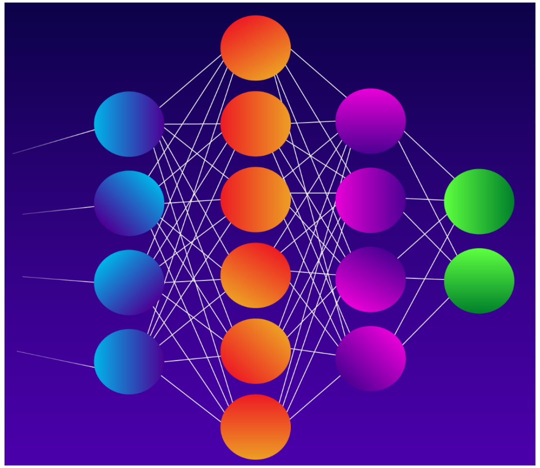
# Applied Deep Learning for NLP

Week 3 - Text Representations

Juan Carlos Medina Serrano

Technische Universität München
Hochschule für Politik
Political Data Science

Munich, 5. November 2020



## Text Representations

Transform the text into a numerical form so that it can be fed into NLP and ML algorithms

A good text representation will give you a great improvement in your model! Important to choose wisely.
(Otherwise: Garbage in, garbage out)

## Text Representations

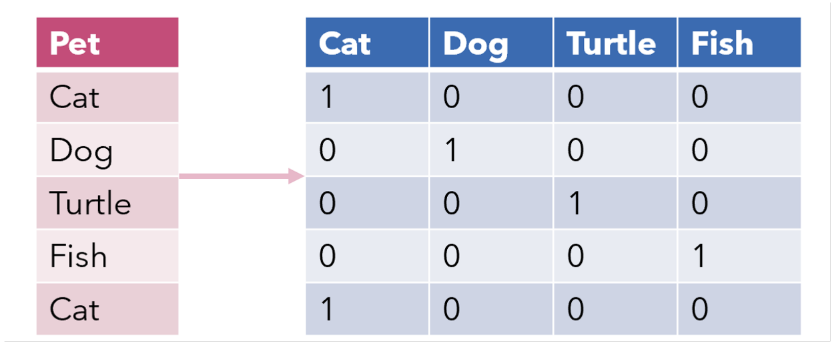
### Word classes

Simplest method to convert text into numbers.

Example:
Class1 = {Rome, Berlin, Paris, Madrid}
Class2 = {Italy, Germany, France, Spain}

## One-Hot Encoding

Each word w in the corpus vocabulary ($V$) is given a unique integer Id between 1 and the length of the vocabulary ($|V|$)



| Pet |
|---|
| Cat |
| Dog |
| Turtle |
| Fish |
| Cat |

| Cat | Dog | Turtle | Fish |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |

One-hot encoding is the way to go if the number of words is lower than 10. For example, when a feature is **categorical**

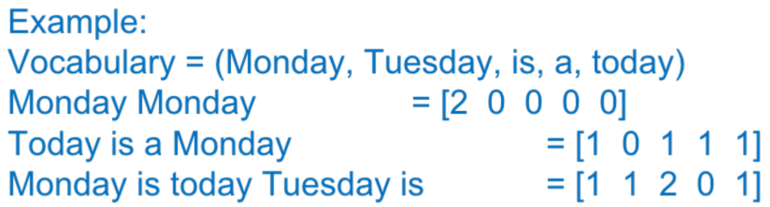
However, normally the vocabulary is large and it will be inefficient to store, compute and to learn from (sparcity leads to overfitting)

No notion of (dis)similarity between words

## Text Representations

### Bag of Words

It begins with one-hot encoding of the words. A document is represented as a vector of size $|V|$ with counts of words as features.



Example:

Vocabulary = (Monday, Tuesday, is, a, today)

| | |
|---|---|
| Monday Monday | = [2 0 0 0 0] |
| Today is a Monday | = [1 0 1 1 1] |
| Monday is today Tuesday is | = [1 1 2 0 1] |

Documents having the same words will have similar vector representations. Still does not capture similarity between words and the word order is lost.

## Text Representations

### Bag of N-Grams

**N-Gram** N continuous words (2-gram is a bigram, 3-gram is a trigram)

Example:
D1) dog bites man
D2) man bites dog
D3) dog eats meat
D4) man eats food
Vocabulary = {dog bites, bites man, man bites, bites dog, dog eats, eats meat, man eats, eats food}
D1 = [1, 1, 0, 0, 0, 0, 0, 0]
D2 = [0, 0, 1, 1, 0, 0, 0, 0]

Now order is important and captures some context. However, vocabulary size is now even larger!
Bag of words is the special case of N=1.

# Text Representations

## TF-IDF

Bag of words treats all words equally. **Term Frequency-Inverse Document Frequency** quantifies the importance of a word relative to other words in the document and in the corpus.

Intuition: If a word $w$ appears many times in document $D_j$ but does not occur much in the rest of the documents, then the word $w$ must be of great importance to document $D_j$

**Term Frequency** How often a term occurs in a document

**Inverse Document Frequency** It weighs down terms that are very common accross a corpus.
Example, word $w$

$$TF(w,d) = \frac{\#w \quad in \quad d}{\#words \quad in \quad d}$$

$$IDF(w) = \frac{\#documents \quad in \quad corpus}{\#documents \quad that \quad include \quad word \quad w}$$

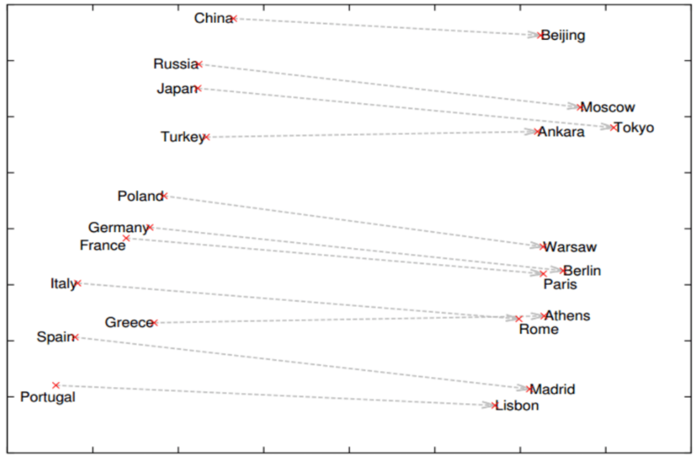
$$TFIDF = TF * log(IDF)$$

## Word Embeddings

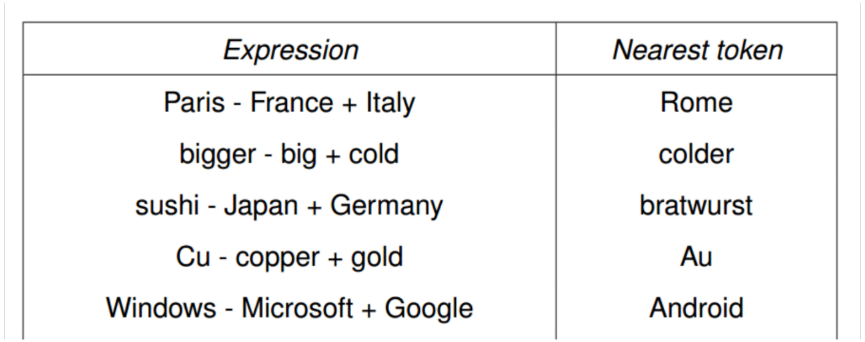
Use neural network architectures to create dense, low-dimensional representations.

Intuition: Words that occur in similar contexts have similar meanings.
Example: dog and cat occur in similar contexts (I have a dog/cat as a pet)

They create a **vector space** where common vector operations lead to contextual meanings.



## Word Embeddings



| *Expression* | *Nearest token* |
|---|---|
| Paris - France + Italy | Rome |
| bigger - big + cold | colder |
| sushi - Japan + Germany | bratwurst |
| Cu - copper + gold | Au |
| Windows - Microsoft + Google | Android |

Note: compound words like *New York* should be better pre-processed as *New_ York*

How to create good word embeddings?

## Text Representations

### Word2Vec

Takes a large corpus of text as input (for example Wikipedia) and learns to represent the words based on the contexts in which they appear

Two similar variants:

- Skipgram
- CBOW

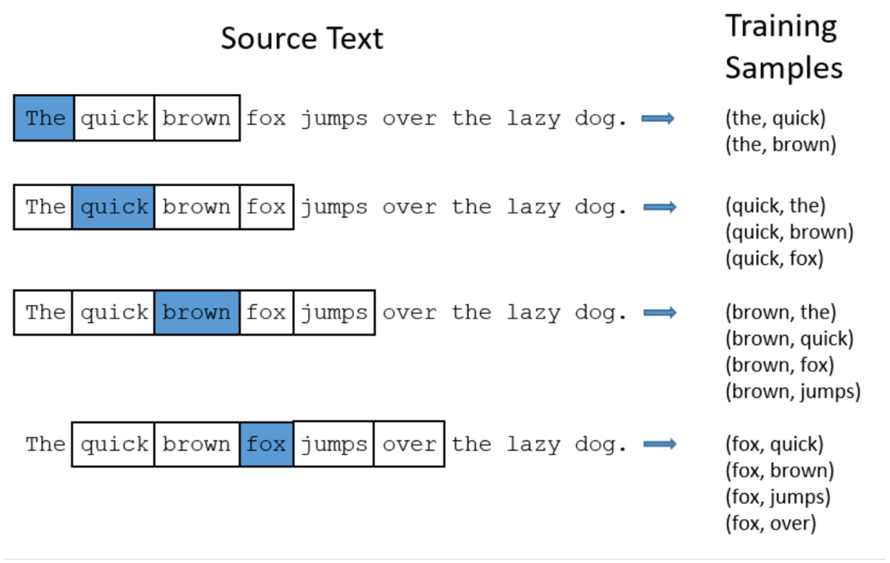
Intuition: Train a simple neural network with a single hidden layer to perform a certain task. However, the model and the outputs are not important. Instead, the goal is actually just to learn the weights of the hidden layer. These weights will be the word embeddings.

Nowadays, no state-of-the-art anymore. However, important to understand how they work. They revolutionized NLP!

# Text Representations

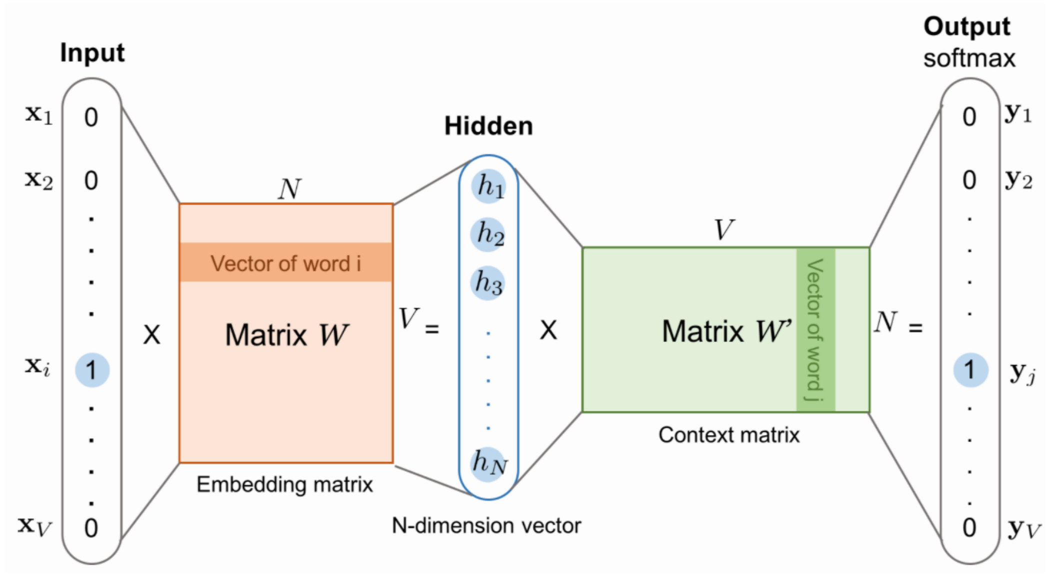
## Skipgram

Task: **Predict the context words from the center word.**
Context words defined by a **sliding window**. Normally size 5 (two to the left, and two to the right)



## Skipgram

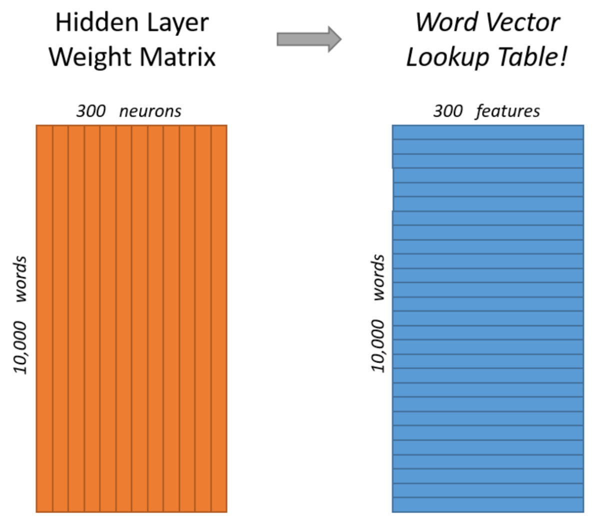
Neural Network architecture for a vocabulary of V words and dimension of word embedding N:



Input and output are one-hot encoding vectors. W= word embedding matrix. W' = context embedding matrix

## Skipgram

**The Hidden Layer** We are only multiplying one-hot vector to a matrix. This means we select the row of the matrix.

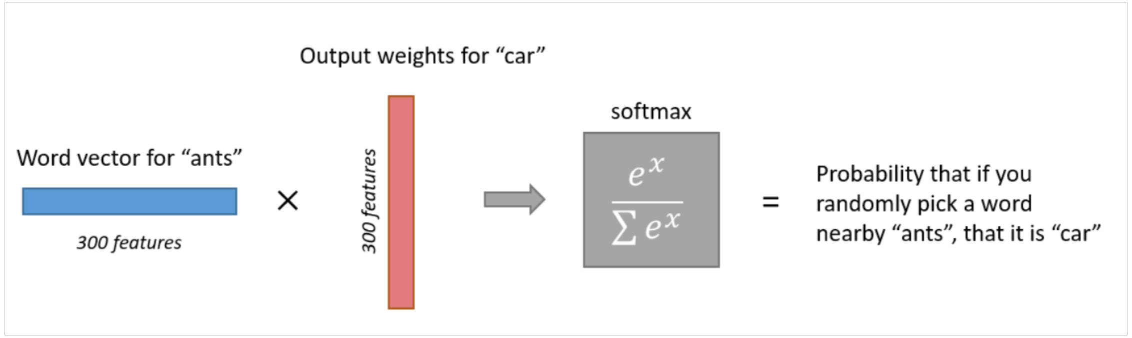


**Always think of weights between neural network layers as matrices!** The weight between word 800 and neuron 1 is in position $W_{800,1}$, and neuron 2 is in $W_{800,2}$

**Each neuron represents one dimension of the word embedding**

## Skipgram

**The Output Layer** 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 (using softmax)



If two different words have very similar contexts, then the model needs to output very similar results for these two words = the two words should have similar word embeddings.

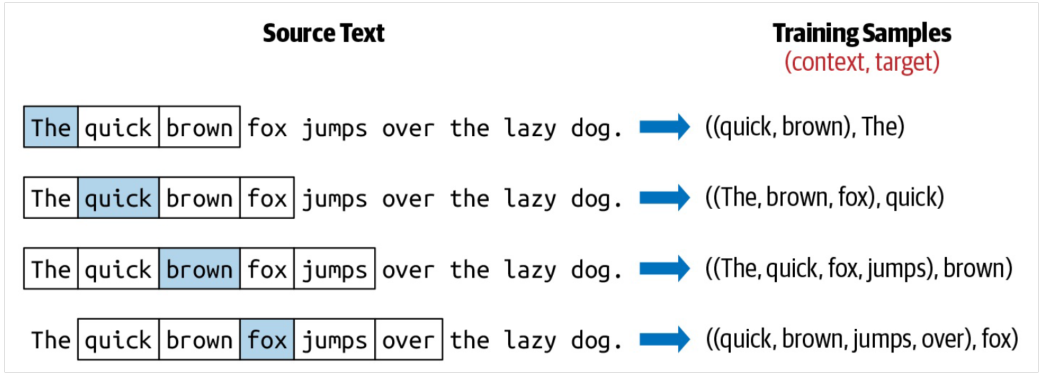
## Skipgram

Training this network is a hard task because of the huge number of weights.
How to make it faster and more efficient?

1. **Subsampling frequent words** Common words like "the"do not convey much context. Subsampling means that each word has a probability to be erased. The higher frequency words have a higher probability of being erased.
2. **Negative Sampling** Instead of having one output with 1 and 99,999 with 0, randomly select just a small number of "negative" words (have 0 as output) to update the weights for one training sample (for example, 5 negative words).
   More frequent words are more likely to be selected as negative sample
3. **Hierarchical Softmax** Uses a tree structure to make the sum calculation faster. (Computer Science power!)

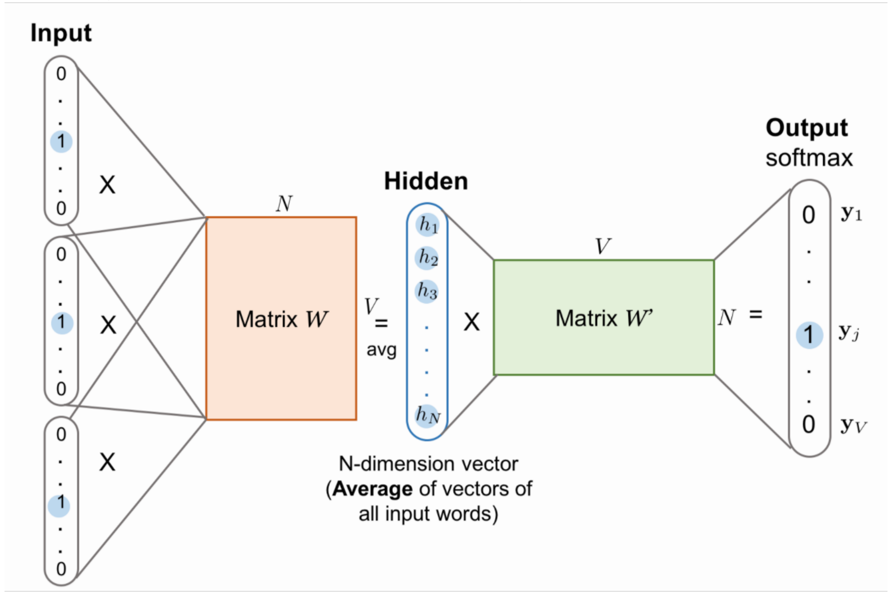
## CBOW

Now the other way around, use the context words as inputs and the center word as output:


Source Text
Training Samples
(context, target)
The quick brown fox jumps over the lazy dog.
((quick, brown), The)
The quick brown fox jumps over the lazy dog.
((The, brown, fox), quick)
The quick brown fox jumps over the lazy dog.
((The, quick, fox, jumps), brown)
The quick brown fox jumps over the lazy dog.
((quick, brown, jumps, over), fox)

## CBOW

Few changes in architecture:
We use several one-hot encoding vectors as input and we average the word vectors of the context words

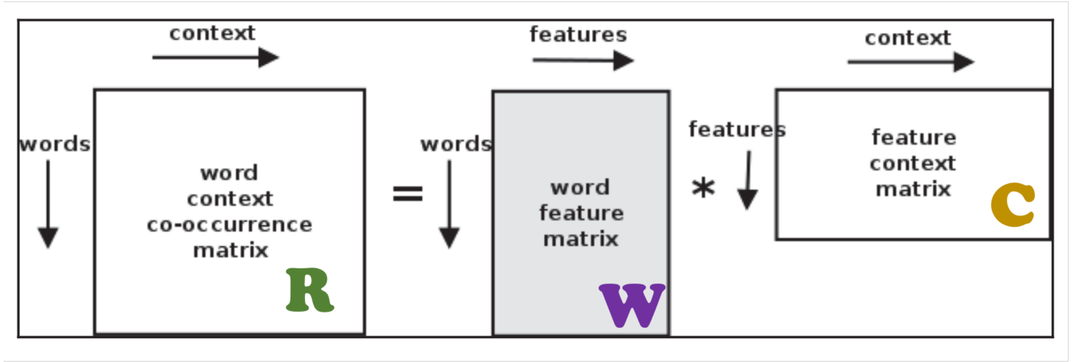


## Text Representations

### GloVe

A non-neural network approach to get word embeddings. Better results than cbow and skipgram.

Based on **matrix factorization** and **stochastic gradient descent**. Uses teh **global** co-occurances of the words.



Initialize W and C randomly. Multiply them (R') and compare with R. The difference tells us how much to change W and C to be more similar to R. Alternate between optimizing W and C.

GloVe implements a smart trick inspired by the word2vec model. It changes the **loss function** to use the ratios of co-occurrence probabilities rather than the probabilities themselves.
(What?? Go and read the original GloVe paper to get an idea of what this means)

## Text Representations

### FastText

The creator of Word2vec (Mikolov) was not happy that a non-neural approach was performing better...

He improved word2vec -> FastText

Two main tricks:

- Uses position weights. Words before and after have different weights, also closer context words than far away ones.
- Exploits **sub-word** information. (morphology)

  Has better results than GloVe (but not huge improvements)

# Text Representations

## Subword Information

Representing words using its **character n-grams**. Incorporates information about the structure of the word.

Idea: same as word2vec but now the word is computed summing the ngram embeddings

Example: disastrous and disaster will have similar word embeddings because of: dis, isa, sas, ast,disa, isas, sast, di, is, sa, as, st, disas, isast

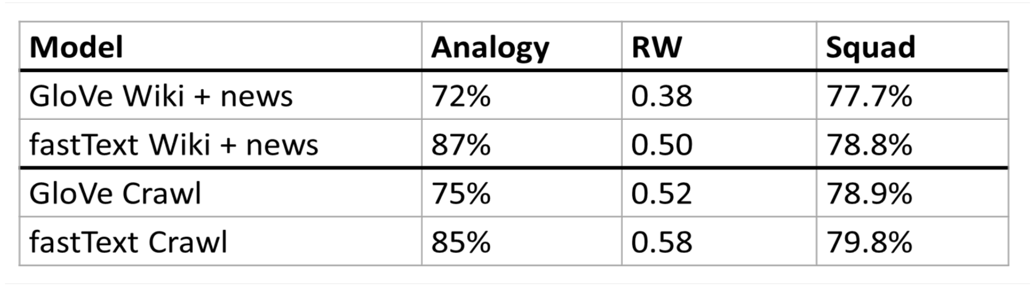
Advantage: Build vectors for unseen words. -> called Out-of-Vocabulary (OOV) words

## Evaluating Word Embeddings

How do we know which embeddings are better?

Benchmarking tasks:

- **Word Analogies** List of human generated analogies (Paris -> France / Warsaw -> ?)
- **Word similarity** List of human generated similarity. Compare cosine distance between word embeddings. Normally, rare words (RW) are used to benchmark.
- **Evaluate on downstream task**. A downstream task means using the embeddings for a NLP task like classification with the same algorithm and see which embeddings give best metrics. For example, SQuAD (Stanford Question Answering Dataset)



| Model | Analogy | RW | Squad |
|---|---|---|---|
| GloVe Wiki + news | 72% | 0.38 | 77.7% |
| fastText Wiki + news | 87% | 0.50 | 78.8% |
| GloVe Crawl | 75% | 0.52 | 78.9% |
| fastText Crawl | 85% | 0.58 | 79.8% |

## Visualizing Word Embeddings

Use a **dimensionality reduction** algorithm like PCA, T-SNE or UMAP! Use first two or first three components to visualize.

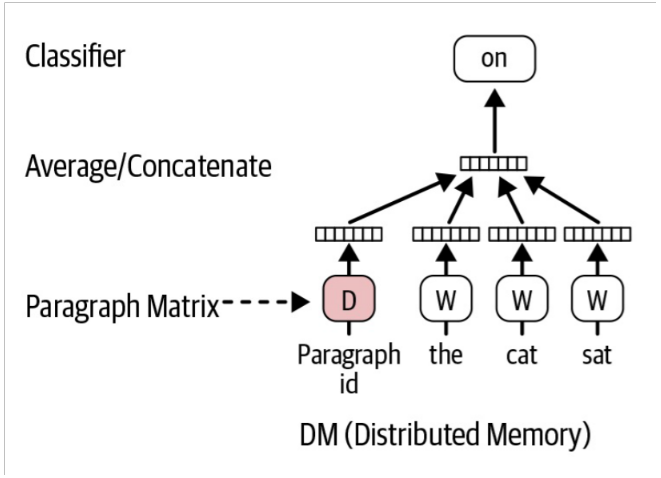
`http://projector.tensorflow.org/`

# Text Representations

## Doc2Vec

**Goal:** Create a numeric representation of a document (a document can be a sentence, a paragraph, or a complete piece of text)

Same as CBOW, but adds a document-unique **token**. The vector representation of the token intends to represent the concept of a document



Documents with similar topics will have similar embeddings, given that they are composed of similar wrods.

## Contextual Word Embeddings

State-of-the-art embeddings also use the context to generate embeddings.

Until now, words get one **fixed** representation. However, words can have different meanings depending on the context. (homonyms). *bat* for baseball and **bat** the animal.

Idea: Use **language modeling** to pre-train contextual embeddings.

Next week: Language Modeling
In four weeks: Transformers

**Final note** Hand-crafted feature representations are still important in domain-specific tasks. For example, grammar correction needs custom features to incorporate domain knowledge.