UNIVERSITÉ DU LUXEMBOURG

# Natural Language Processing
## Lecture 8: Text Representation (Word Embeddings)

Salima Lamsiyah
University Luxembourg, FSTM, DCS, MINE Research Group
Salima.lamsiyah@uni.lu

# Lecture Plan

1. Recap
2. Bag-of-words (BOW) Methods
3. Neural Word Embeddings Models
4. Sentence Embeddings Models (Contextual Embeddings)
5. Topic Modeling Methods
6. Summary

# Text Representation Types

1. Bag-of-words Models
2. Neural Word Embeddings Models
3. Sentence Embeddings Models
4. Topic Modeling Methods
5. …

# Bag-of-words (BOW) Representation

- Usually, the **vocabulary** is built using the words contained in the given corpus after removing the **stop-word list.**

- Several BOW methods exist, including:
  - **One-hot encoding**
  - **Term Count**
  - **Term Frequency-Inverse Document Frequency (TF-IDF)**
  - **Positive Pointwise Mutual Information**

# Sparse versus dense vectors

- tf-idf (or PMI) vectors are
  - **long** (length $|V| = 20{,}000$ to $50{,}000$)
  - **sparse** (most elements are zero)

- Alternative: learn vectors which are
  - **short** (length 50-1000)
  - **dense** (most elements are non-zero)
  - Usually called ***Embeddings***

# Sparse versus dense vectors

- **Why dense vectors?**
  - Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
  - Dense vectors may **generalize** better than explicit counts
  - Dense vectors may do better at capturing synonymy:
    - ***car*** and ***automobile*** are synonyms; but are distinct dimensions
      - a word with ***car*** as a neighbor and a word with ***automobile*** as a neighbor should be similar, but aren't



- **In practice, they work better**

# Common methods for getting short dense vectors

- “Neural Language Model”-inspired models
  - Word2vec (skipgram, CBOW), GloVe, FastText

- Alternative to these "static embeddings":
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

- Topic Modeling Models
  - A special case of this is called LSA – Latent Semantic Analysis

# Neural Word Embeddings

- **Word embedding models** learns high-quality word vectors from huge data sets with billions of words, and with millions of words in the **vocabulary.**

- **The learned vectors** reflect similarities and dissimilarities between words rather than treating individual words as independent symbols.

- Several methods have been proposed to represent words as dense vectors in a low-dimensional vector space, inspired from **neural networks** and **language Models.**

- The vectors' estimation is based on the idea that words in similar contexts have similar meanings and words representing closer meanings are placed close to one another in the semantic space.

# *Word2vec*

# Word2vec

https://arxiv.org/pdf/1301.3781

Cited More than 50212 times



## Efficient Estimation of Word Representations in Vector Space

**Tomas Mikolov**
Google Inc., Mountain View, CA
tmikolov@google.com

**Kai Chen**
Google Inc., Mountain View, CA
kaichen@google.com

**Greg Corrado**
Google Inc., Mountain View, CA
gcorrado@google.com

**Jeffrey Dean**
Google Inc., Mountain View, CA
jeff@google.com

### Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

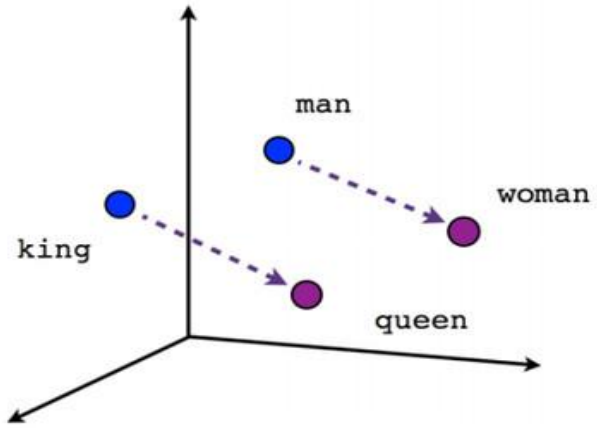
### 1 Introduction

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train N-grams on virtually all available data (trillions of words [3]).

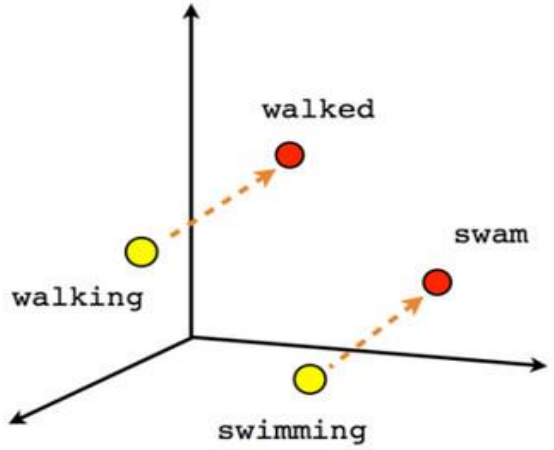
However, the simple techniques are at their limits in many tasks. For example, the amount of

# Word2vec Intuition

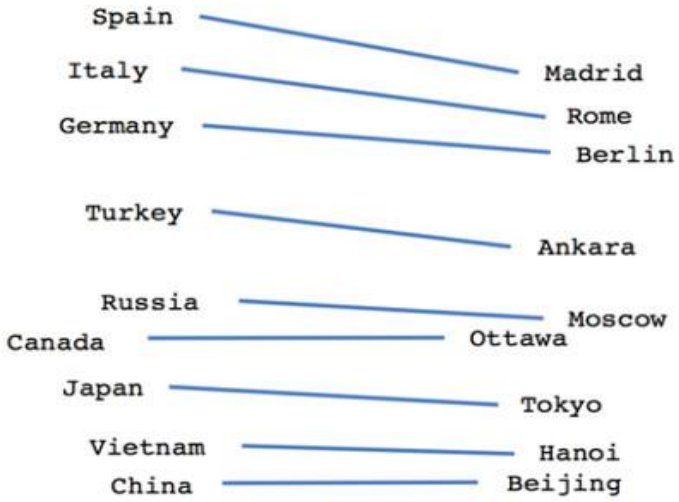
***"You shall know a word by the company it keeps."***
***— J.R. Firth (1957)***




Male-Female




Verb tense




Country-Capital

- Word2Vec embeddings make meaning *geometric* - words become points in space, and relationships become vectors.
- Word2Vec doesn't just capture meaning - it captures **relationships between meanings**.
- That's why we can do analogies like:

$$\text{king} - \textit{man} + \textit{woman} \approx \textit{queen}$$

Source

# Word2vec

- Instead of **counting** how often each word $\boldsymbol{w}$ occurs near "***apricot***"
  - Train a classifier on a binary **prediction** task:
    - Is $\boldsymbol{w}$ likely to show up near "***apricot***"**?**
- We don’t actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word $\boldsymbol{c}$ that occurs near “**apricot**” in the corpus serves as the gold "correct answer" for supervised learning
    - No need for human labels
    - Bengio et al. (2003) – “A Neural Probabilistic Language Model”
    - Collobert et al. (2011)— “Natural Language Processing (Almost) from Scratch”

# Approach: predict if candidate word $c$ is a "neighbor"

1. Treat the target word $\boldsymbol{w}$ and a neighboring context word $\boldsymbol{c}$ as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

# Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**
- Word2vec provides various options. We'll do:
  - **Skip-gram with negative sampling (SGNS)**
  - **CBOW (Continuous Bag-of-words) Model**

# Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

# Skip-Gram Classifier

(assuming a +/- 2 word window)

…lemon, a [tablespoon of apricot jam, a] pinch…
c1 c2 [target] c3 c4

**Goal**: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

- (apricot, jam)
- (apricot, *aardvark*)
- …

Assigns each pair a probability:

$P(+|w, c)$

$P(-|w, c) = 1 - P(+|w, c)$

# Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product

- So:
  - Similarity(w,c) $\propto w \cdot c$

- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

# Turning dot products into probabilities

- Sim(w,c) ≈ w · *c*
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

Because $1-\sigma(x) = \sigma(-x)$

$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$

# How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words.

- We'll assume independence and just multiply them:

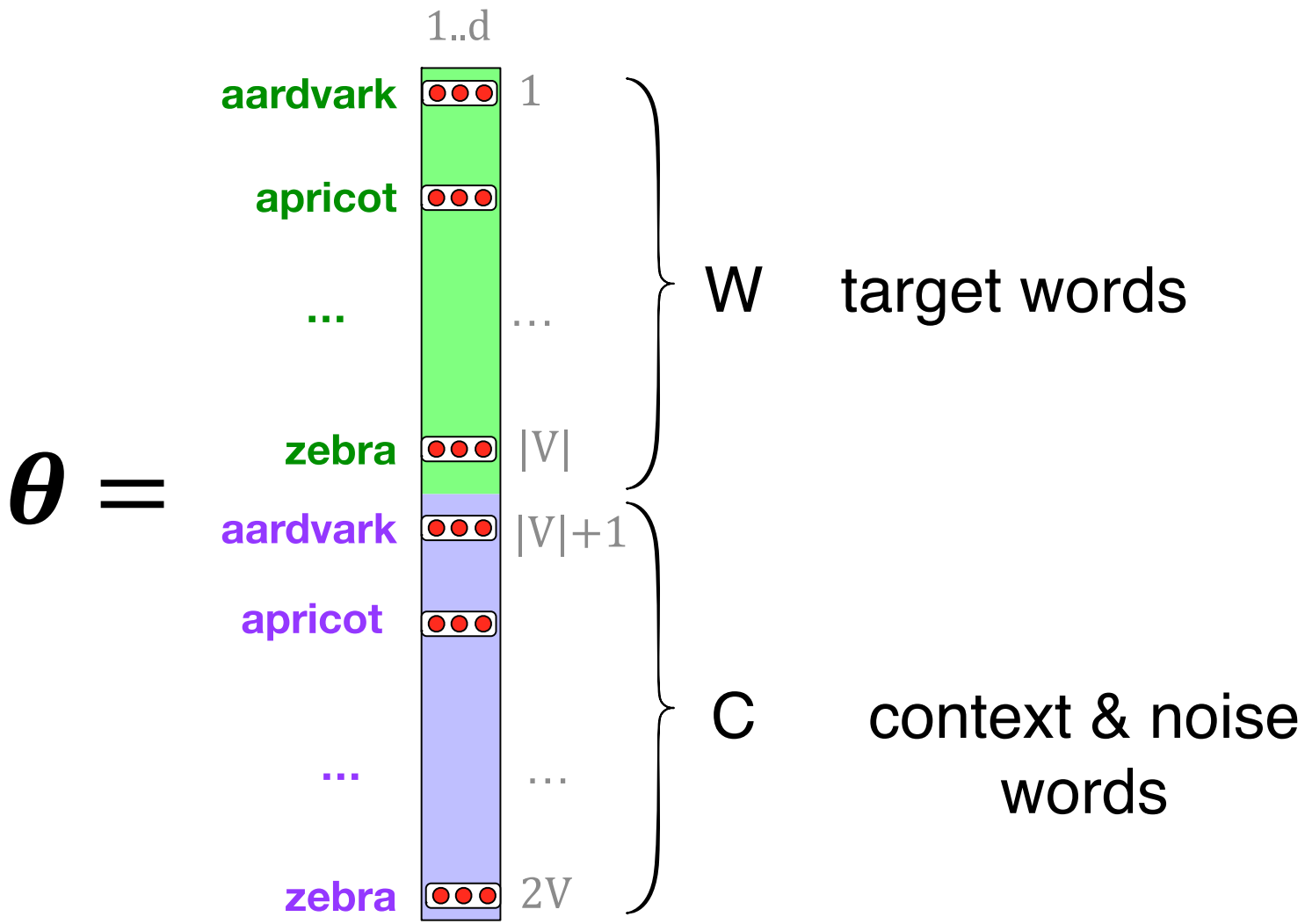
$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

# Skip-gram classifier: summary

- skip-gram trains a probabilistic classifier that, given
  - a test target word *w*
  - its context window of *L* words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).
- **To compute this, we just need embeddings for all the words.**

# These embeddings we'll need: a set for w, a set for c



# *Word2vec Learning the embeddings*

# Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

- c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

For each positive example we'll grab $k$ negative examples, sampling by frequency

# Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

**negative examples -**

| t | c | t | c |
|---|---|---|---|
| apricot | aardvark | apricot | seven |
| apricot | my | apricot | forever |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | if |

k=2

The noise words are chosen according to their weighted unigram frequency.

# Weighted Unigram Frequency in Skip-Gram (Negative Sampling)

- Weighted Unigram Formula:

$$P(w_i) = \frac{f_i^{3/4}}{\sum_j f_j^{3/4}}$$

*Mikolov et al., 2013*

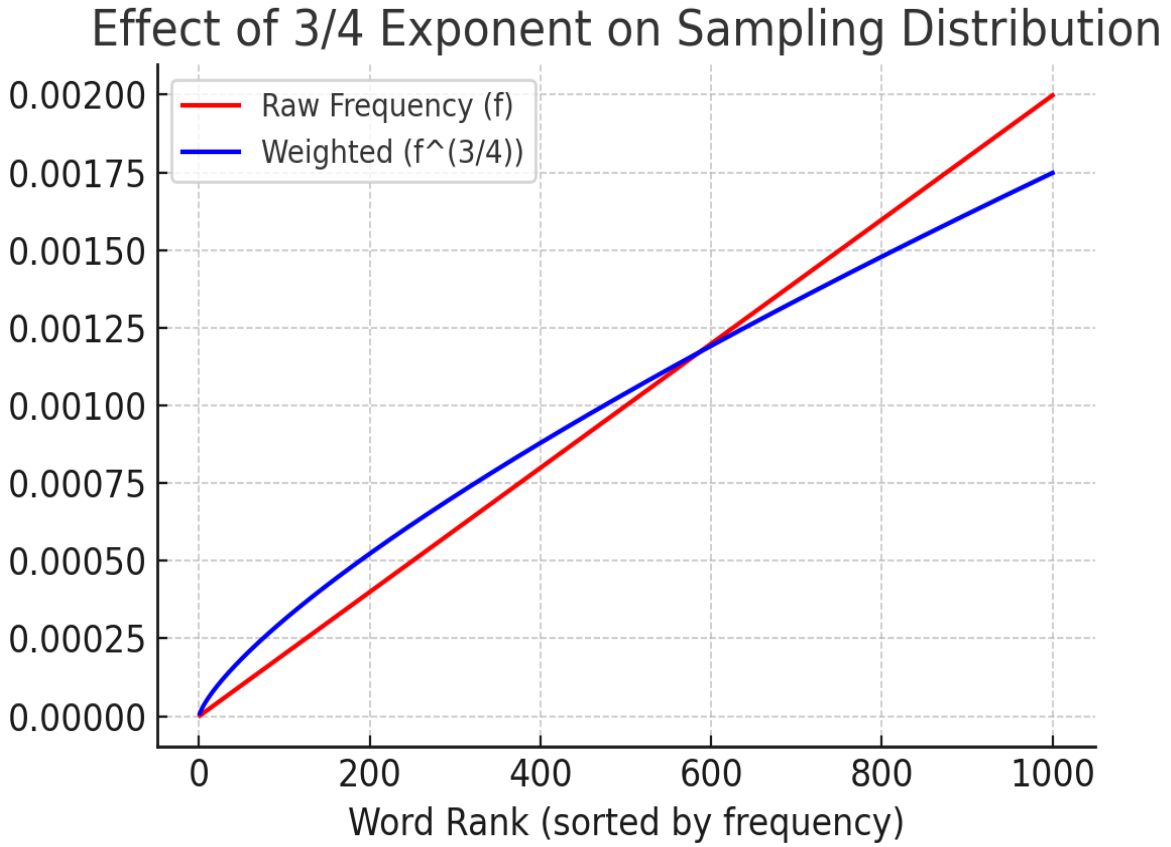
- Where:
  - $f_i$: frequency of word i in the corpus
  - The 3/4 exponent makes common words less dominant and rare words slightly more frequent.

- **Why use the 3/4 exponent?**
  - Using raw frequency ($f_i$): very common words (the, of, and) dominate
  - Using uniform sampling: rare words appear too often
  - Using 3/4 power: balances both → better training stability

# Weighted Unigram Frequency in Skip-Gram (Negative Sampling)

- Using f^(3/4) flattens the frequency curve:
- Raw frequency → common words dominate.
- Weighted 3/4 → smoother distribution, fairer sampling.

- **Effect**: Better balance between frequent and rare words → improved Skip-Gram training stability.



# Word2vec: How to learn vectors?

- Given the set of positive and negative training instances, and an initial set of embedding vectors

- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the target word, context word pairs ($w$ , $c_{pos}$) drawn from the positive data
  - **Minimize** the similarity of the ($w$ , $c_{neg}$) pairs drawn from the negative data.

# Loss function for one *w with* $c_{pos}$ , $c_{neg1}$ …$c_{negk}$

- ***Maximize*** the similarity of the target with the actual context words and ***minimize*** the similarity of the target with the *k* negative sampled non-neighbor words.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

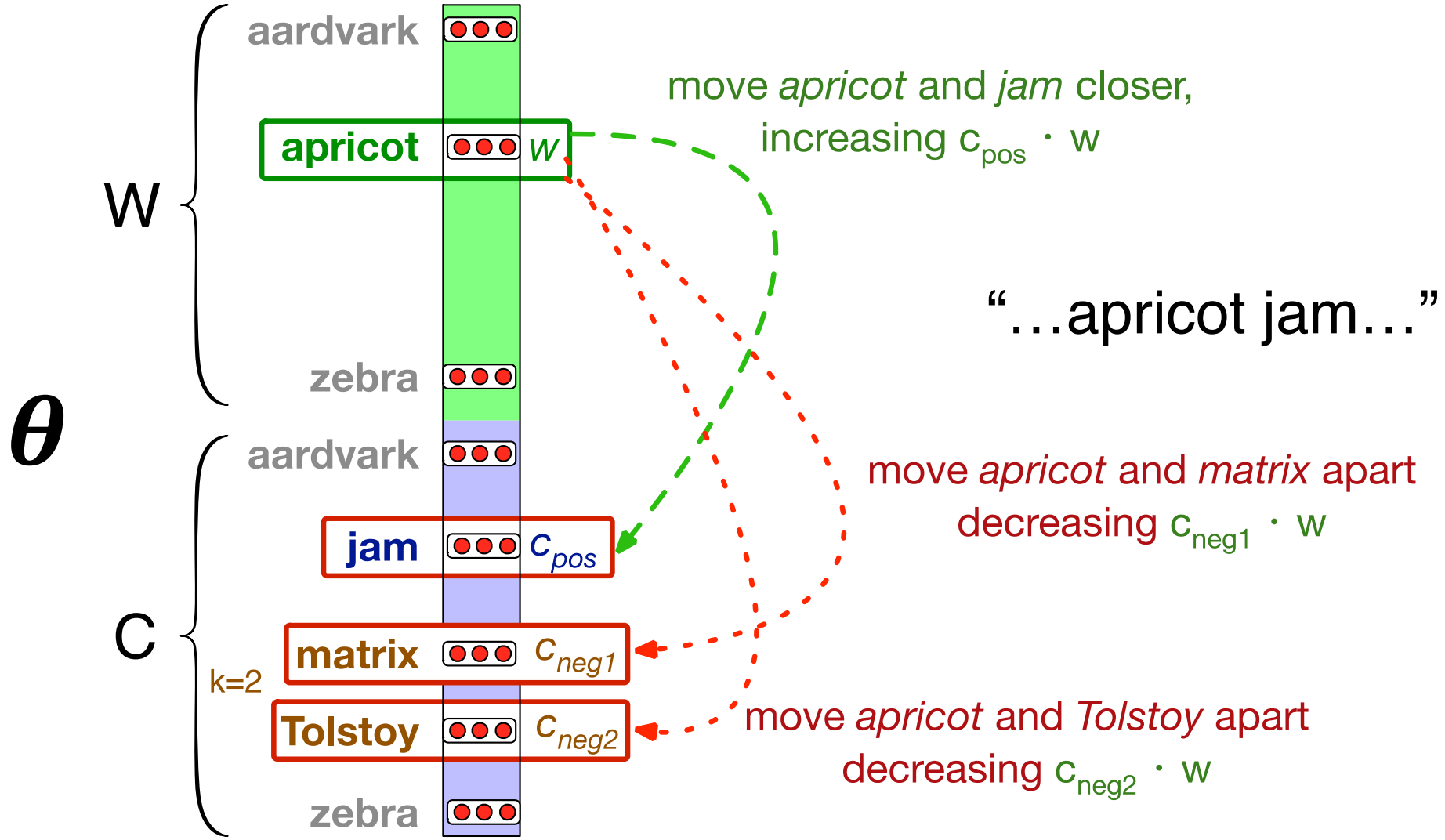
Because $1-\sigma(x)=\sigma(-x)$

# Learning the classifier

- How to learn?
  - Stochastic gradient descent!

- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

# Intuition of one step of gradient descent



# Reminder: gradient descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
  - Higher learning rate means move *w* faster

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

# The derivatives of the loss function

## Proof as an Exercise

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Check Chap 5
Chap 6

# Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta \left[ [\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i} \right]
\end{aligned}
$$

# Two sets of embeddings

- Skip-gram with negative sampling (SGNS) learns two sets of embeddings
  - Target embeddings matrix W
  - Context embedding matrix C

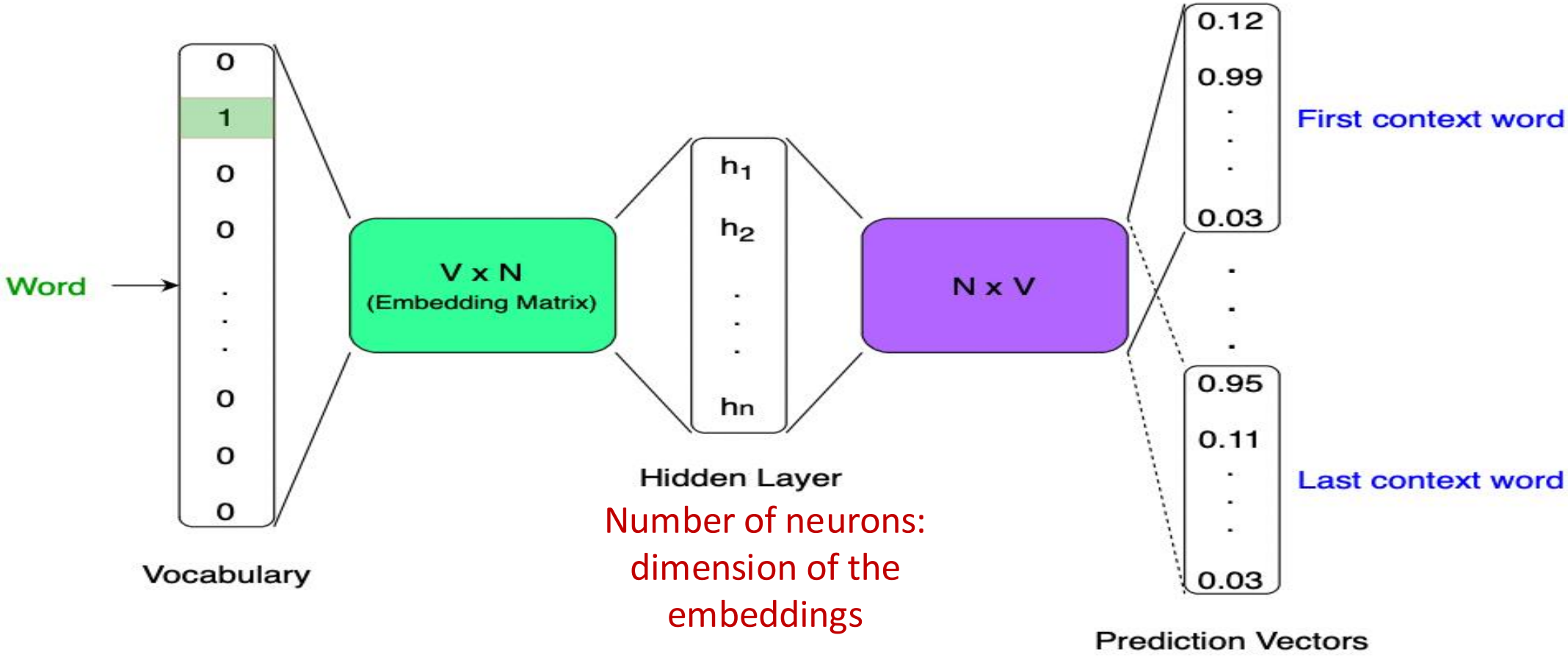
- It's common to just add them together, representing word *i* as the vector $w_i + c_i$

# Summary: How to learn word2vec (skip-gram) embeddings

- Start with V random d-dimensional vectors as initial embeddings

- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

# Summary: How to learn word2vec (skip-gram) embeddings

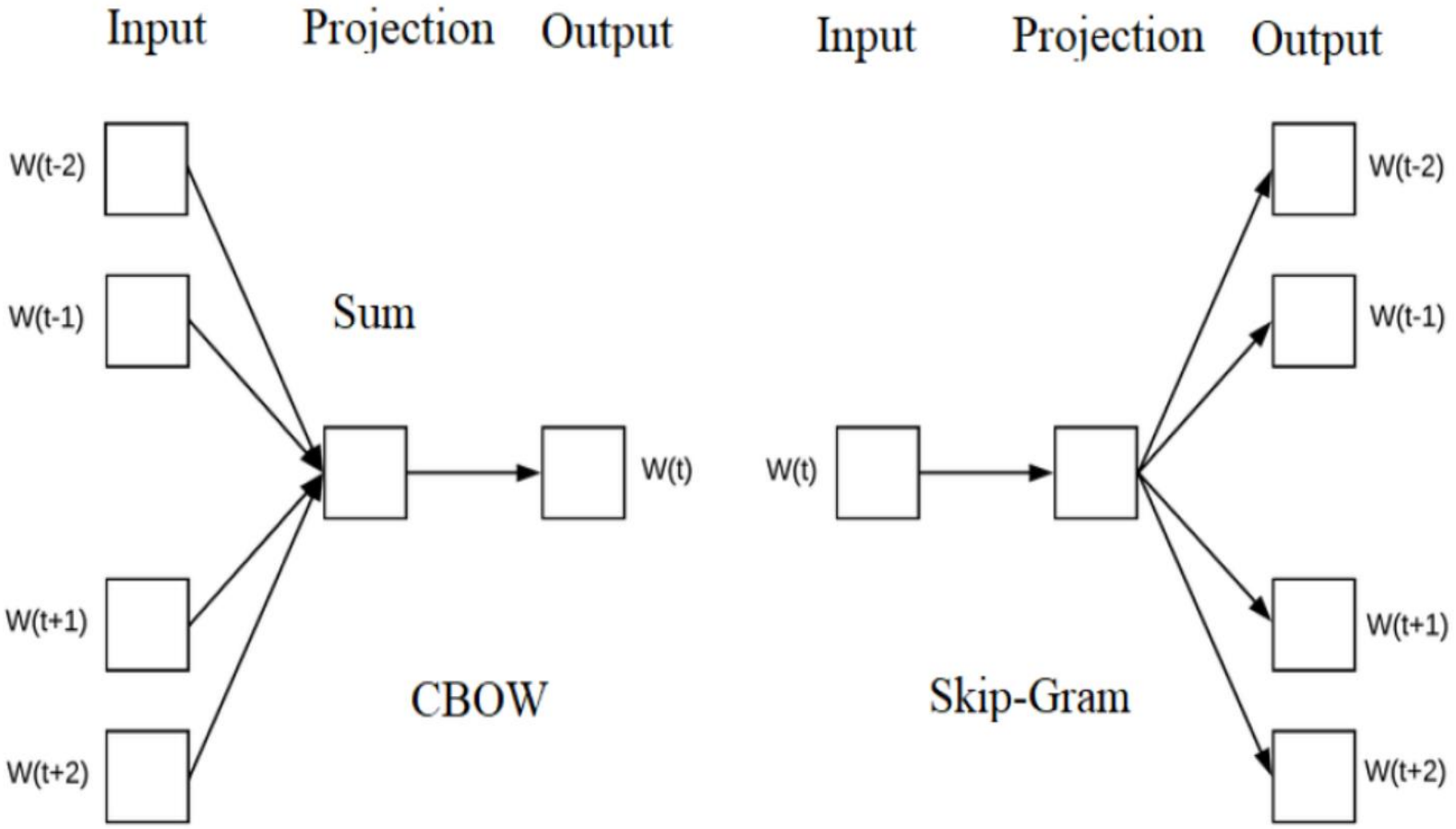



Number of neurons: dimension of the embeddings

https://towardsdatascience.com/implementing-word2vec-in-pytorch-from-the-ground-up-c7fe5bf99889/

# *To recap!*

# CBOW Versus Skip-Gram Model




Architectures of CBOW and Skip-Gram embedding models [Mikolov *et al.* 2013].

# Word2vec Embeddings

- **Skip-Gram Model** is trained to predict the neighbouring words (context) given the middle word

Formally, given a sequence of training words $\{w_{t-c}, ..., w_{t+c}\}$ that represents the context on the left and the context on the right of the word $w_t$, the skip-gram model is trained to maximize the average log probability, as defined in Equation 1.3, to predict the context of the word $w_t$.

$$\frac{1}{|C|}\sum_{t=1}^{|C|}\sum_{j=t=c, j\neq t}^{t+c} \log Probability(w_j|w_t) \tag{1.3}$$

where $|C|$ is the number of words in the corpus and $c$ is the size of the dynamic context of $w_t$.

https://arxiv.org/pdf/1301.3781 [Mikolov et al., 2013]

# Word2vec Embeddings

- **CBOW (Continuous Bag-of-words) Model** trained with the objective of predicting the middle word based on surrounding context words. The context consists of a few words before and after the current (middle) word.

> Given a target word $w_t$ and its context $\{w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}\}$, the CBOW model, based on a simple neural architecture, is trained to maximize the log probability of predicting the word $w_t$ given its context, formally defined in Equation 1.2:
>
> $$\frac{1}{|C|}\sum_{t=1}^{|C|} \log Probability(w_t|w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}) \quad (1.2)$$
>
> where $|C|$ is the number of words in the corpus and $c$ is the size of the dynamic context of $w_t$.

- This architecture is called a bag-of-words model as the order of words in the context is not important.

https://arxiv.org/pdf/1301.3781 [Mikolov et al., 2013]

# Common Neural Word Embeddings Models

- **Word2vec (Mikolov et al)**
- https://code.google.com/archive/p/word2vec/
- https://stackabuse.com/implementing-word2vec-with-gensim-library-in-python/
- https://www.kaggle.com/code/pierremegret/gensim-word2vec-tutorial

- **GloVe (Pennington, Socher, Manning)**
- http://nlp.stanford.edu/projects/glove/

- **FastText Model:** https://fasttext.cc/

# Neural Word Embeddings

Neural Word Embeddings (**word2vec**, **gloVe**, **FastText**) provide ***static embeddings*** where each word is mapped to a fixed embedding.

Sometimes, they are called (Context-Free) because they generate static embedding vectors

# Sentence Embeddings: Motivation

- Sentence 1: *”The cat sat on the mat"*
- Sentence 2: *”The mat sat on the cat”*

- By applying ***BOW*** and ***word embeddings*** representations, these two sentences will have identical meaning while their meanings are entirely different.

- Solution: Use Sentence Embedding Models (Contextual Embeddings)

# Sentence Embeddings (Contextual Embeddings)

- **Sentence embedding methods** encode sentences into dense vectors, represented in a low dimensional vector space,
- that accurately capture the **semantic** and **syntactic** relationships between these sentences constituents.
- The main goal is to map sentences with similar meanings to similar vectors, and simultaneously sentences with different meanings are mapped to different vectors.

# Contextualized Embedding Model: *ELMo*

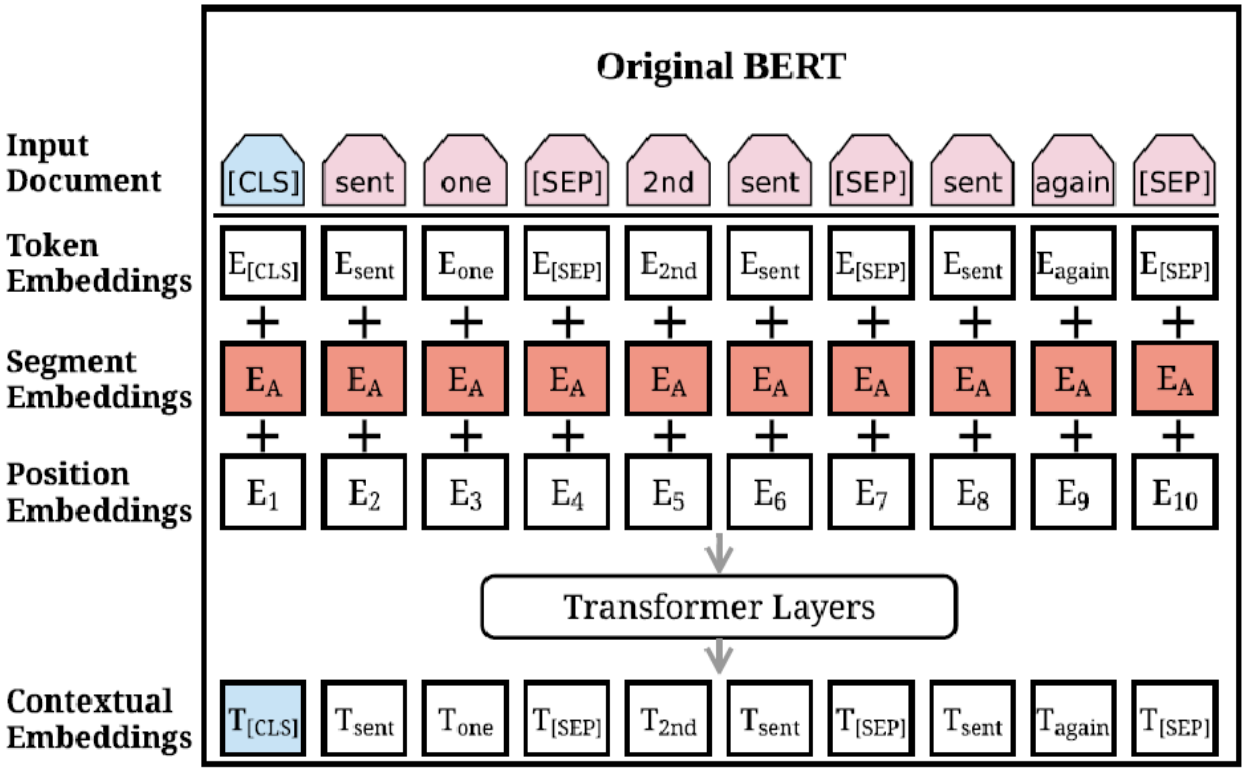
- **ELMo** provides contextualized word embeddings that vary depending on the context, improving understanding in tasks like question answering, sentiment analysis, and named entity recognition

- **Architecture**:
  - Uses **Bidirectional LSTM layers** to capture context from both directions.
  - **Character-based CNNs** create embeddings that capture word morphology.
  - **Layer-wise embedding** combination provides contextualized word representations.

- **Key Feature**:
  - Dynamic, context-dependent embeddings that adapt to different meanings of words based on usage.

***Deep contextualized word representations Peters et al., NAACL 2018***

# Sentence Embedding Model: *BERT*




- The Bidirectional Encoder Representations from Transformers (BERT) model is based on a multi-layer bidirectional transformer encoder with attention mechanisms.

- BERT is trained ***simultaneously*** on two unsupervised tasks : ***the masked language modelling*** and ***the next sentence prediction.***

https://arxiv.org/abs/1810.04805

- BERT is trained using two versions:

  $BERT_{BASE}$ : L = 12, H = 768, A = 12, Number of parameters = 110M

  $BERT_{LARGE}$ : L = 24, H = 1024, A = 16, Number of parameters = 340M

# Sentence Embeddings Models

| Models | Learning pardigms | Training method | Word Vectors | Embedding size | Datasets |
|---|---|---|---|---|---|
| uSIF | Non-parameterized | Unsupervised | ParaNMT | 300 | – – – |
| GEM_GloVe | Non-parameterized | Unsupervised | GloVe | 300 | – – – |
| GEM_LexVec | Non-parameterized | Unsupervised | LexVec | 300 | – – – |
| ELMo | Parameterized | Supervised | – – – | 1024 | One Billion Word Benchmark |
| NNLM | Parameterized | Supervised | – – – | 128 | English Google news |
| USE-DAN | Parameterized | Both | Skip-gram | 512 | SNLI, Wikipedia, news, web pages ... |
| USE-Transformer | Parameterized | Both | – – – | 512 | SNLI, Wikipedia, news, web pages ... |
| InferSent_GloVE | Parameterized | Supervised | GloVe | 4090 | SNLI dataset |
| InferSent_FastText | Parameterized | Supervised | FastText | 4090 | SNLI dataset |
| $\text{BERT}_{BASE}$ | Parameterized | Unsupervised | – – – | 768 | BookCorpus and Wikipedia |
| $\text{BERT}_{LARGE}$ | Parameterized | Unsupervised | – – – | 1024 | BookCorpus and Wikipedia |
| $\text{SBERT}_{BASE}$ | Parameterized | both | – – – | 768 | BookCorpus, Wikipedia, NLI |
| $\text{SBERT}_{BASE}$ | Parameterized | both | – – – | 1024 | BookCorpus, Wikipedia, NLI |
| Skip-Thought | Parameterized | Unsupervised | CBOW | 2400 | BookCorpus |

https://www.sciencedirect.com/science/article/pii/S0957417420308952

# **Dense** vectors: Summary

- Dense vector representations for words
  - dimensionality typically much lower than in BOW
  - **learned** based on context of words
- semantics
  - similar words have similar vectors in the embedding space => This captures the semantic relationships between words, enabling us to identify synonyms, antonyms, and other linguistic associations.
  - directions in the vector space map to semantic relationships: For example, vector arithmetic operations like "king - man + woman" might lead to a vector close to "queen."
- **context-free** and **contextual** embeddings: Context-free embeddings, such as Word2Vec and GloVe, represent words regardless of their context. On the other hand, contextual embeddings, such as BERT and GPT, consider the context in which words appear, leading to richer representations.
- either learn from data or use a pre-trained embedding- Pre-trained embeddings save time and resources and offer excellent generalization.

# *Topic Modeling Methods*

# Topic Modeling Methods

**Motivation**:

- Suppose you're given a massive corpora and asked to carry out the following tasks
  - **Organize** the documents **into thematic categories**
  - **Describe** the evolution of those categories **over time**
  - Enable a domain expert **to analyze and understand** the content
  - Find **relationships** between the categories
  - Understand how **authorship** influences the content

# Topic Modeling Methods

- Topic modeling methods (usually unsupervised) aim to discover latent or hidden structures in a corpus by identifying the main 'topics' that occur across all documents to extract relevant information.

- Common Topic Modeling methods are:
  - **Latent Semantic Analysis (LSA)**. https://doi.org/10.1037/0033-295X.104.2.211
  - **Latent Dirichlet Allocation (LDA)** (https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf)
  - **BERTopic Model** (https://arxiv.org/abs/2203.05794)

# Idea for Presentations (+2 points in Final Written Exam)

- **GloVe Word Embedding Model (Filip) +**
- **FastText Word Embedding Model (Olexander)**
- **ELMo Contextualized Embedding Model (Federico)**
- **BERT Sentence Embedding Model (Georgi)**
- **Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) (Kim)**
- **TopicBERT Method (Souhail)**

# Idea for Presentations (+2 points in Final Written Exam)

- **SimCSE: Contrastive Learning for Representation Learning(Adelaide)**
- **Transfer Learning Across Domains (Martin)**
- **Knowledge Distillation (Ayoub)**
- **Active Learning for Data-Efficient Training (Hedi)**
- **Catastrophic Forgetting in Neural Networks ()**
- **Parameter-Efficient Fine-Tuning (e.g. LoRA) (Giorgos)**
- **Curriculum Learning: Teaching Models Like Humans ()**
- **Federated Learning ()**
- **Reinforcement learning from human feedback (RLHF) ()**

# Reading

- Please refer to this chapter: Chapter 5
- Very Important: http://d2l.ai/chapter_natural-language-processing-pretraining/index.html

# Next Class

- **Lecture 9: RNNs and Transformer Architectures**