Advanced Text Representations

Word Embeddings

Deep Learning for NLP

CiTIUS Centro Singular de Investigación en Tecnoloxías Intelixentes

USC Universidade de Santiago de Compostela

CAMPUS VIDA

Tecnologías de Gestión de Información No Estructurada

Prof. Dr. David E. Losada

Máster Interuniversitario en Tecnologías de Análisis de Datos Masivos: Big Data

Advanced Text Representations

BIBLIOGRAPHIC REFERENCES:

Almeida, F., Xexéo, G. : Word Embeddings: A Survey. Arxiv, (2019).

Wang, S., Zhou, W. & Jiang, C. A survey of word embeddings based on deep learning. Computing 102, 717–740 (2020).

Ali, Z. A simple Word2vec tutorial, (2019).

Koehrsen, W. Neural Network Embeddings Explained, (2018).

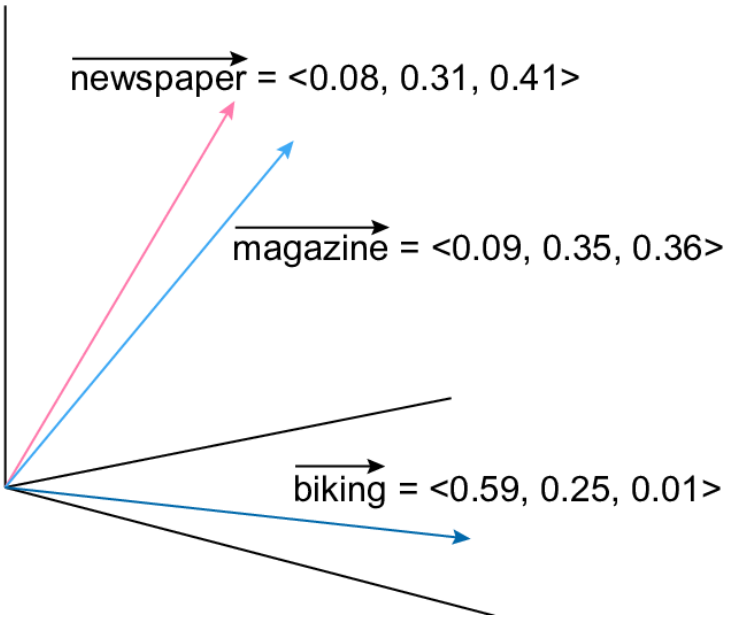
Park, J. Word. Embedding in NLP: One-Hot Encoding and Skip-Gram Neural Network, (2020).

McCormick, C., Ryan, N. BERT Word Embeddings Tutorial, (2019).

Word Embeddings

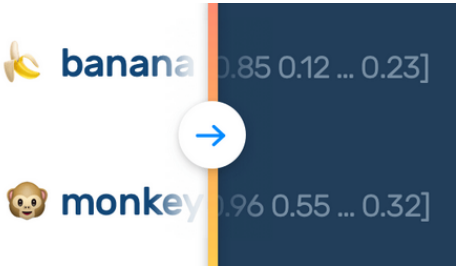
fixed-length, dense, distributed representations of words

based on the **distributional hypothesis**



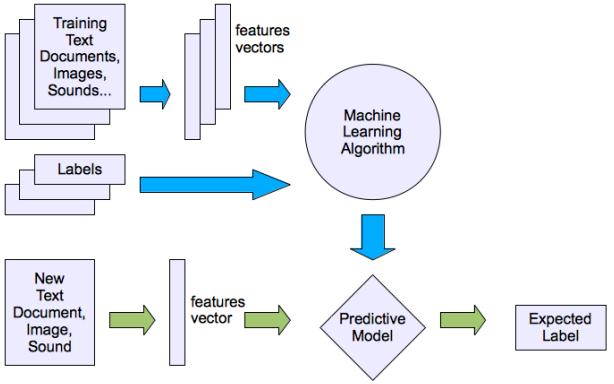
encode surprisingly **good syntactic and semantic information**.
capture lexical semantics in numerical form to handle **abstract semantic concept of words**

word embeddings



powerful in many **downstream NLP tasks** (Text Classification, Knowledge Mining, Question-Answering, Named-Entity Recognition etc.)

can be used as **standalone features** in many NLP tasks



classical vectorial representation

vector-space model (Salton 1975)

encodes **docs as vectors**

doc: t-dimensional vector

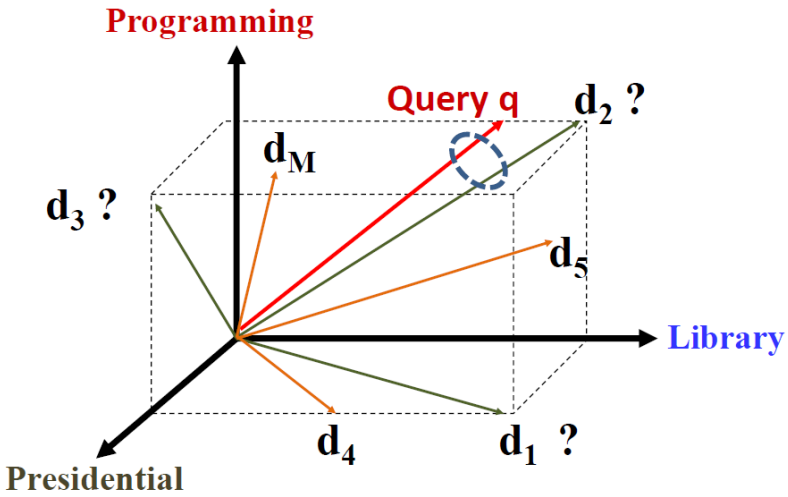
each element in the vector: represents a **distinct term** (with binary or tf/idf weights)

similarity between doc vectors

(e.g. inner product): **clustering** docs

similarity between query vector

and doc vector: **search** tasks



classical vectorial representation

BUT…follows

one-hot encoding

“I ate an apple and played the piano”

I	ate	an	apple	and	played	the	piano
1	2	3	4	5	6	7	8

Position of each word in the vocabulary

number of words
in the source vocabulary

number of dimensions

	1	2	3	4	5	6	7	8
I	1	0	0	0	0	0	0	0
ate	0	1	0	0	0	0	0	0
an	0	0	1	0	0	0	0	0
apple	0	0	0	1	0	0	0	0
and	0	0	0	0	1	0	0	0
played	0	0	0	0	0	1	0	0
the	0	0	0	0	0	0	1	0
piano	0	0	0	0	0	0	0	1

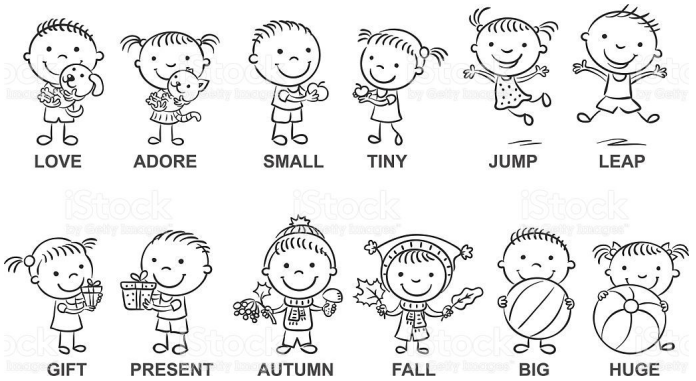
One-hot vector representation of each word in the vocabulary

problems with one-hot encoding

no semantic relatedness between words. each word is embedded in isolation, and every word contains a single one and N zeros where N is the number of dimensions.

the resulting set of vectors do not say much about one another.

“orange,” “banana,” and “watermelon,” we can see the similarity between those words, such as the fact that they are types of fruit, or that they usually follow some form of the verb “eat.” we can easily form a mental map or cluster where these words exist close to each other. but with one-hot vectors, all words are equal distance apart!



data-sparse problems. curse of dimensionality, most of the matrix is taken up by zeros, so useful data becomes sparse. Imagine we have a vocabulary of 50,000. (There are roughly a million words in English language.) each word is represented with 49,999 zeros and a single one. not computationally efficient.

distributional hypothesis

words with similar contexts (other words) have the same meaning (or contextually similar words have similar semantics)

What is tezgüino?

– A bottle of tezgüino is on the table.

– Everybody likes tezgüino.

– Tezgüino makes you drunk.

– We make tezgüino out of corn.

theoretical basis for **integrating semantics** into word representations

word embeddings

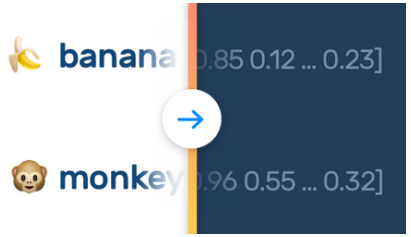
dense, distributed, fixed-length word vectors

built using **co-occurrence statistics** as per the distributional hypothesis

low dimensional space

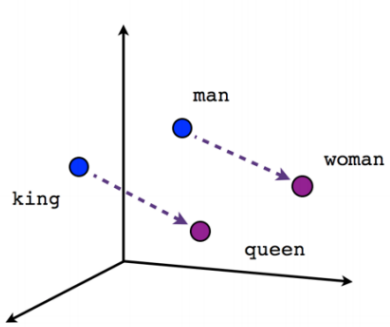
neural network embeddings useful: reduce the dimensionality of categorical variables and **meaningfully represent** categories in the transformed space

NN word embeddings

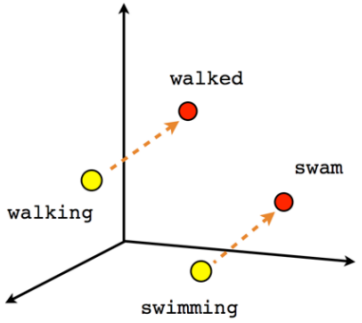


finding **nearest neighbors in the embedding space**

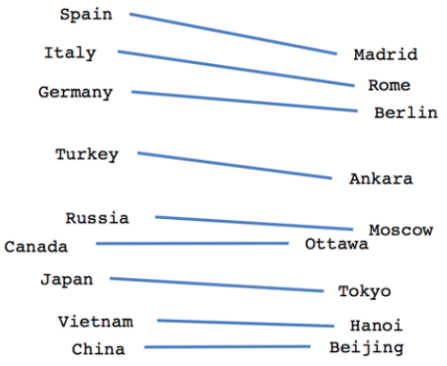
can be used to make recommendations based on user interests or cluster categories



Male-Female

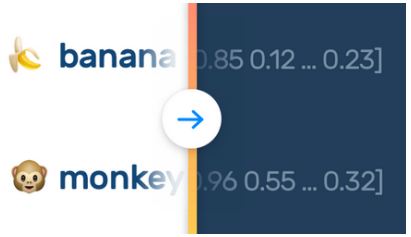


Verb tense

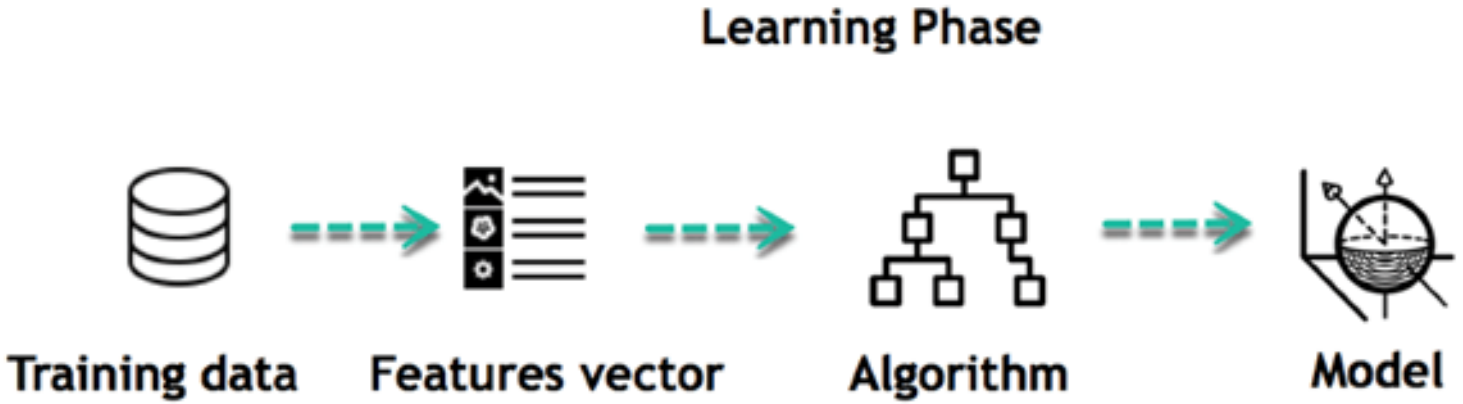


Country-Capital

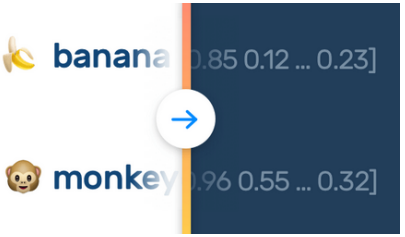
NN word embeddings



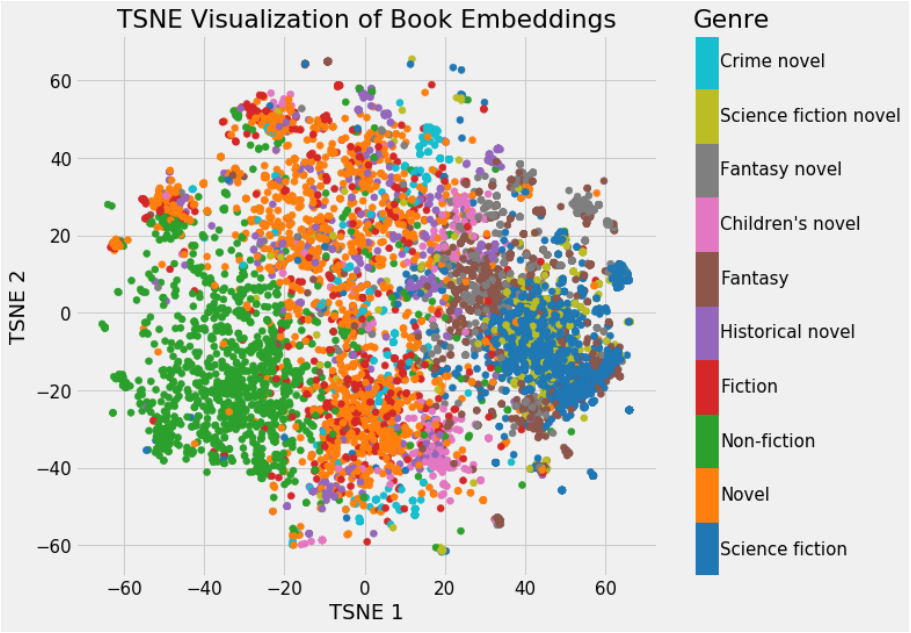
input to a machine learning model for a supervised task



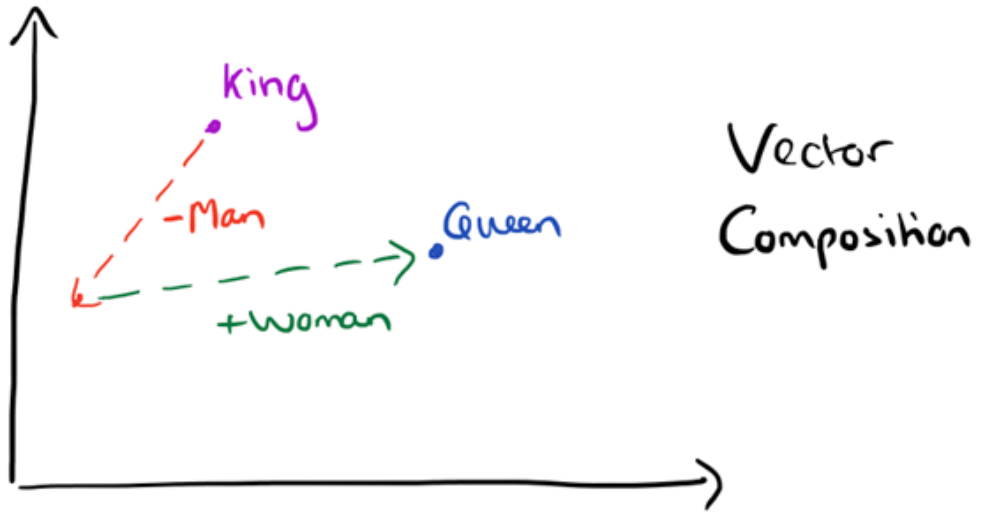
NN word embeddings



visualization of concepts and relations between categories



king + woman – man ≈ queen



how to obtain the embeddings?

Neural Network Language Models, Word-Context Matrices, …

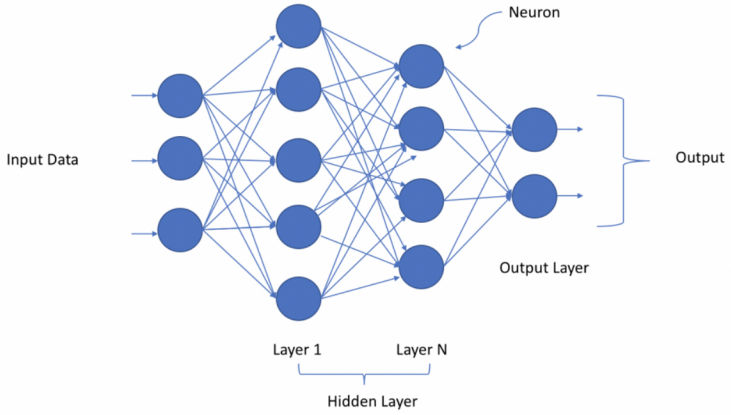
<u>Prediction-based models:</u>

derived from **Neural Network Language Models**

leverage language models

(e.g. predict next word given its context)

and "**local data**"



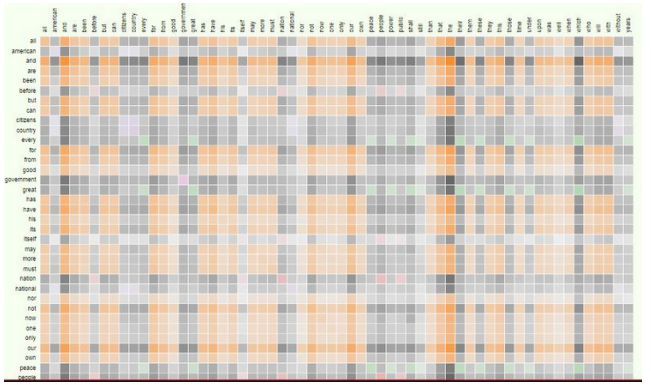
<u>Count-based models:</u>

matrix-based models.

based on global word-context

co-occurrence counts.

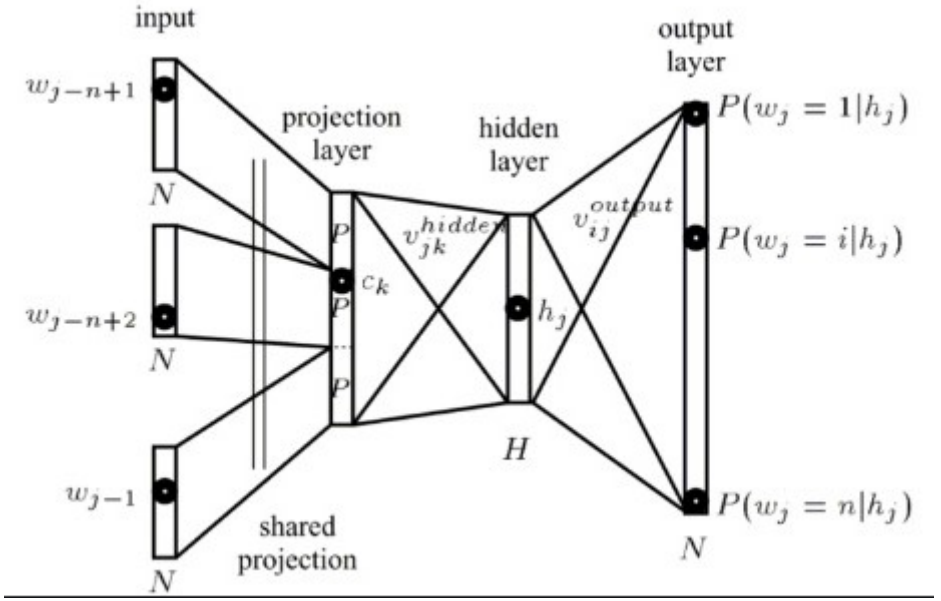
leverage "**global data**"



language modelling (NNs vs n-gram models)

Neural LMs better at modelling language but **long training times** were among the major factors that hindered the development of early NLMs

Since 2003, many **advances** in terms of **efficiency and performance** of NLMs. Improvement of hardware performance and breakthrough in optimization methods.

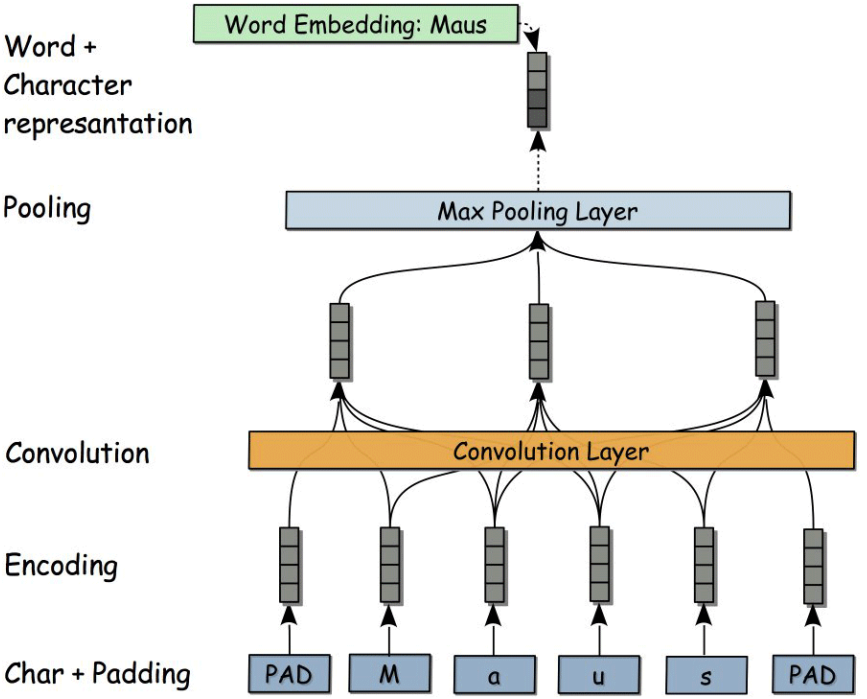


language modelling (NNs vs n-gram models)

now, **NNs** have become **mainstream** methods of word distribution representations

originally embeddings were treated as an interesting **by-product** of the main task (language modelling)

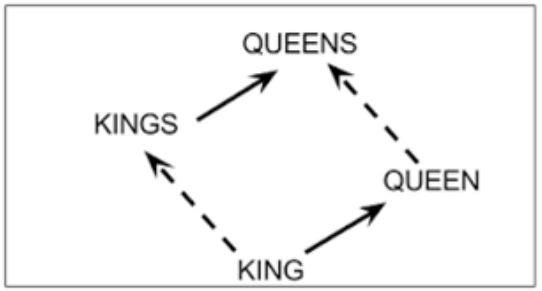
now many models are built with the specific intent of learning embeddings only

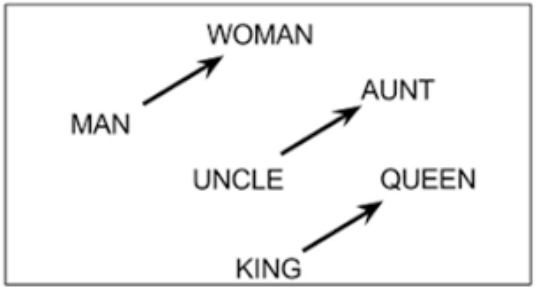


NN-based distributed representations

produce word embeddings that **model the context and relationship** between the target word and its context words through NNs

finds **not only syntactic** but also **semantic** regularities in the data



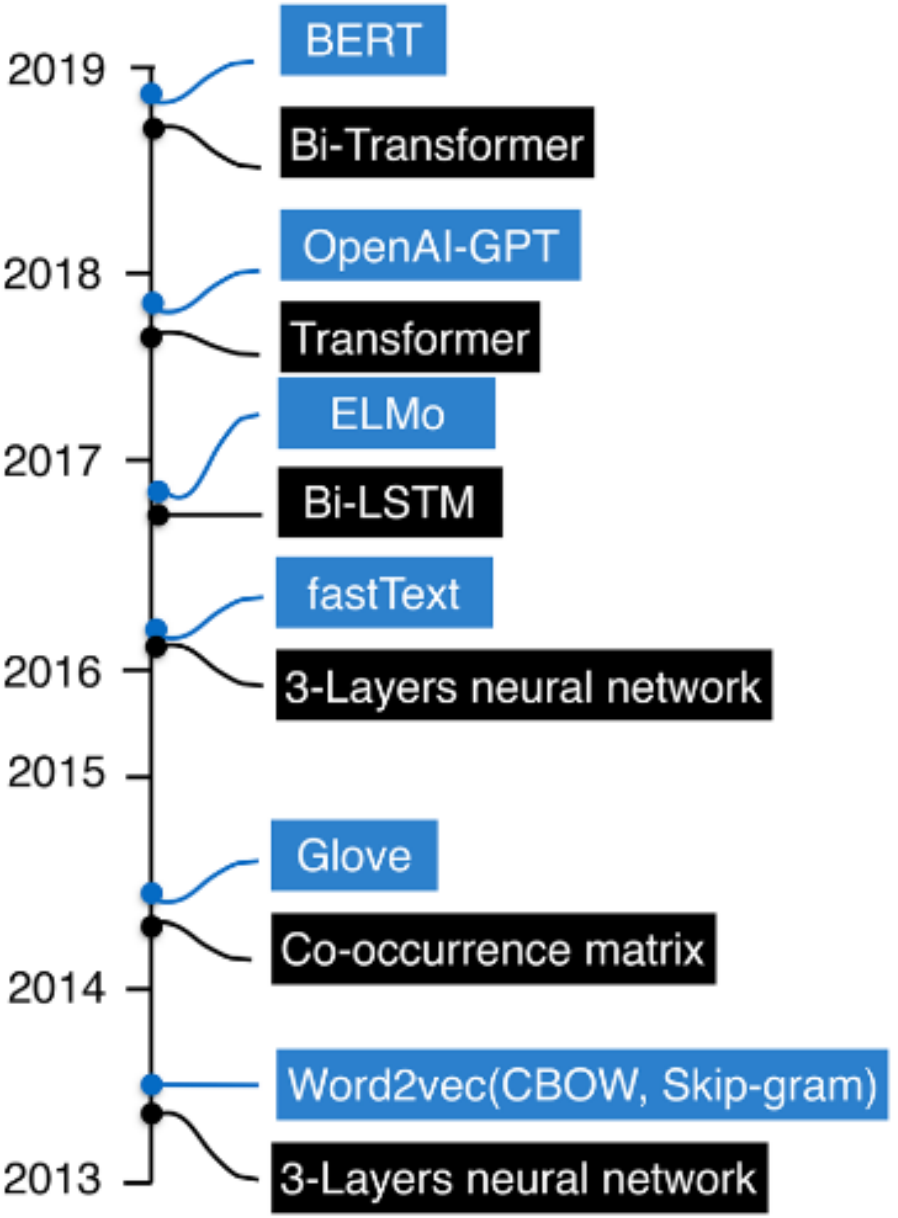


common relationships such as male-female, singular-plural correspond to arithmetical operations on word vectors

timeline

by Wang et al 2020,

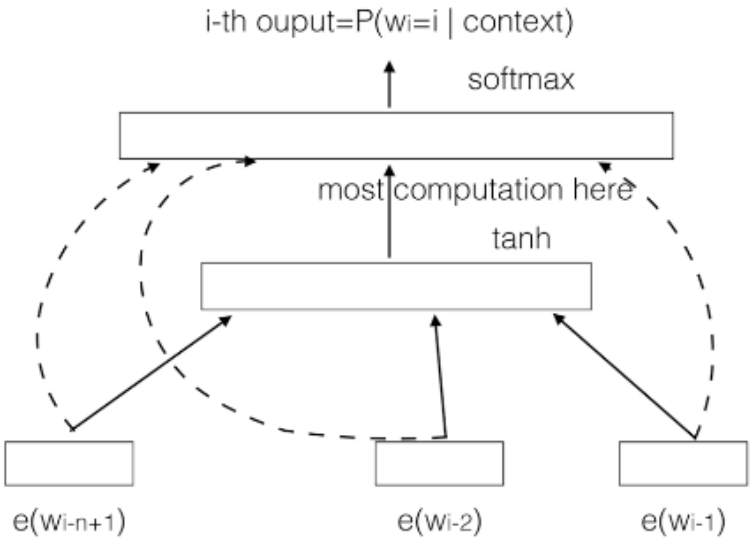
“A survey of word embeddings based on deep learning”



neural network language model (NNLM)

NNLM (Bengio et al 2003)

obtains word embeddings as the product while training language model. Similar to the traditional language model, NNLM **uses the previous n−1 words to predict the n th word**



NNLM uses a **3-layer network** structure while the input of the model is the sequential splicing of word embeddings of the sequence $w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1}$

neural network language model (NNLM)

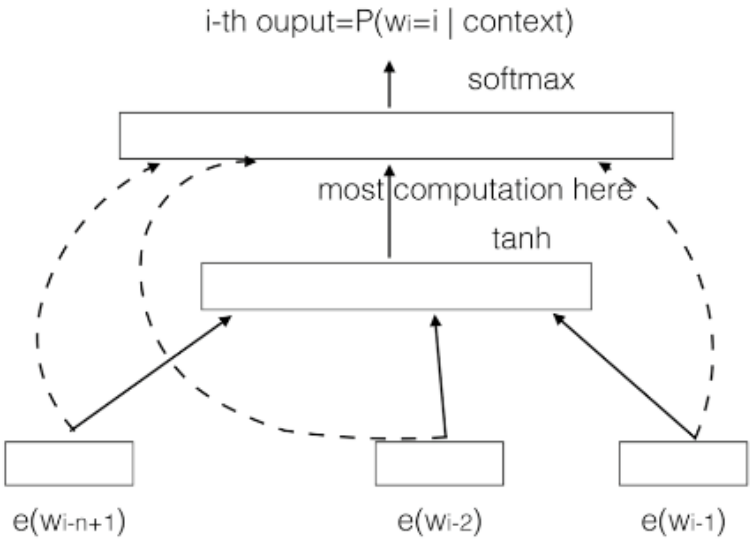
NNLM (Bengio et al 2003)

$e(w_{i-1})$: the embedding of word w_{i-1}

$$x = [e(w_{i-(n-1)}); \cdots; e(w_{i-2}); e(w_{i-1})]$$

$$h = tanh(b^{(1)} + Hx)$$

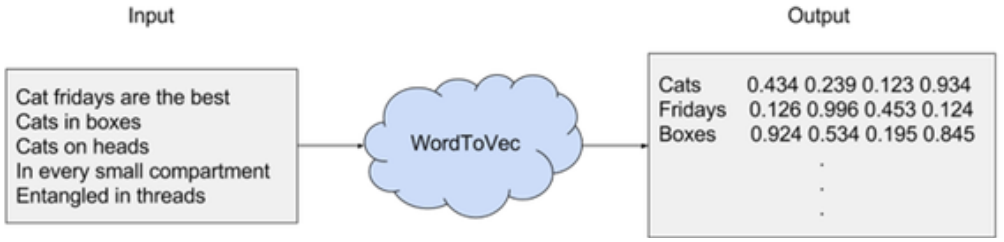
$$y = b^{(2)} + Wx + Uh$$



$b^{(1)}$ and $b^{(2)}$: bias terms in the model, y is the output, H is the weight matrix of the hidden layer, U is the weight matrix of the output layer, and W represents the direct-edge weight matrix from the input layer to the output layer.

Word2Vec (Mikolov et al 2013)

Word2Vec is an algorithm that accepts text corpus as an input and outputs a vector representation for each word



two models for learning embeddings:

Continuous bag-of-words (CBOW): trains a model that aims to predict the centre word based upon its context

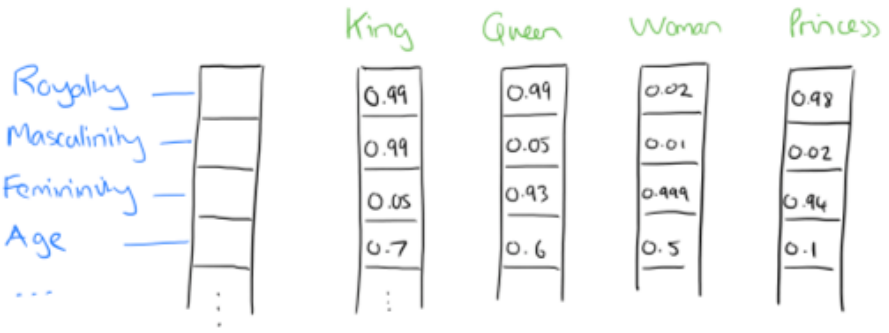
Skip-gram models (SG): the centre word is used to predict each word appearing in its context

Word2Vec (Mikolov et al 2013)

Distributed representation of words. Take a vector with several hundred dimensions. 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 the elements in the vector, and each element in the vector contributes to the definition of many words.

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:

Word2Vec (Mikolov et al 2013)

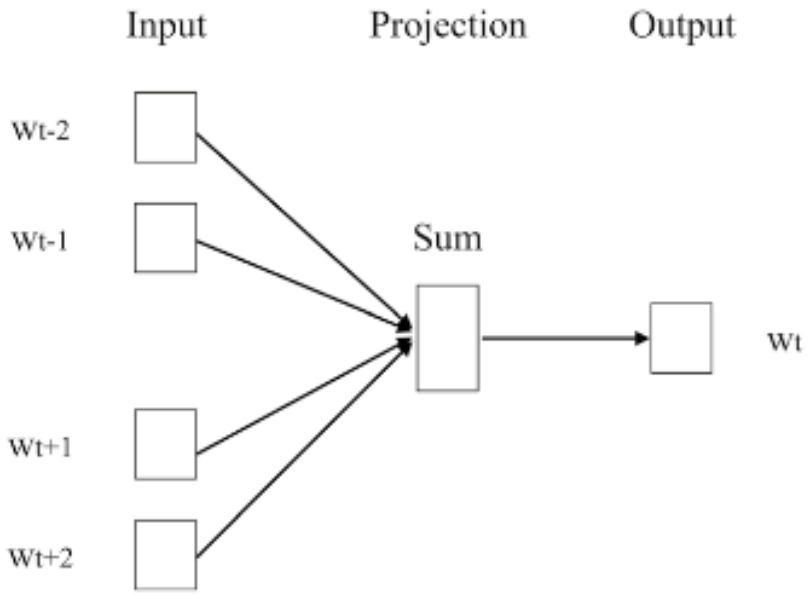


Vector: represents in some **abstract** way the '**meaning**' of a word.

From a large **corpus** it's possible to learn word vectors that are able to capture the relationships between words in a surprisingly expressive way.

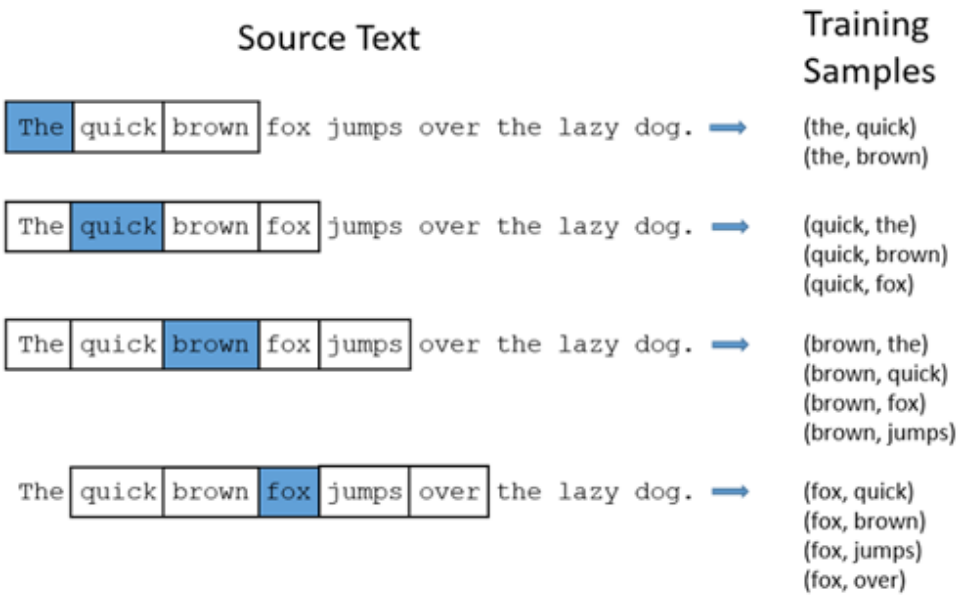
when the feature vector assigned to a word cannot be used to accurately predict that word's context, the components of the vector are **adjusted**. Each word's context in the corpus is the teacher sending error signals back to adjust the feature vector. The vectors of words judged similar by their context are nudged closer together by adjusting the numbers in the vector.

Word2Vec. CBOW

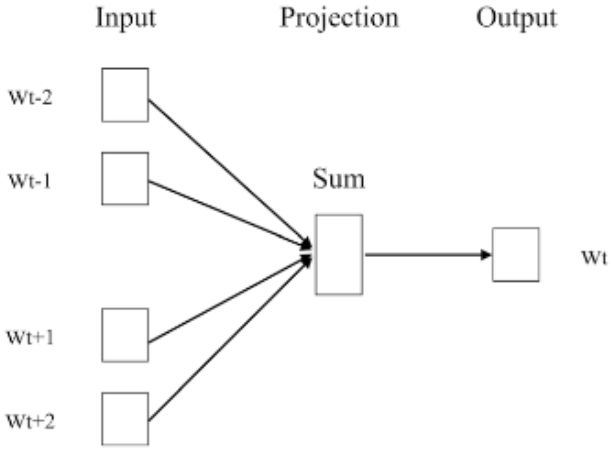


CBOW: find word representations that are useful to

predict the target word by the context words of it



Word2Vec. CBOW



the objective of the CBOW model is **to maximize**:

$$\frac{1}{N}\sum_{t=1}^{N}\sum_{-c\leq j\leq c, j\neq 0} logp\left(w_t|w_{t+j}\right)$$

c is the context window size, and the basic CBOW formulation defines $p(w_t|w_{t+j})$ using the **softmax function:**

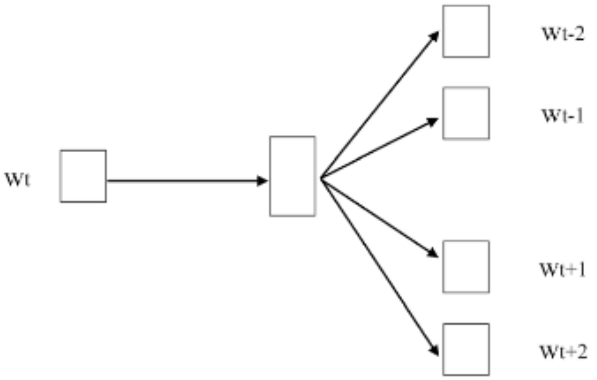
$$p(w_t|w_{t+j}) = \frac{exp\left(sim\left(w_t, w_{t+j}\right)\right)}{\sum_{w'\in V} exp\left(sim\left(w', w_{t+j}\right)\right)}$$

w is a word in the vocabulary V, and similarity is computed between the embeddings:

$$sim\left(w_t, w_{t+j}\right) = \overrightarrow{w_t} \cdot \overrightarrow{w_{t+j}}$$

Word2Vec. Skip-gram

predicts the **context words by the target word**:



maximizing the average log probability:

$$\frac{1}{N}\sum_{t=1}^{N}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

where c is the context window size,
and the model defines $p(w_{t+j} \mid w_t)$ using the soft-max function

$$p(w_{t+j}|w_t) = \frac{\exp\left(sim\left(w_{t+j}, w_t\right)\right)}{\sum_{w'\in V} exp\left(sim\left(w', w_t\right)\right)}.$$

FastText (Joulin et al 2016)

improvement over Word2Vec's Skip-Gram

learns not word embeddings but **n-gram embeddings** (which can be composed to form words)

fastText is a **text categorizer** which is based on CBOW
and open sourced by Facebook AI Research.

facebook
Artificial Intelligence Research

fastText greatly **shortens the training time** while maintaining the classification effect

existing word representation approaches assigning a distinct vector to each word while words are regarded as atomic tokens. This is a limitation, especially for languages which consist with sub-word level information

fastText uses **sub-word n-gram information**

better capture the internal semantics of words

FastText (Joulin et al 2016)

For a word **“subword”**, first add two characters “<” and “>” to express the boundaries of it: **<subword>** and then extract n-grams.

3-grams: **<su, sub, ubw, bwo, wor, ord, rd>**

In practice, extract **multiple n-grams** of words at the same time, such as 2/3/4/5-gram

each n-gram will train a vector, and the original word vector will be summed by the vector of all its corresponding n-grams

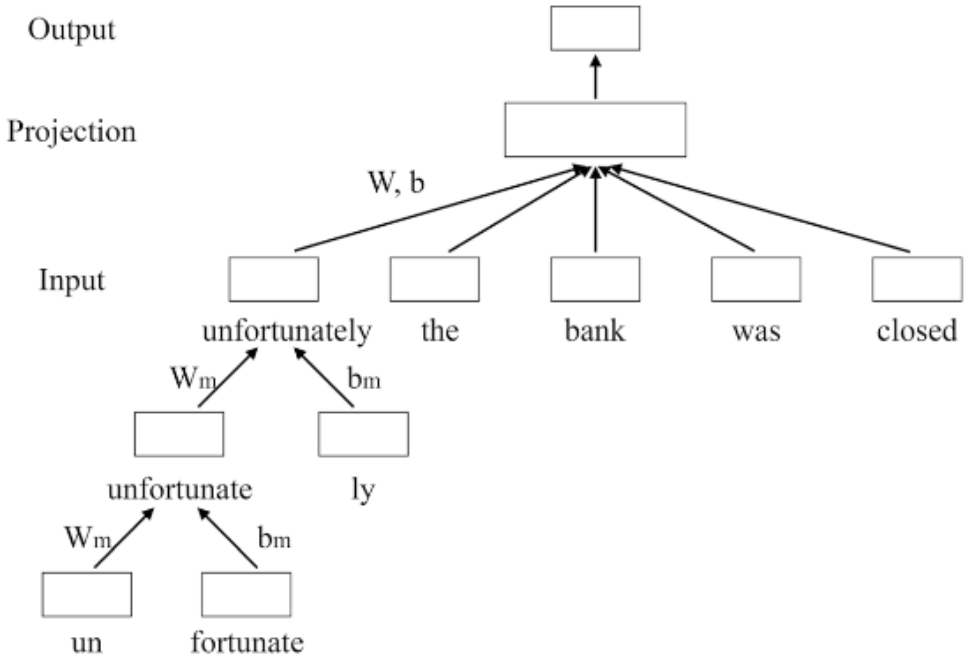
MorphoRNN (Luong et al 2013)

introduce **morphology** for learning word embeddings

prefix, root and suffix

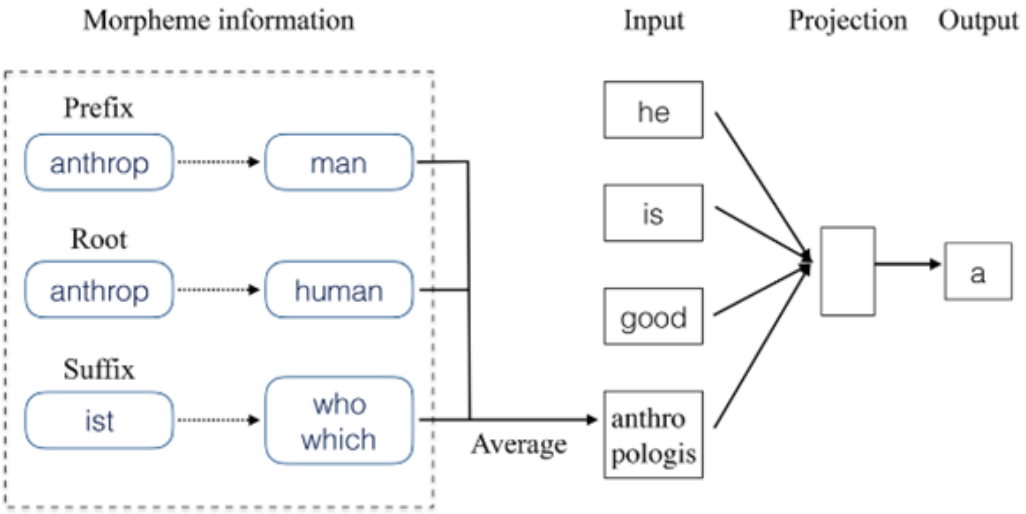
the **embedding of the parent word** is obtained by all morphemes

morpheme as the minimum unit of natural language, assigning a distinct vector to each morpheme



MWE (Xu & Liu 2017)

the idea of MWE is to represent the **meaning of prefix and suffix**

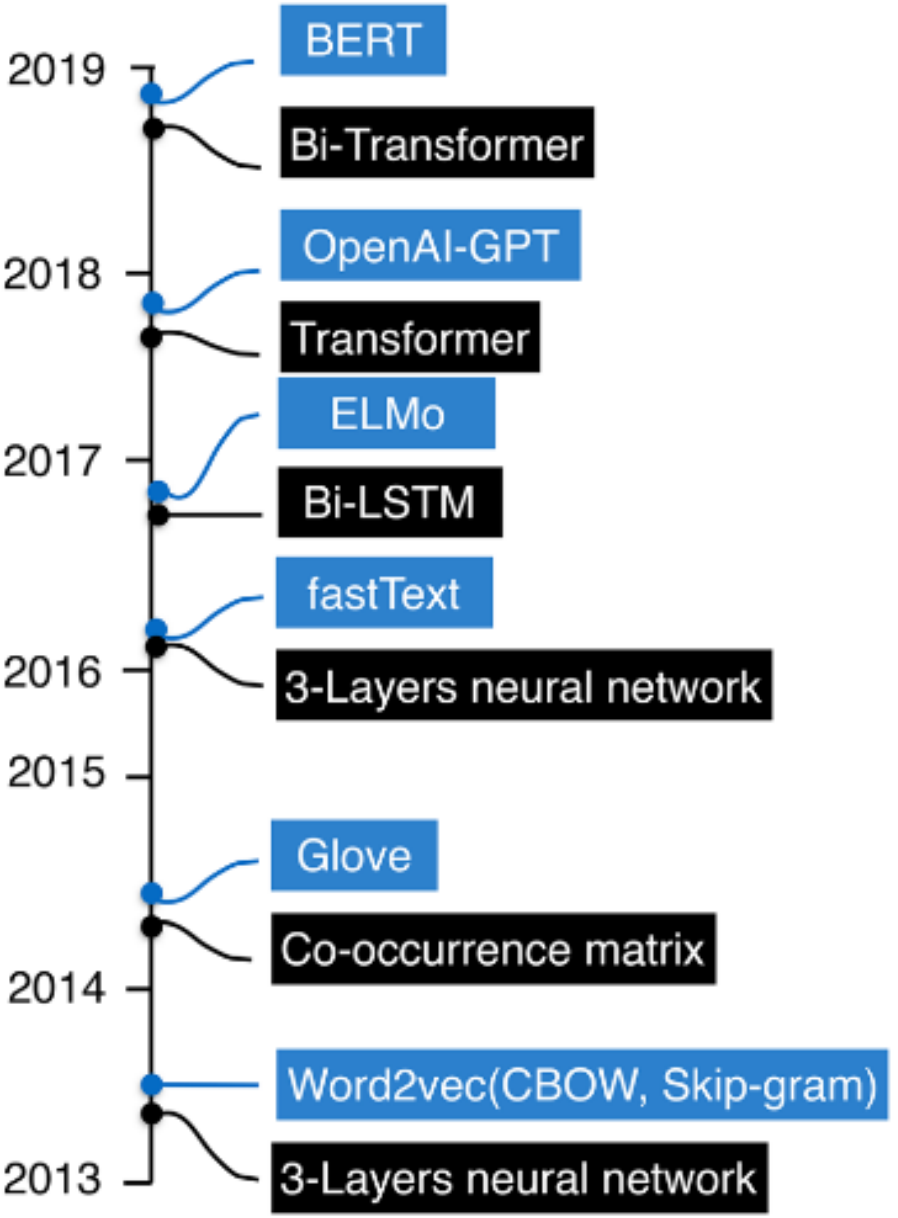


this model is built on the assumption that all **meanings of morphemes of token ti have equal contributions to ti**.

timeline

by Wang et al 2020,

“A survey of word embeddings based on deep learning”



embeddings based on context

context-specific representation problems

"The man was accused of robbing a bank"

"The man went fishing by the bank of the river"

for most downstream task in natural language processing (NLP), understanding the actual context is necessary.

ELMo (BiLSTM)

OpenAI-GPT (Transformer)

BERT (Transformer)

T5 (Transformer)

ELMo (Peters et al 2018)

ELMo: **Embeddings from Language Models**.

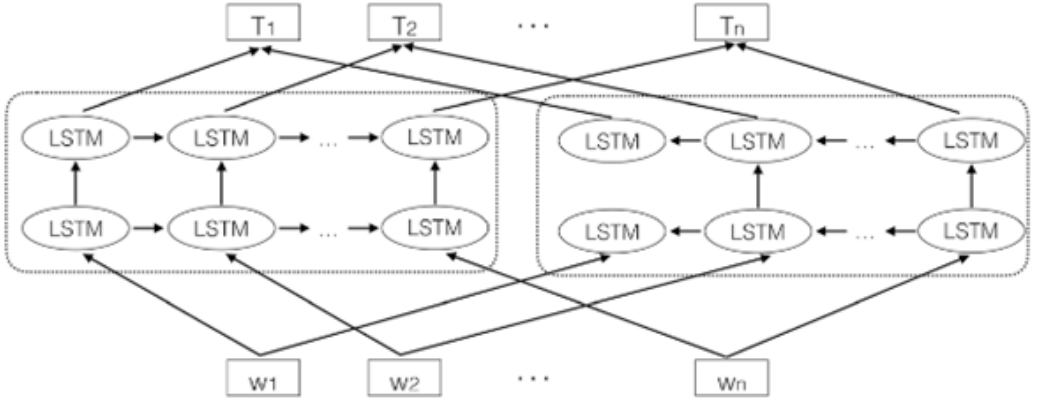
a good word representation model should take into account: the **complex nature of word usage in semantics** and **grammar** and these usages should change as the language environment changes

the characteristic of this algorithm is that the representation of each word is a function of the entire input statement

ELMo (Peters el at 2018)

train the **bidirectional LSTM model** on the large corpus with the **language model as the target**, and then use LSTM to generate the representation of the words.

two-way LSTM language model consisting of a forward and a backward language model. The objective function is to take the maximum likelihood of the language models in both directions.



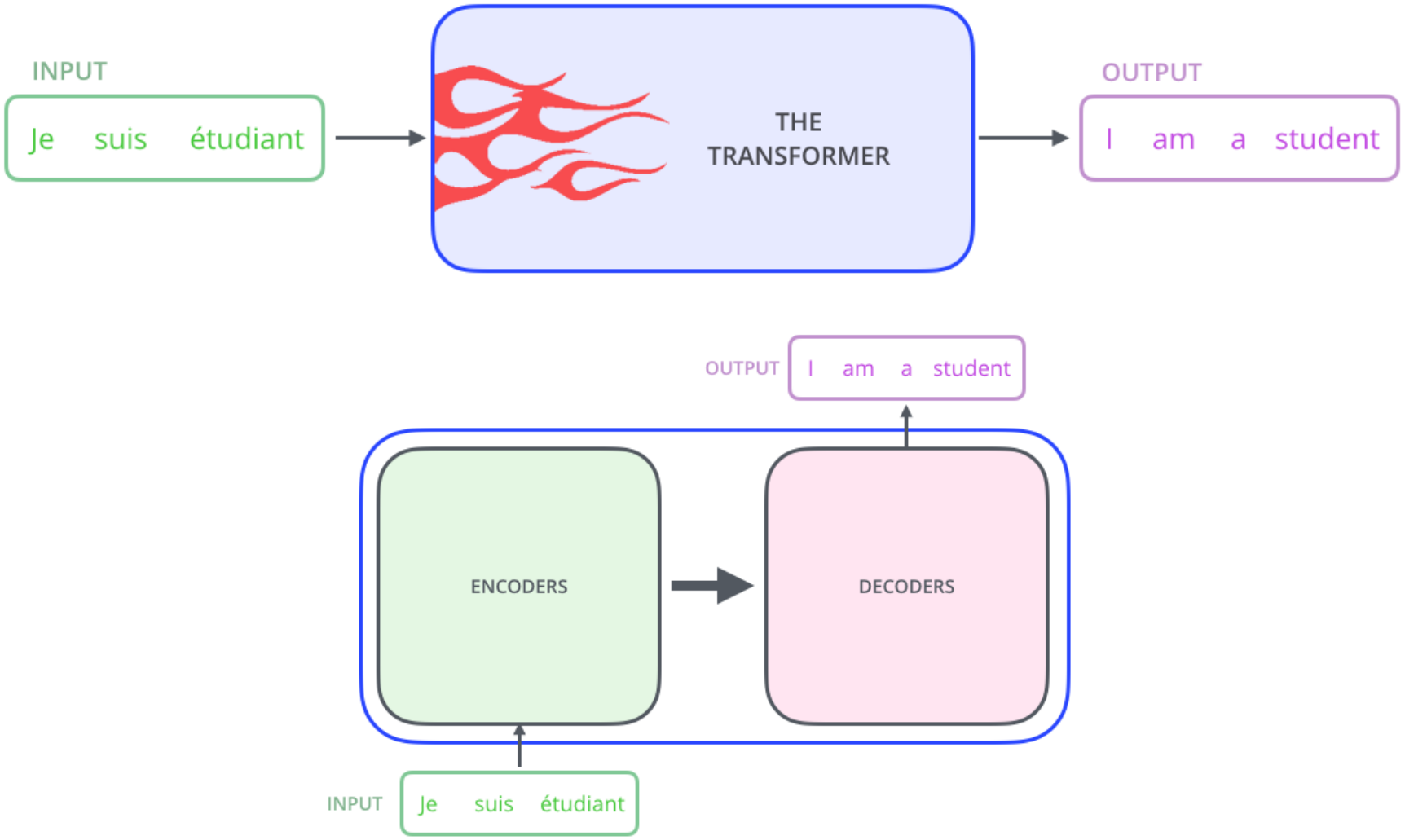
Transformers

Cutting-edge technology used in NLP

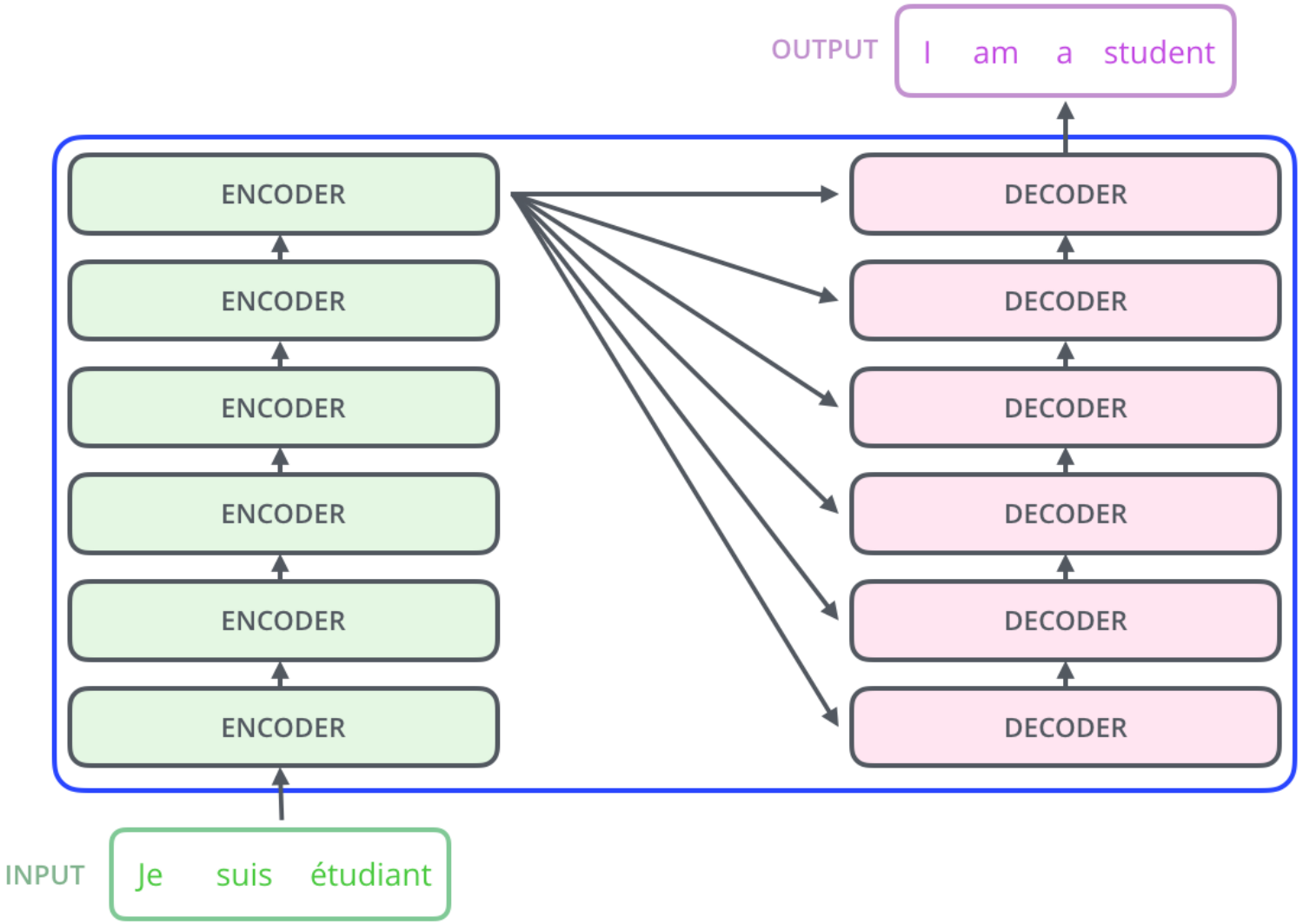
Based on **Attention –** a concept that helped improve the performance of neural machine translation applications

Attention allows the model to focus on **relevant** parts of the input sequence as needed

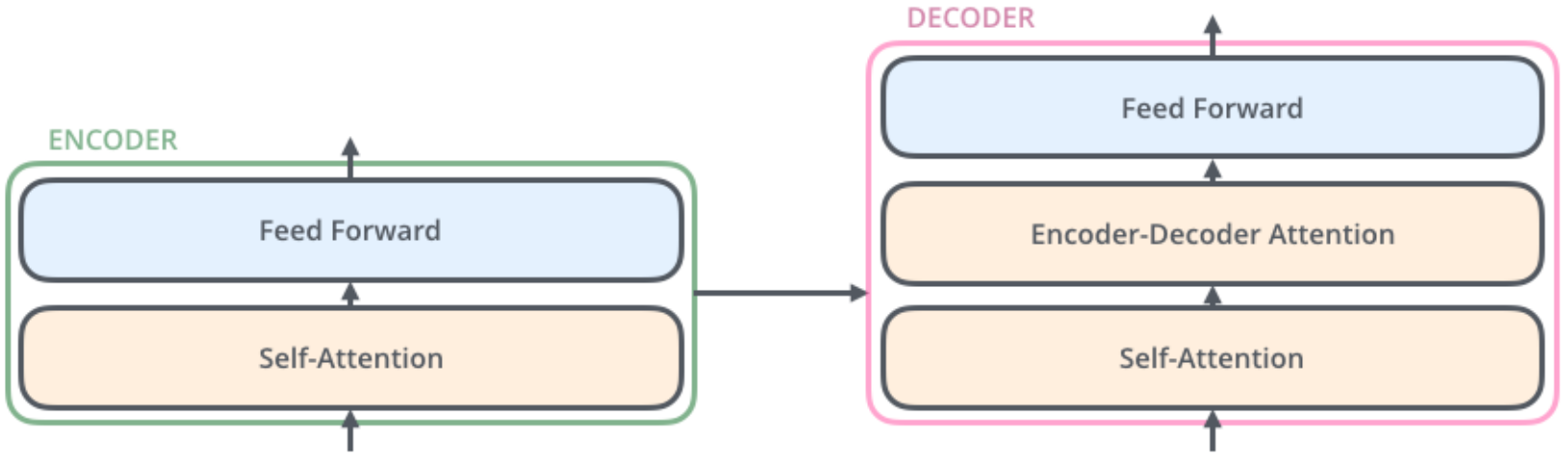
Transformers



Transformers



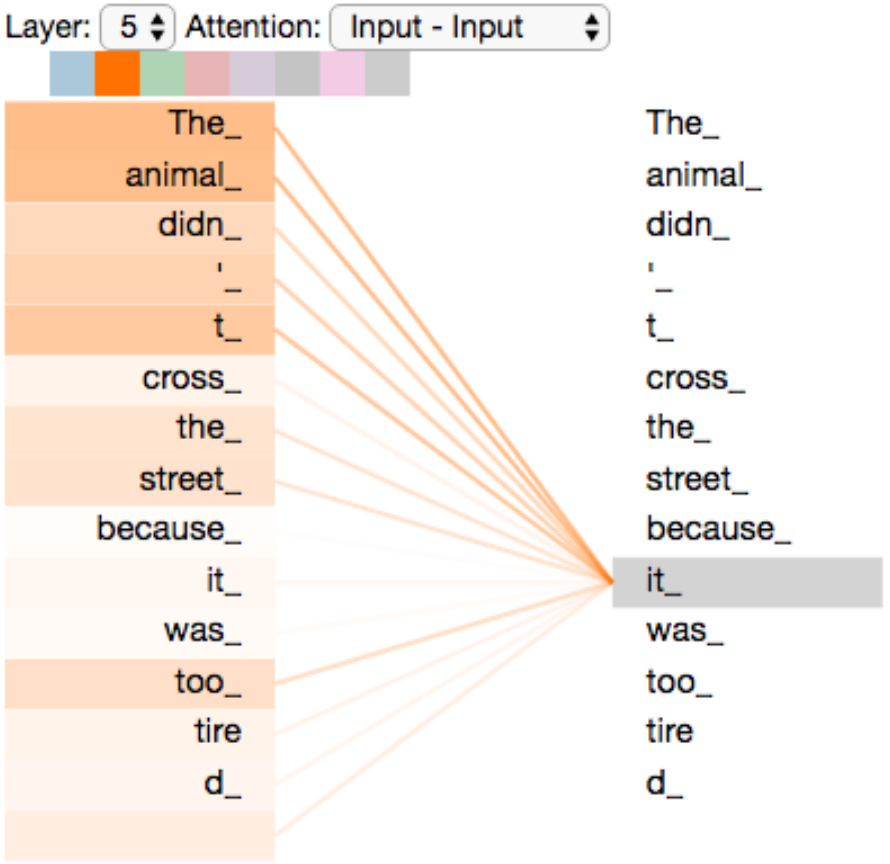
Transformers



Self-attention- helps the encoder look at the other words in the input sequence as it encodes a specific word (**context**)

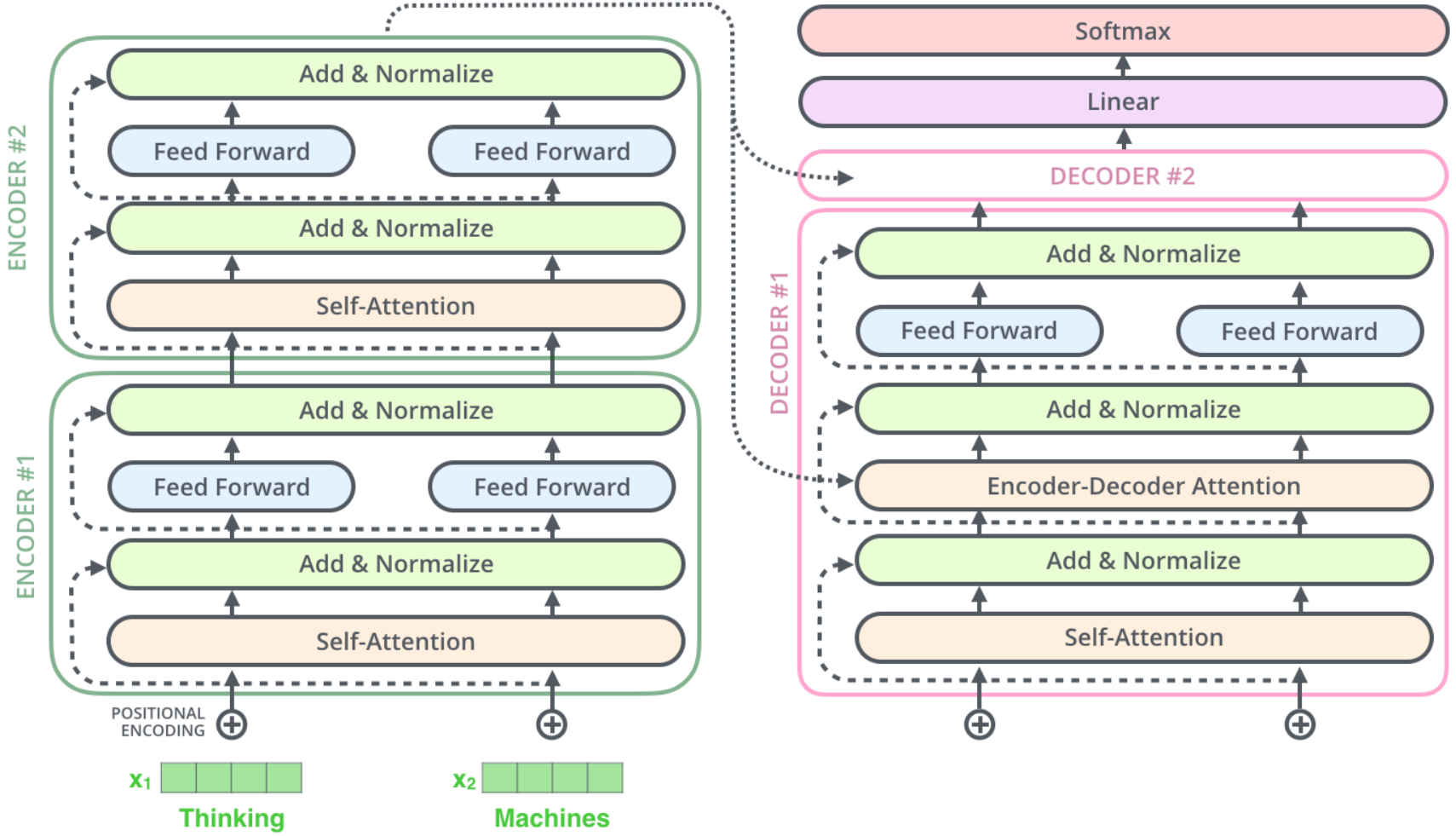
Encoder-Decoder Attention – helps the decoder focus on the relevant parts of the input sequence

Transformers context



Self-attention

Transformers



http://jalammar.github.io/illustrated-transformer/

Transformers: Pretraining & Fine-tuning

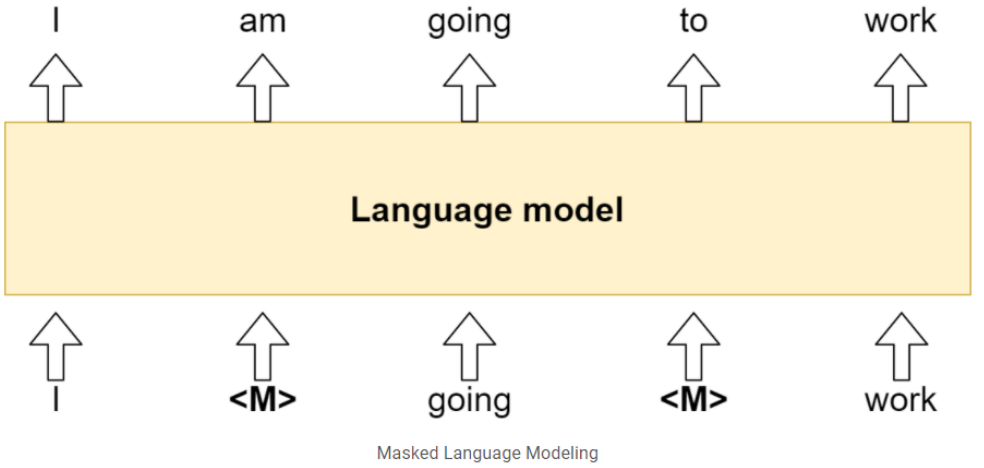
Pretraining: **unsupervised** language models on huge datasets. Masked language modeling

Fine tuning on downstream tasks (e.g. text classification)

GPT: only decoder

BERT: only encoder

T5: encoder-decoder

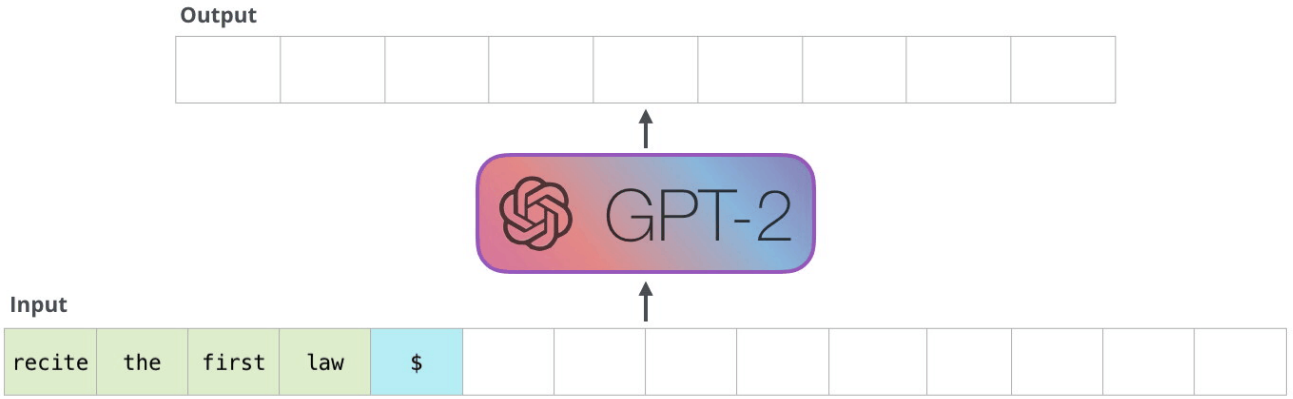


OpenAI-GPT (Ratford et al 2018)

OpenAI

Generative Pre-Training (GPT): pre-trained on next token prediction

Only decoder => it's a text generator, receives an input and tries to complete it based on previous knowledge.



OpenAI-GPT (Ratford et al 2018)

OpenAI

Fine-tune it for **any** downstream task =>

=> Training process

E.g.: Classifying texts as relevant to a query

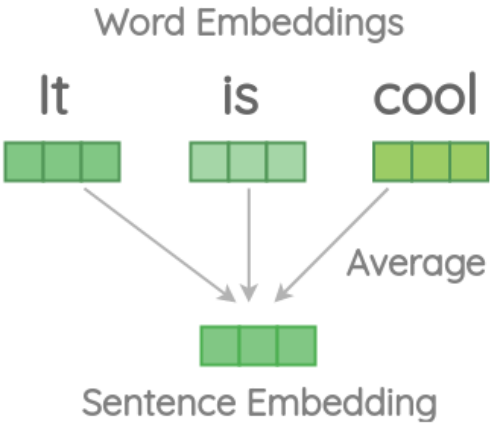
Training time: text+classify: relevant/not relevant

Prediction time: text+classify: …

BERT (Devlin et al 2018)

Pretraining: masked language model

Encoder only



BERT (Devlin et al 2018)

useful for **keyword/search expansion, semantic search, sentiment analysis** or **information retrieval**. For example, if you want to match customer questions or searches against already answered questions or well documented searches, these representations will help you accurately retrieve results matching the customer's intent and contextual meaning, even if there's no keyword or phrase overlap.

BERT fine tuning (Devlin et al 2018

Fine-tune it for **any** downstream task =>

=> Training process

E.g.: Classifying texts as relevant to a query

Training time: text: 1 (relevant)/ 0 (not relevant)

Prediction time: text

T5: sequence-to-sequence

Pretraining: masked language model

Encoder & decoder

Fine-tune it for **any** downstream task =>

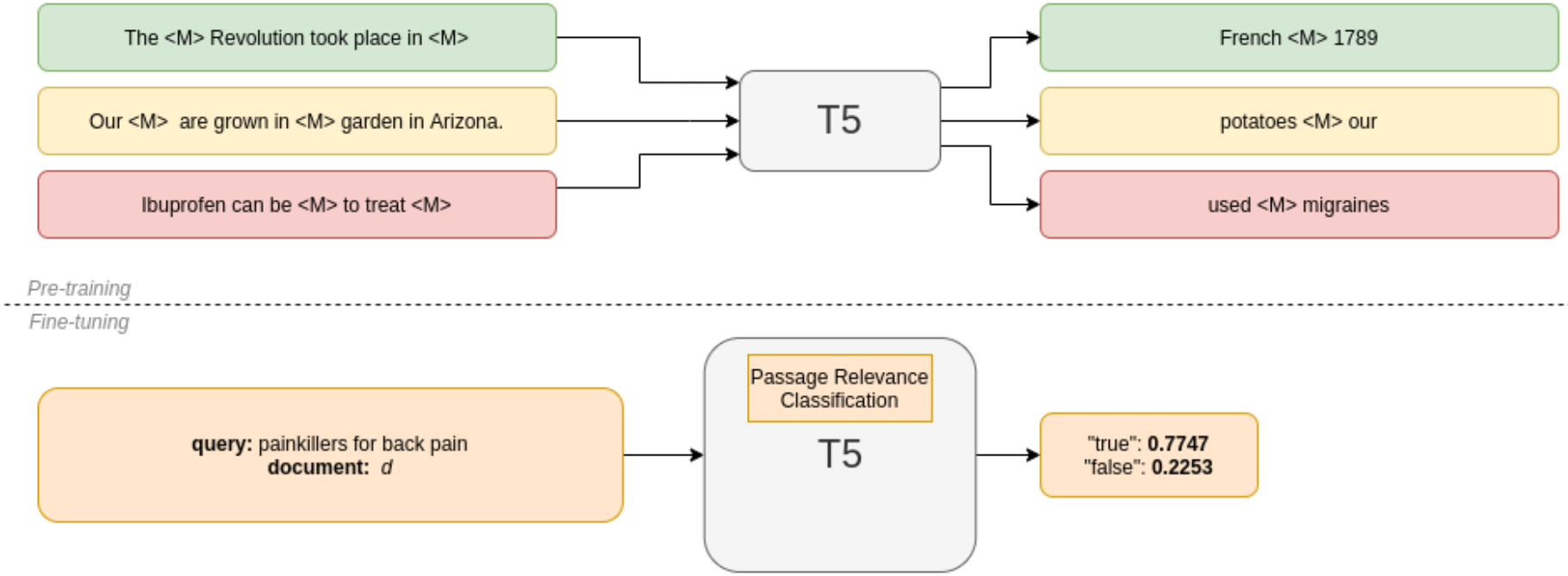
=> Training process

E.g.: Classifying texts as relevant to a query

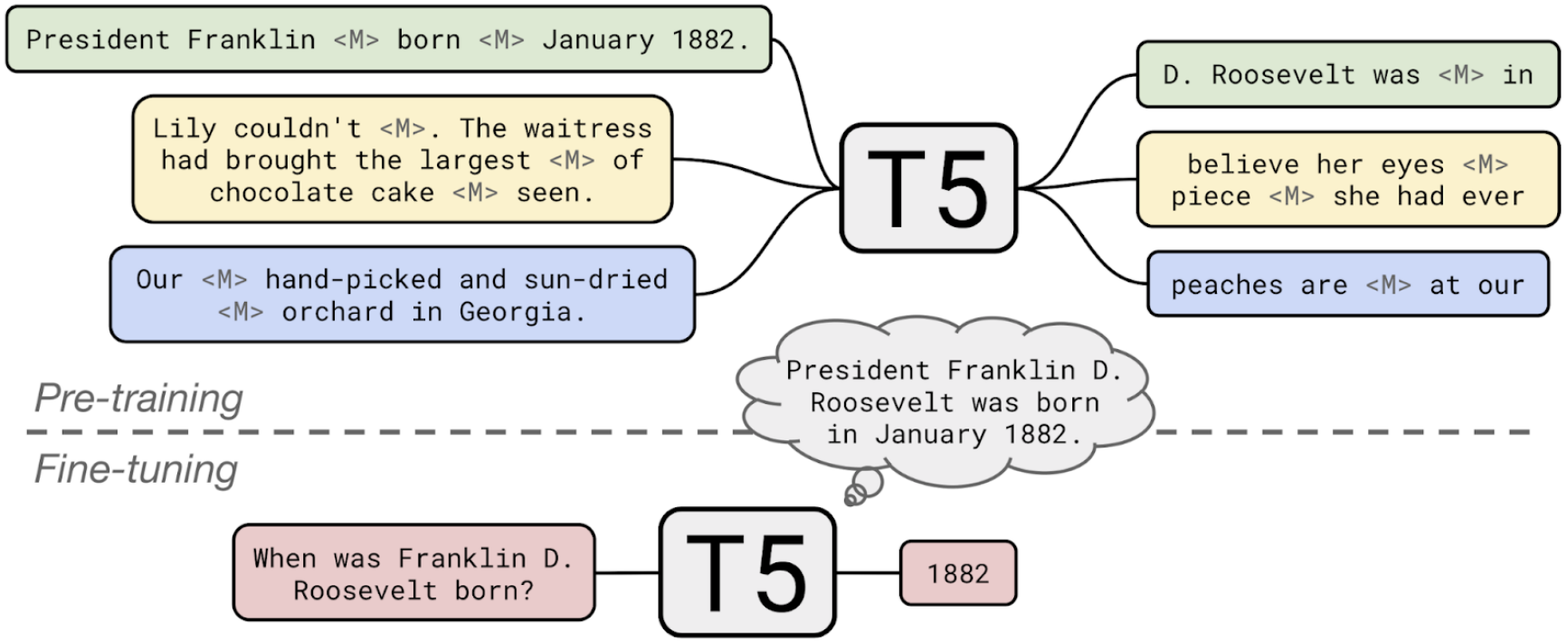
Training time: text: relevant / not relevant

Prediction time: text

T5: sequence-to-sequence



T5: sequence-to-sequence



Transformers

State-of-the-art technology in NLP and other areas

Based on **Attention**

Very interesting, and a great open source community:
https://huggingface.co/models

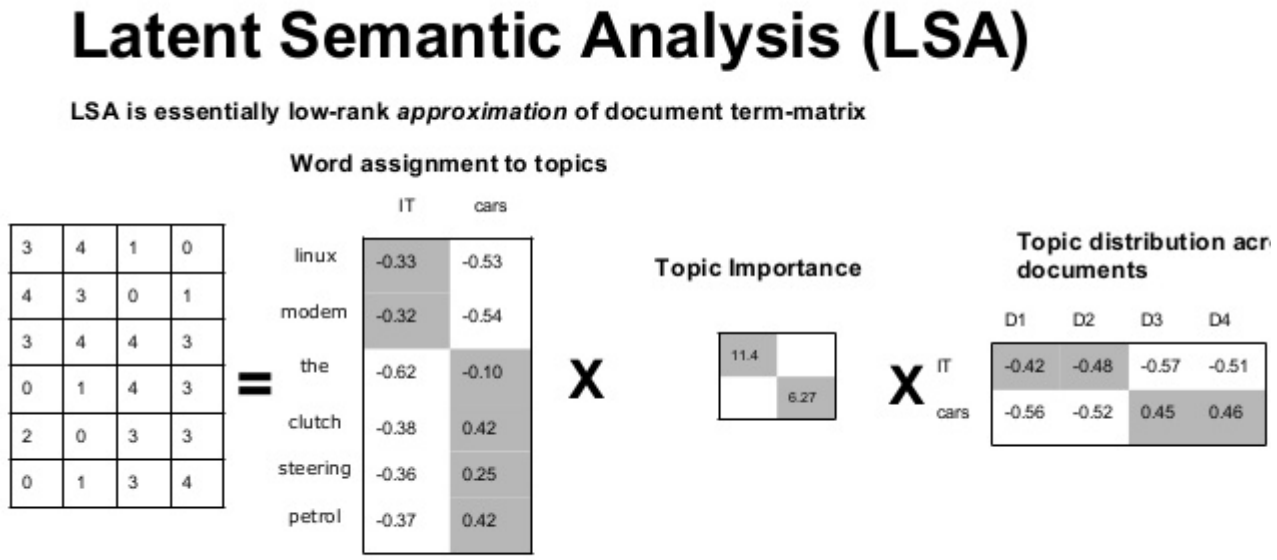
count-based models. LSA

(Deerwester et al 1990)

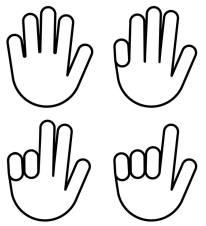
Singular Value Decomposition applied to a term-doc matrix

extracts **latent concepts**

The factorized matrix can be employed to produce **doc vectors** (in the conceptual space) or **word vectors**



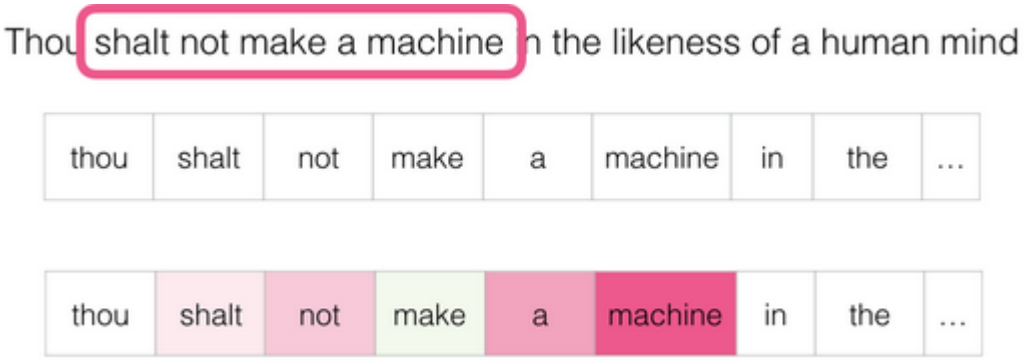
other count-based approaches. HAL



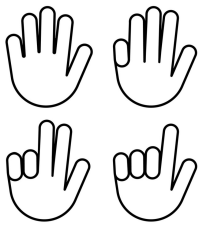
Hyperspace Analogue to Language, HAL (Lund and Burgess 1996)

for each word in the vocabulary, **analyze all contexts it appears in** and calculate the **co-occurrence count between the target word and each context word,** inversely proportional to the distance from the context word to the target word.

good results with a context window size of 8.



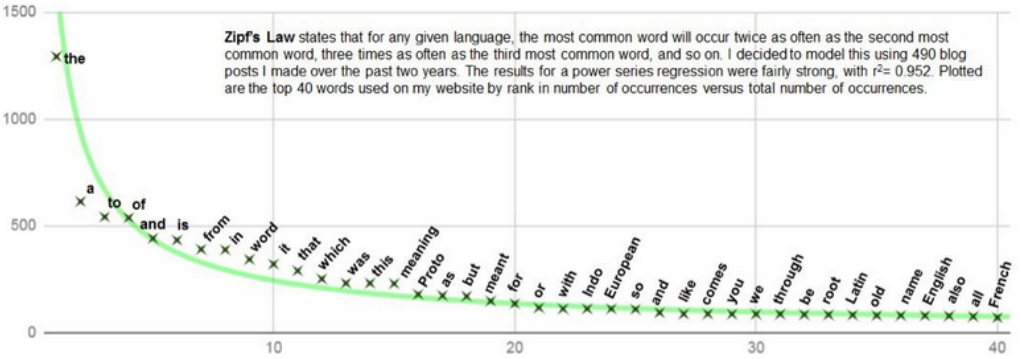
other count-based methods. COAL



COAL (Rohde et al. 2006)

HAL does not apply any normalization to word co-occurrence counts.

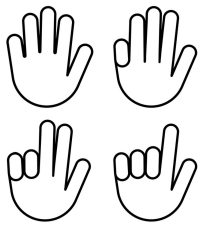
Therefore, **very common words**
like “the” contribute disproportionately
to all words that co-occur with them



COAL introduced normalization strategies to factor out such frequency differences in words.

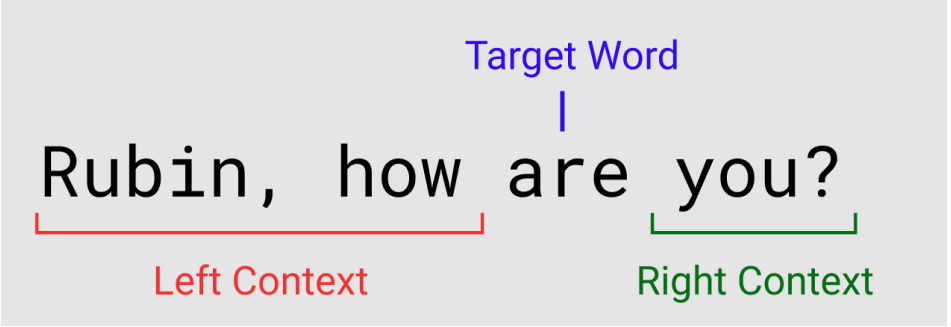
instead of using raw counts, they suggest it’s better to consider the **conditional co-occurrence**, i.e. how much more more likely a word a is to co-occur with word b than it is to co-occur with a random word from the vocabulary.

other count-based approaches. LR-MVL



Dhillon et al. 2011: **Low Rank Multi-View Learning** (LR-MVL)

iterative algorithm where embeddings are derived by leveraging **Canonical Correlation Analysis (CCA)** between the left and right contexts of a given word

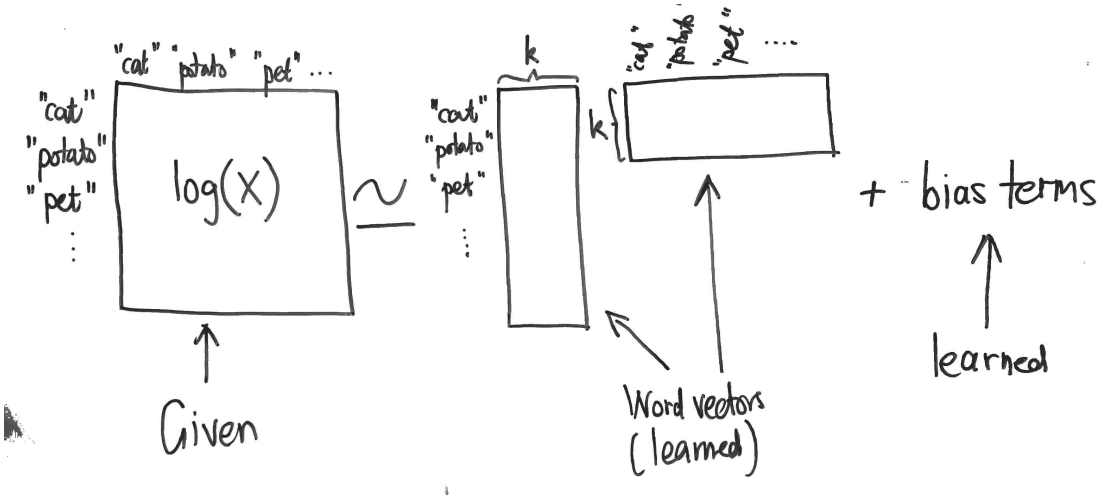


other count-based approaches. Glove

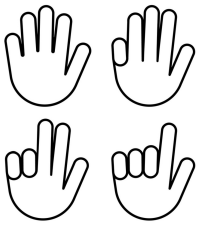
(Pennington et al. 2014)

Word2vec only focuses on the information obtained from local context window while the **global statistic information is not used well**.

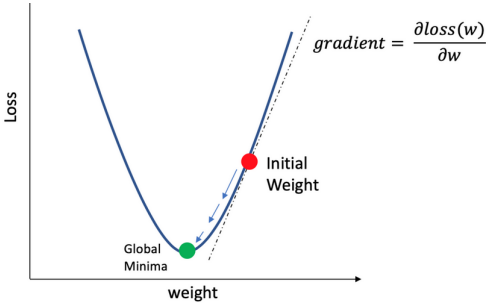
based on the global co-occurrence matrix, Xij in the matrix represents the frequency of the word wi and the word wj co-occur in a particular context window.



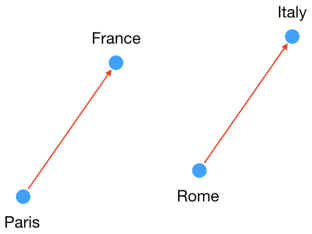
other count-based approaches. Glove



GloVe: **ratios of co-occurrences**, rather than raw counts, encode actual semantic information about pair of words. This relationship is used to derive a suitable **loss function** for a log-linear model, which is then trained to maximize the similarity of every word pair, as measured by the ratios of co-occurrences mentioned earlier.



better results than other count-based models, as well as some prediction based models , in tasks such as **word analogy** and **named entity recognition**.



final remarks. challenges

Out-of-vocabulary (OOV) words. The datasets used for training cannot completely include all words and there will always be a steady stream of new words, which causes the problem of how to deal with the OOV words.

Word2Vec, for example, cannot learn embeddings

for OOV words

OOV words: words that are not included in the existing vocabulary and words that do not appear in the existing training corpus. OOV words can be roughly divided into the 4 types including: (1) **Emerging common vocabulary** (such as online terms, etc.). (2) **Proper names** (such as names of people, names of places and organizations, time and digital expressions, etc.). (3) **Professional nouns and research field names**. (4) **Other terminology** (such as the name of a new product, the name of a literary work such as a movie or a book, etc.).

tools & libraries

Hugging Face

Keras pandas

scikit learn TensorFlow

many of the advances seen in the literature have been incorporated in widely used **toolkits**, such as Hugging Face, Word2Vec, Gensim, FastText, and GloVe, resulting in ever more accurate and faster word embeddings, **ready to be used in NLP tasks**.

GENSIM
topic modelling for humans

fastText