463 0.00808658 -0.06516143 -0.041018885 0.03562783 0.04265964 -0.021564214 -0.09891411 -0.03703419 0.039846916 -0.012012674 0.0
up 0.058075316 -0.0097382385 0.10009729 0.0703514 -0.10481886 -0.045563154 0.029863931 -0.043674525 -0.02844746 0.07837807 0.00240505 -0.06020002 0.03375923 -0.051701196 -0
been -0.04887466 0.008558968 0.007112798 0.013340185 -0.032583106 -0.024791494 -0.08027722 0.015347116 0.0065815514 0.06799953 -0.0399025 -0.1322213 0.015819333 0.010742981
you 0.08773034 0.0056667924 0.032531586 0.12257062 -0.046803508 0.04344541 -0.012697813 0.0206733 -0.047852915 -0.0014035805 -0.03987743 -0.024765981 -0.0558284 -0.0478529
its 0.00593918 0.111031406 0.075521365 0.05176464 -0.024882039 -0.067018956 0.028383028 -0.013316267 -0.043512307 -0.022256296 -0.09152589 -0.012566054 0.06601868 0.0171298
one 0.03171886 -0.10109327 0.10855653 0.11534131 0.076328814 0.005215802 0.051225115 -0.021541687 0.10923501 0.069204785 0.03612897 -0.1031287 0.04477957 0.08209588 -0.0184
about 0.10312007 -0.041347664 0.09365495 -0.069743045 -0.02478369 0.054050863 0.0323807 -0.05604352 0.0747247 0.108101726 -0.053054534 -0.08817514 -0.011270975 -0.044834815
would 0.046935193 0.06822274 0.11182374 0.09335743 -0.05950254 -0.049756434 0.048217572 -0.015580945 0.014555039 0.048987005 -0.03616318 -0.05309063 -0.016414493 -0.0297512
which -0.051041763 0.032889586 0.067576416 0.038461044 -0.0025835398 0.0095254 -0.07656264 -0.075124845 0.080516584 0.08375163 -0.049603965 -0.0095254 -0.027318126 -0.03037
out 0.056238256 -0.08026958 0.010529188 0.084729 -0.05029236 -0.07828762 0.01015757 -0.13477361 0.018704794 0.04533745 -0.017094448 -0.13080968 0.073332705 0.015607974 -0.0
can 0.047697335 -0.017409528 0.044835497 0.073930874 -0.09730256 0.016336339 0.09730256 -0.02993008 -0.006707438 -0.009897198 -0.037203923 -0.009956819 -0.047697335 -0.0393
It -0.06630041 0.07849359 0.08001774 0.00061918487 -0.098307505 0.014669918 0.036579534 -0.022290654 0.062871076 0.072397 0.06401419 -0.16232169 -0.022481173 -0.023624284 -
all -0.005154986 -0.018445184 0.02674149 0.07764698 -0.008779585 -0.059926715 0.015626052 -0.015867691 0.091501005 0.05251642 -0.09665599 -0.041078795 -0.021586504 0.036568
also 0.033924885 0.007629227 -0.0041244295 0.0054217856 0.0104563 0.11649094 -0.044923365 -0.037642684 0.0090234 0.04058594 -0.038881946 -0.048950978 -0.0034079794 0.012625
two 0.016895514 -0.056664955 0.0012834092 0.027942581 0.099293634 -0.12112784 -0.0031679089 -0.04756737 0.040809166 0.05614509 -0.04548792 -0.012151773 -0.0059784125 0.1008
after 0.050843693 -0.027143117 0.094802305 -0.06779159 0.04660672 -0.022508927 -0.0582584 -0.0606417 0.059052832 0.021449683 -0.002283994 -0.15464957 -0.01721271 0.03707352
first 0.068539105 -0.049970742 0.015086795 0.04123269 0.035771403 -0.096118584 -0.09775697 -0.055432025 0.12560952 0.06662766 -0.0182953 -0.123425 0.01344841 0.037136726 0.
He -0.015623449 0.14181285 0.042263433 -0.01271909 0.020030065 -0.007611424 -0.049273957 -0.010265407 0.12098158 0.027441187 0.0047571403 -0.115373164 0.014421646 -0.036454
do 0.06330653 0.0065944307 0.009985852 0.11857414 -0.08943304 0.013565686 0.076872215 -0.025498465 0.0153242005 -0.020725353 -0.08692087 -0.05376031 -0.032658134 0.03592394
time -0.028173165 0.111530885 0.0013433081 0.10339843 -0.009512074 -0.089457065 0.06767369 -0.009221629 0.13070026 -0.006026734 0.008931184 -0.061574344 0.0031767422 -0.039
than -0.037155204 -0.050408013 -0.0008504873 0.10176265 0.044018265 0.009347965 0.021535818 -0.037155204 0.071943834 0.077623606 -0.015146069 -0.09371631 0.026150636 0.0442
when 0.10877369 0.015461404 0.08826208 0.014140581 0.04382026 -0.05780545 -0.02020083 -0.085154265 0.1137462 0.09945024 0.004176133 -0.16036351 -0.07147986 -0.009867328 -0.
We -0.102923416 0.11613656 0.067456566 0.04155185 -0.06954285 0.019645855 0.03198971 -0.014169355 -0.06397942 0.015212498 -0.035292994 -0.10918227 0.014864784 0.040160995 -
over 0.04922319 0.05300959 0.008181327 0.033671908 -0.020825196 -0.11467381 0.0027214745 -0.1795835 0.08059621 0.1525378 -0.097905464 -0.019608138 -0.006761427 0.015280825
last 0.080515 0.040808972 0.01709565 -0.021645298 0.07224291 -0.036121454 -0.13621373 -0.052665632 0.044393543 0.13125047 -0.045772225 -0.11084599 -0.021645298 0.07555174 -
new 0.0064339901 0.016484698 0.04757609 -0.02837872 -0.0745637 -0.06260012 -0.065660566 -0.04256808 -0.05091476 0.12464379 0.021144928 0.053975213 0.013354692 -0.048132535 -
other -0.025468545 -0.04947338 0.026346771 0.07904031 -0.017344957 0.03600725 0.026054028 0.034397174 0.0077576605 0.027371367 0.011929233 -0.04010564 -0.042447574 0.121780
```


python™

# word.py

```python
class Word:
    """A single word (one line of the input file)"""

    def __init__(self, text, vector, frequency):
        self.text = text
        self.vector = vector
        self.frequency = frequency

    def __repr__(self):
        vector_preview = ', '.join(map(str, self.vector[:2]))
        return f"{self.text} [{vector_preview}, ...]"

    def __str__(self):
        return self.text
```



wolf@paulus.com

# vector.py

```python
def add(v1, v2):
    assert len(v1) == len(v2)
    return [x + y for (x, y) in zip(v1, v2)]


def sub(v1, v2):
    assert len(v1) == len(v2)
    return [x - y for (x, y) in zip(v1, v2)]


def dot(v1, v2):
    assert len(v1) == len(v2)
    return sum([x * y for (x, y) in zip(v1, v2)])


def normalize(v):
    length = math.sqrt(sum([x * x for x in v]))
    return [x / length for x in v]


def cosine_similarity_normalized(v1, v2):
    """
    Returns the cosine of the angle between the two vectors.
    Each of the vectors must have length (L2-norm) equal to 1.
    Results range from -1 (very different) to 1 (very similar).
    """
    return dot(normalize(v1), normalize(v2))
```

# load.py

```python
def load_words(file_path):
    """Load and cleanup the data."""
    print(f"Loading {file_path}...")
    words = load_words_raw(file_path)
    print(f"Loaded {len(words)} words.")

    # num_dimensions = most_common_dimension(words)
    words = [w for w in words if len(w.vector) == 300]
    # print(f"Using {num_dimensions}-dimensional vectors, {len(words)} remain.")

    words = remove_stop_words(words)
    print(f"Removed stop words, {len(words)} remain.")

    words = remove_duplicates(words)
    print(f"Removed duplicates, {len(words)} remain.")

    return words


def load_words_raw(file_path):
    """Load the file as-is, without doing any validation or cleanup."""

    def parse_line(ln, freq):
        tokens = ln.split()
        word = tokens[0]
        vector = v.normalize([float(x) for x in tokens[1:]])
        return Word(word, vector, freq)

    words = []
    # Words are sorted from the most common to the least common ones
    frequency = 1
    with open(file_path) as f:
        for line in f:
            w = parse_line(line, frequency)
            words.append(w)
            frequency += 1
    return words
```

# *Thanks*

**https://wolfpaulus.com**

**https://github.com/wolfpaulus/word2vec**



wolf@paulus.com