Advanced Human Language Technologies

Word Embeddings

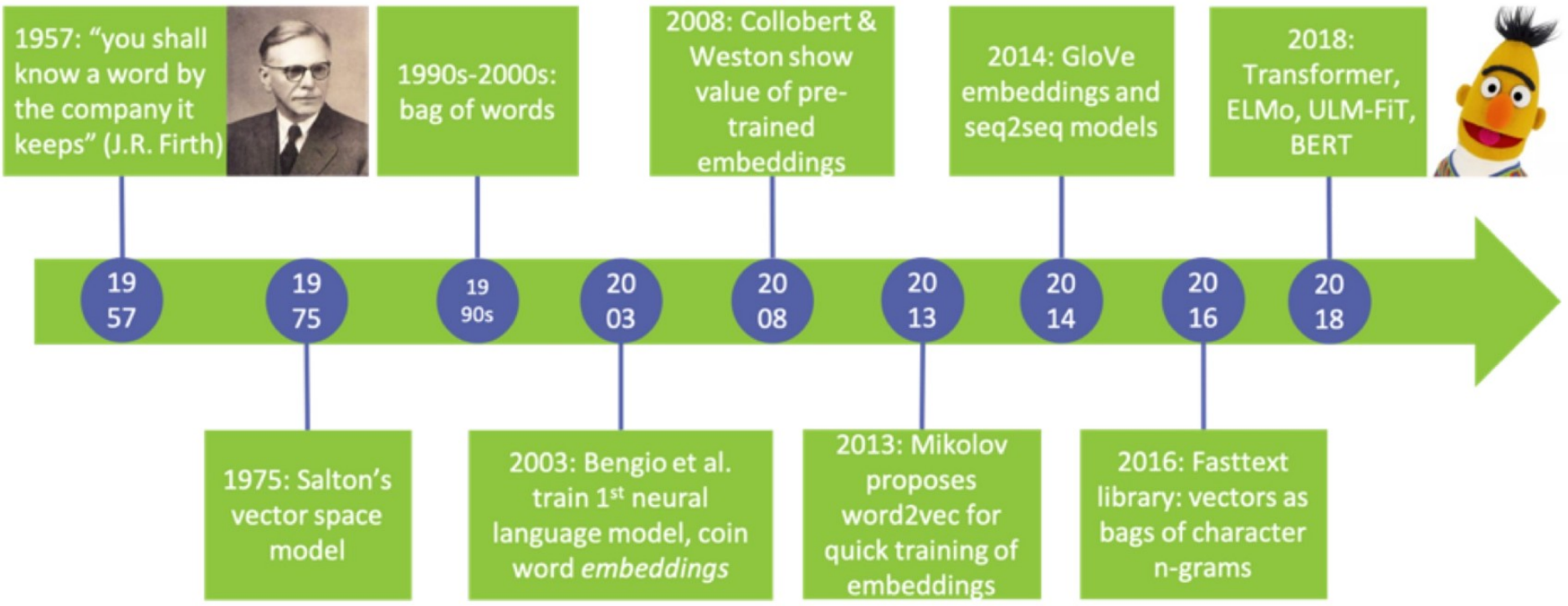
Master on Artificial Intelligence

Lluís Padró
Computer Science Department – UPC
padro@cs.upc.edu

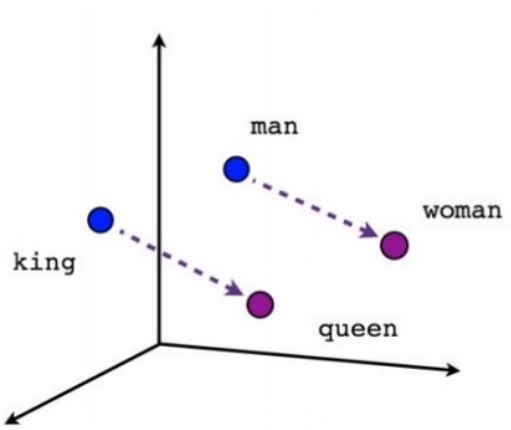
Distributional semantics hypothesis

- Never ask for the meaning of a word in isolation, but only in the context of a sentence (Frege, 1884)
- For a large class of cases… the meaning of a word is its use in the language (Wittgenstein, 1953)
- You shall know a word by the company it keeps (Firth, 1957)
- Words that occur in similar contexts tend to have similar meaning (Harris, 1954)

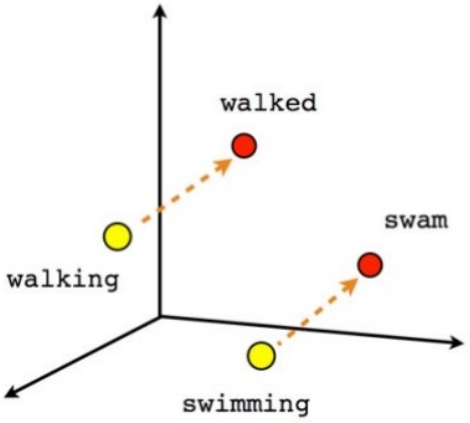
Word Embeddings: Timeline



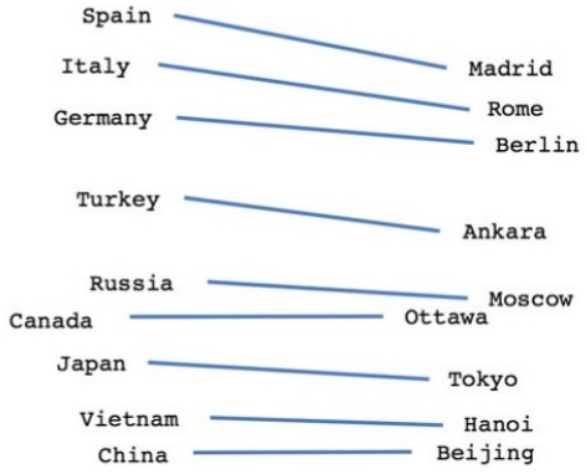
Word Vectors



Male-Female



Verb tense



Country-Capital

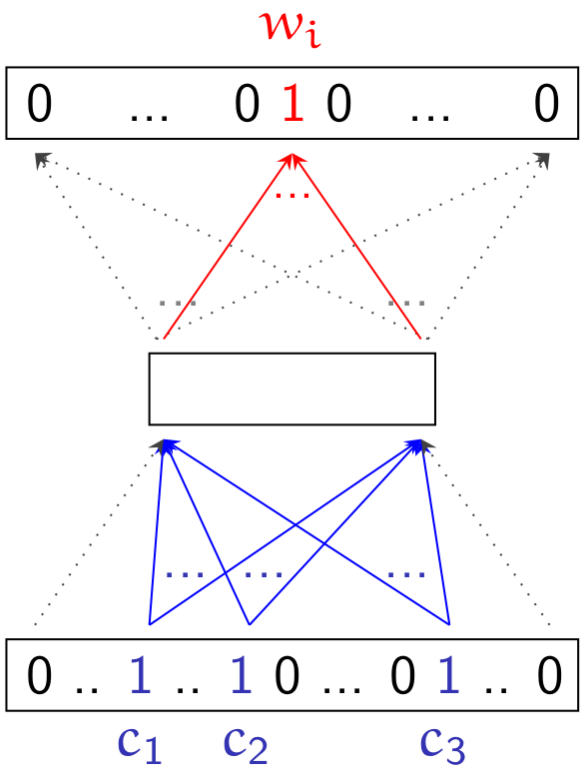
Word2Vec (Mikolov 2013)

2 methods to pretrain embeddings

- CBOW (Continuous bag-of-words)
  - Train a NN that given a set of context words, learns to predict the missing one.
- Skip-gram
  - Train a NN that given a word, learns to predict likely words in its context.

Word2Vec: CBOW

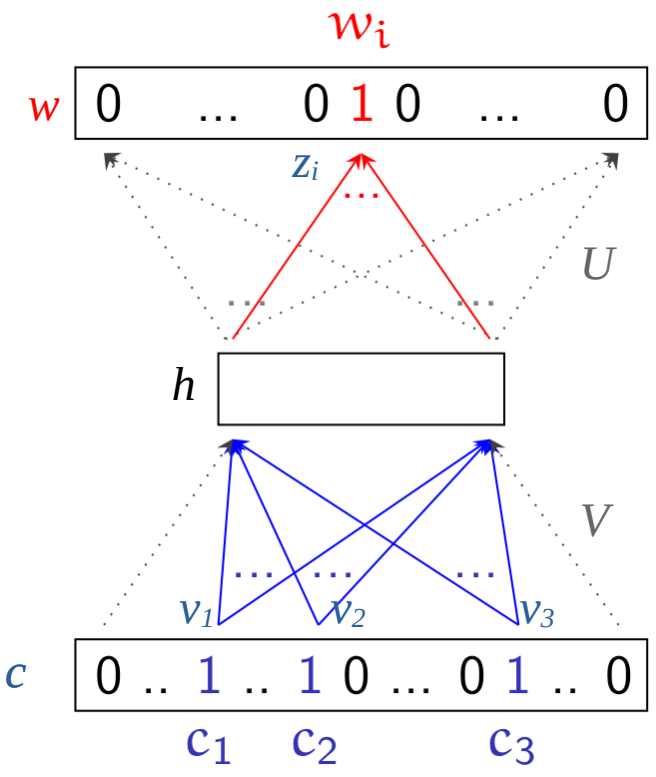
CBOW



- A dense low-rank vector representing a word is extracted from the internal representation of a Neural Network.
  - Word context words c_1, c_2, c_3 appearing around word w_i are used as a training example.
  - The NN is trained to predict the missing word for a given input context.
  - The hidden layer input weights encode the distribution of likely words in each context.
  - Words appearing in similar contexts will have similar hidden layer weights.
  - Hidden layer weights for each word are used as their dense vector representation

Word2Vec: CBOW

CBOW



- c : input vector, context words
  V : input layer weights
  h_c : hidden layer state for input c

$$h_c = V \cdot c = \sum_i V_i^T \cdot c = \sum_{i:c_i=1} V_i^T$$

- U : hidden layer weights
  z : output scores

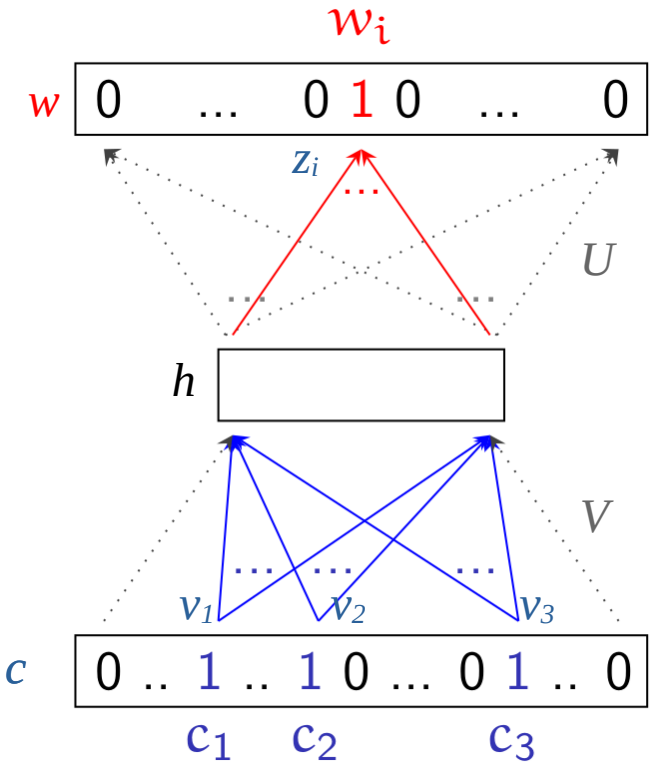
$$z_i = U_i \cdot h_c$$

- Softmax to convert scores to prob. distribution

$$w_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} = \frac{\exp(U_i \cdot h_c)}{\sum_j \exp(U_j \cdot h_c)}$$

Word2Vec: CBOW

CBOW



- Training: Use cross entropy of estimated probability distribution $Q(w)$ with respect to gold distribution $P(W)$

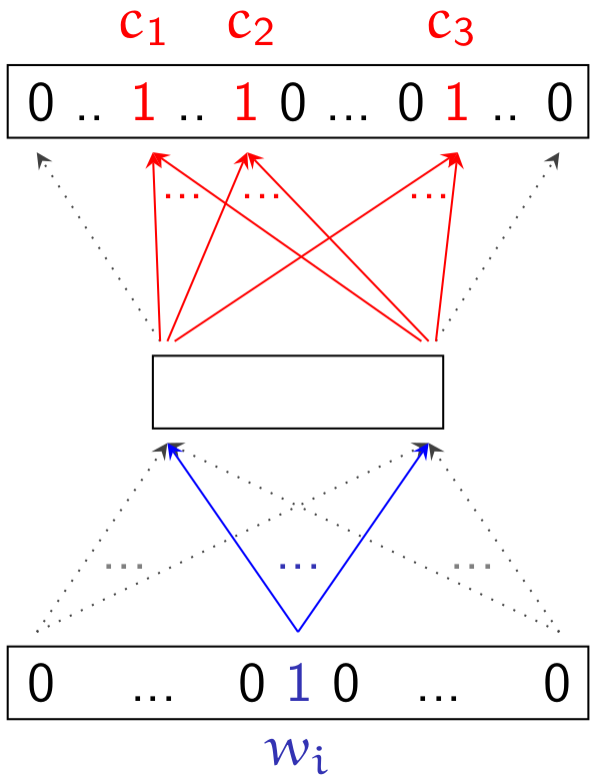
$$\begin{aligned}\mathrm{CE}(P,Q) &= E_P[-\log Q(w)] = -\sum_w P(w)\log Q(w)\\ &= E_P[-\log Q(w)+\log P(w)-\log P(w)]\\ &= E_P[\log \frac{P(w)}{Q(w)}]+E_P[-\log P(w)]\\ &= D_{KL}(P\|Q)+H(P)\end{aligned}$$

- In our case equals to minimizing the negative log-likelihood of the target vector given the context

$$\begin{aligned}\mathrm{Loss} &= -\sum_w P(w)\log Q(w) = -1\cdot\log Q(w_i|c)\\ &= -\log \frac{\exp(U_i\cdot h_c)}{\sum_j \exp(U_j\cdot h_c)}\\ &= -U_i\cdot h_c+\log\sum_j \exp(U_j\cdot h_c)\end{aligned}$$

Word2Vec: Skip-gram

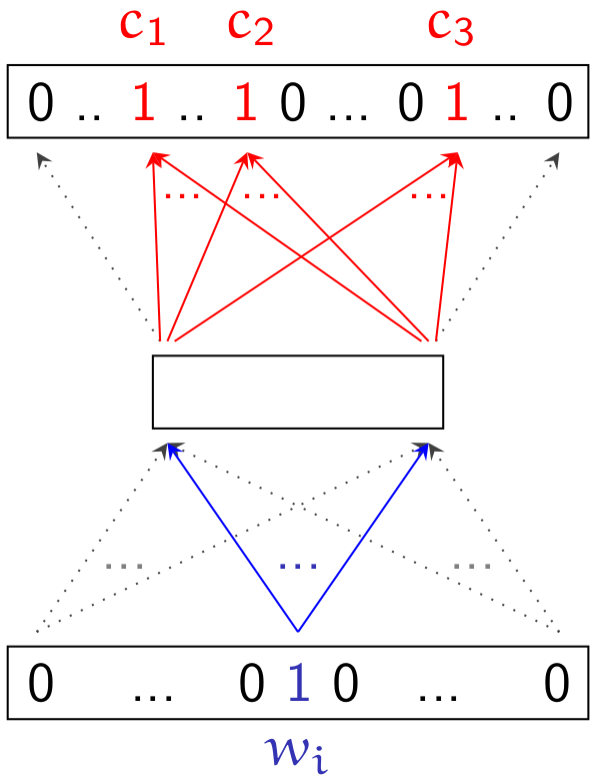
Skip-gram



- A dense low-rank vector representing a word is extracted from the internal representation of a Neural Network.
  - Word w_i appearing near context words c_1, c_2, c_3 is used as a training example.
  - The NN is trained to predict usual context words for a given input word.
  - The hidden layer input weights encode the usual contexts of each input word.
  - Word appearing in similar contexts will have similar hidden layer weights.
  - Hidden layer weights for each word are used as their dense vector representation

Word2Vec: Skip-gram

Skip-gram

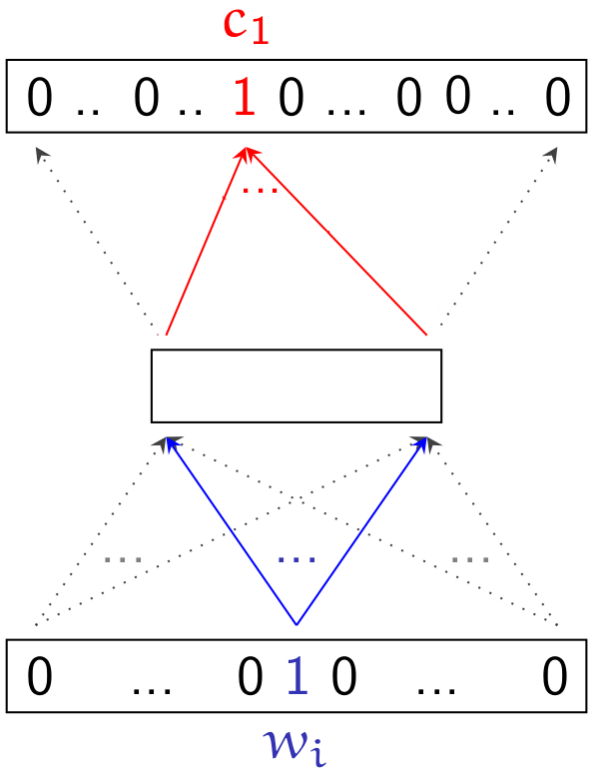


- **BUT**:
  - The size of the vocabulary is usually large (e.g. 10,000 words)
  - If the hidden layer has, e.g. 300 dimensions, that means 300 x 10,000 x 2 parameters to update for each training example.

Too high computational cost

Word2Vec: Skip-gram

Skip-gram



- We show the network only one target word at a time, so the amount of parameters to update is much smaller

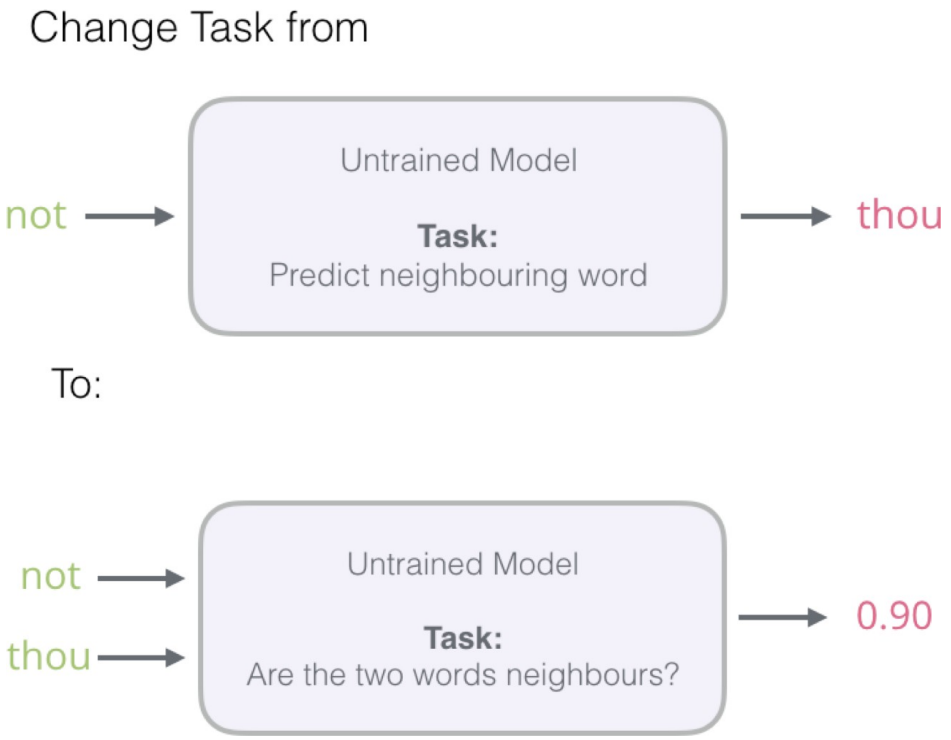
But...

- If we use only positive examples, the network will overgeneralize and may end up assigning prob 1 to all words

We need to introduce negative examples

Skip-gram with negative sampling

- We change the task to predict whether the words belong in the same context (0 or 1)



Negative examples

- This can computed very efficienty – processing millions of examples in minutes.
- We can afford to introduce some negative examples at a resonable cost

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Negative examples

- We add a few **negative examples** for each positive pair in the dataset, with the same input word, a non-related context word, and a 0 label.

input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

Negative examples

We are contrasting the actual signal (positive examples of neighboring words) with noise (randomly selected words that are not neighbors), allowing the model to learn a trade-off.

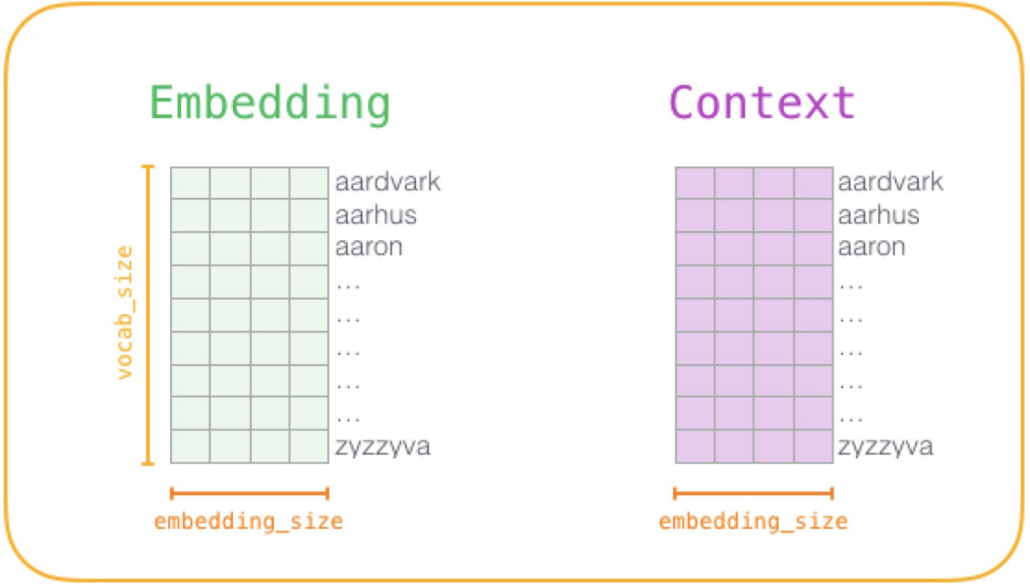
Training process

1) Pre-process the training text:
   - Determine model vocabulary (which words are to be included in the model, and which will be the size of the model (number of vectors) |*V*|
   - Decide the dimension of the embedding *E*. Usual values are 100, 200, 300...

2) Create two matrices:
   - Embedding matrix |*V*|x*E*
   - Context matrix: |*V*|x*E*

   Both matrices have an entry for each word in the vocabulary.

   Both are initialized with random values.



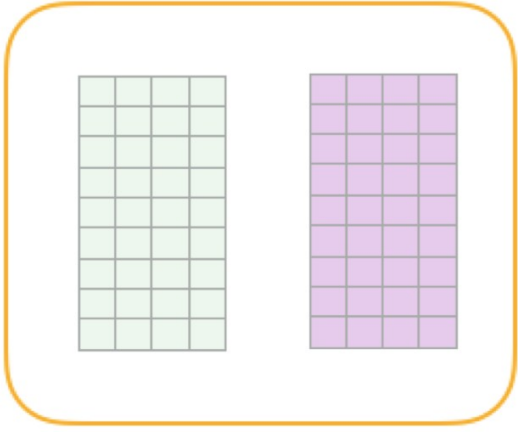
Training process

- In each training step, we take one positive example and its associated negative examples.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

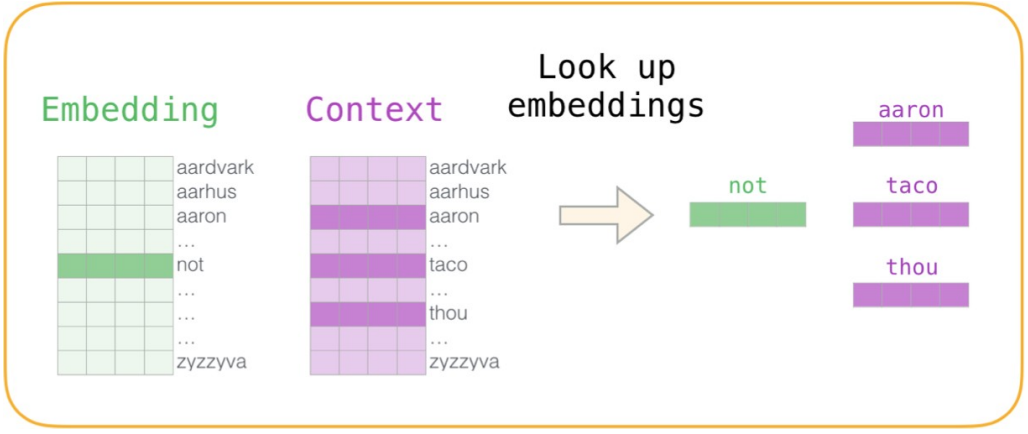
model



Training process

- For each eaxample, we have four words:
  - input word: *not*
  - output/context words:
    - *thou* (actual neighbor)
    - *aaron*, *taco* (negative examples).

- **Step 1**: Look up current embeddings
  - For input word, we look up in the Embedding matrix.
  - For context words, we look up in the Context matrix



Training process

- **Step 2:**
  - Compute the **dot product** of the input word embedding with each of the context words embeddings, obtaining a similarity value
  - Use sigmod to convert these scores to probabilty-like values

input word	output word	target	input • output	sigmoid()
not	thou	1	0.2	0.55
not	aaron	0	-1.11	0.25
not	taco	0	0.74	0.68

Note that *taco* has the highest score and *aaron* still has the lowest score both before and after the sigmoid operations.

Training process

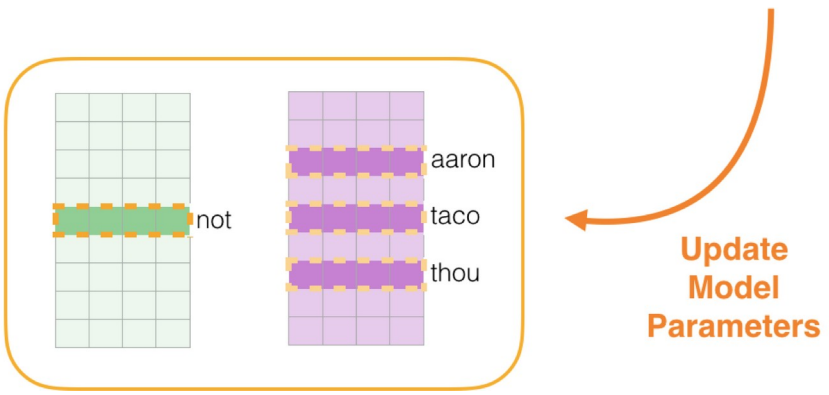
- **Step 3:**
  - Compute error in the model prediction.
    (subtract the sigmoid scores from the target labels)

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

Training process

- **Step 4:** Update model.
  Use error score to **adjust the embeddings** of *not*, *thou*, *aaron*, and *taco* so that the next time the result would be closer to the target scores

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



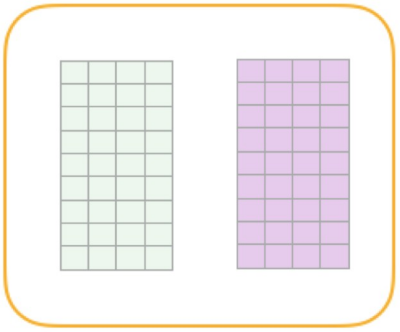
Training process

- After processing this example, embeddings for the words involved in this step are slightly improved. We can proceed to the next example and repeat the process:

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



- Cycle through the entire dataset a number of times (or *epochs*).
- Finally, discard the Context matrix, and use the Embeddings matrix as our pre-trained embeddings model

Other methods: Not simply words

- Instead of words, other units may be used:
  - **Phrases**: **Washington_Post** is a newspaper
    Phrases can be generated using counts, e.g. $PMI = \log \frac{P(w_1, w_2)}{P(w_1) \cdot P(w_2)}$
  - **Characters**: W a s h i n g t o n _ P o s t _ i s _ a _ n e w s p a p e r
    Create an embedding for each character, word representations are created combining its character vectors.
  - **Subwords**: Wash #ing #ton Post is a new #spaper
    Different strategies to decide what subwords are used: n-grams, Byte Pair Encoding (BPE), Wordpiece, Sentencepiece, ...

Other methods: fastText

- **fastText**: Subword (n-gram) based skip-gram.
  - Embedding vector for a word w is the sum of the embeddings of its n-grams ($3 \leq n \leq 6$).
  - e.g. The fastText representation of word *where* is the sum of 15 embedding vectors, for the n-grams:
    - 3-grams: _wh, whe, her, ere, re_
    - 4-grams: _whe, wher, here, ere_
    - 5-grams: _wher, where, here_
    - 6-grams: _where, where_
    - Whole word: _where_

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- Ratios of word-word co-occurrence probabilities have the potential for encoding semantic similarity

$$P(j|i) = \frac{x_{ij}}{x_i}$$

X_{ij} : occurrences of word j the context of word i

X_i : occurrences of any word in the context of word i

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- ***solid*** has a much higher co-occurrence probability with ***ice*** than with ***steam***

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- **gas** has a much higher co-occurrence probability with ***steam*** than with ***ice***

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- ***water*** has a similar (high) co-occurrence probability with both ***ice*** and ***steam***

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- ***fashion*** has a similar (low) co-occurrence probability with both ***ice*** and ***steam***

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- Training: Learn two vectors for each word, such that we have a multinomial logistic regression of the the co-ocurrence probability

$$w_i \cdot \tilde{w}_j \approx \log P(j|i)$$

- We can solve it minimizing :

$$L = \sum_{i,j \in V} (w_i \cdot \tilde{w}_j - \log P(j|i))^2$$

- To reduce noise introduced by rare co-occurrences, the loss is actualy weigthed:

$$L = \sum_{i,j \in V} f(X_{ij}) (w_i \cdot \tilde{w}_j - \log P(j|i))^2$$

where $f(X_{ij})$ is a weight in [0,1] that grows with the co-occurrence frequency X_{ij}

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- Stochastic gradient descent (SGD) is used to minimize the squared error between the dot product of word vectors and the logarithm of their co-occurrence count.
- Once the model is trained, we have 2 vectors for each word
  - w_i : vector representing word i as the center word
  - $\widetilde{w}_i$: vector representing word i as a context word
- Either one of them (or their sum $w_i + \widetilde{w}_i$) are used as the word representation vector.

Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- Stochastic gradient descent (SGD) is used to minimize the squared error between the dot product of word vectors and the logarithm of their co-occurrence count.
- Once the model is trained, we have 2 vectors for each word
  - w_i : vector representing word i as the center word
  - $\widetilde{w}_i$: vector representing word i as a context word
- Either one of them (or their sum $w_i + \widetilde{w}_i$) are used as the word representation vector.

No neural net involved

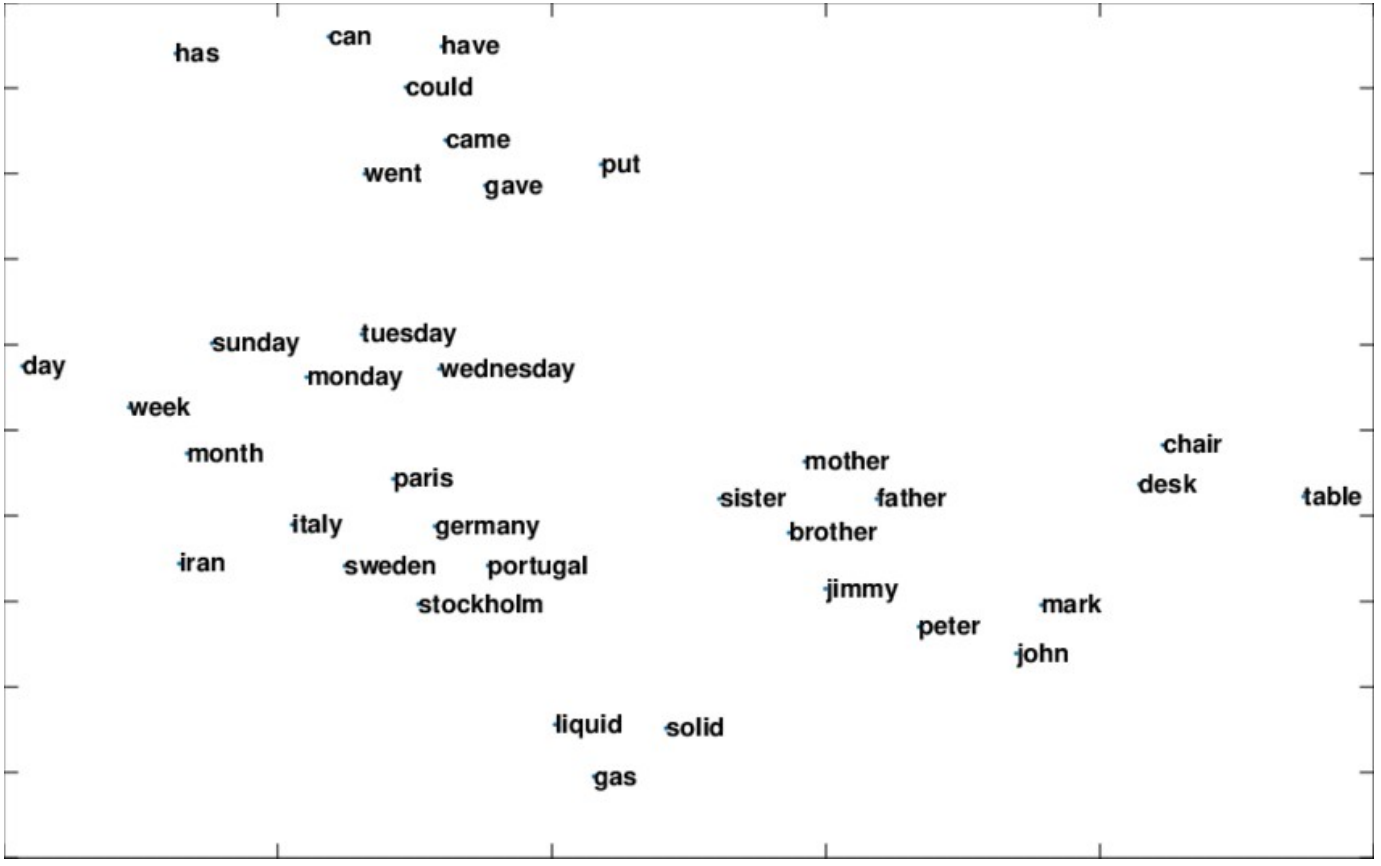
Other methods: GloVe

GloVe: Global Vectors for Word Representation.

- GloVe captures both local and global statistical information from text.
- More effective for analogy tasks due to its explicit modeling of word relationships.
- Often results in better embeddings for uncommon words compared to Word2Vec.

Embedding Space

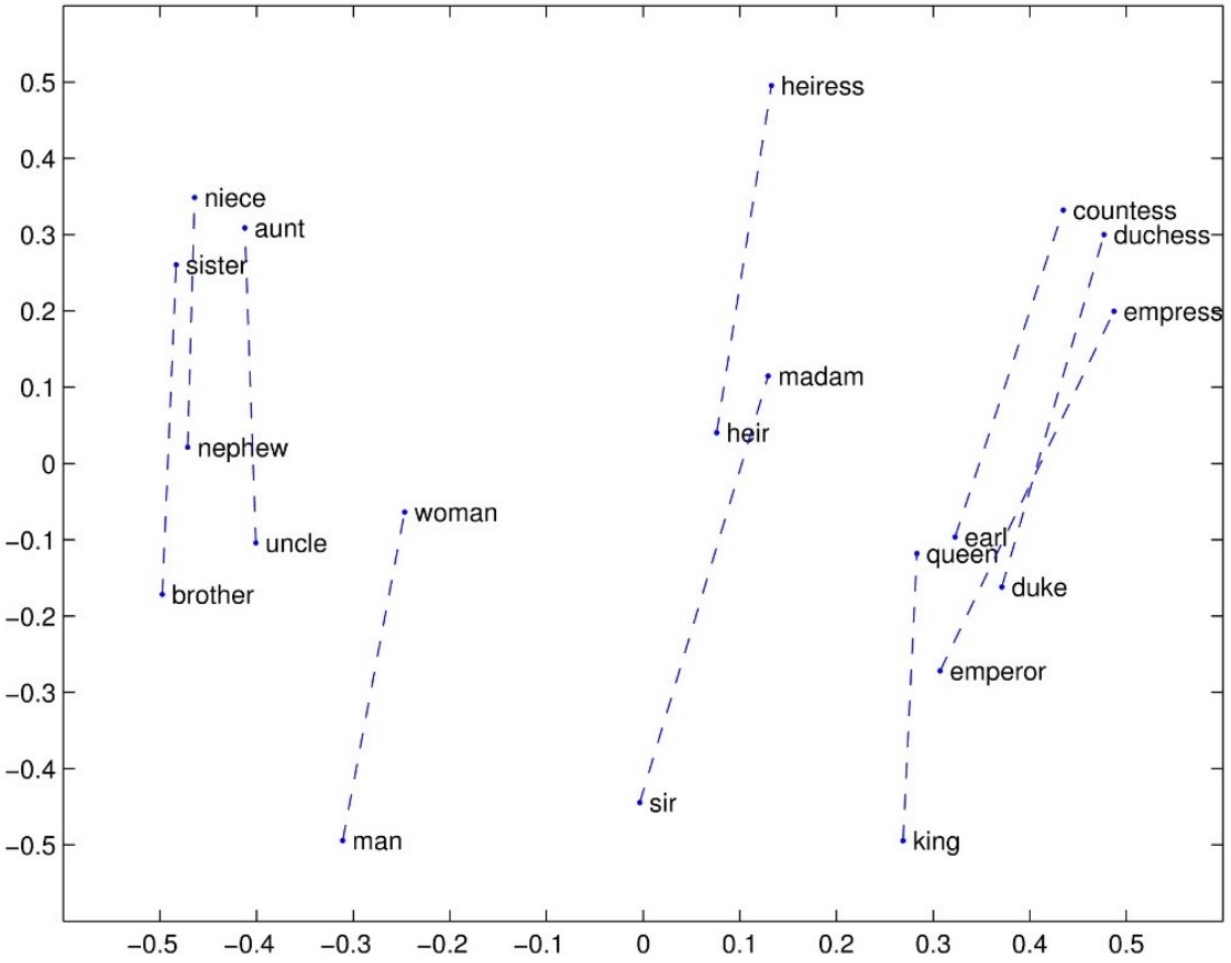
Distributional semantics methods produce close vectors for words occurring in similar contexts



Source: Ali Basirat 2018, Principal Word Vectors, PhD Thesis, Uppsala Univ.

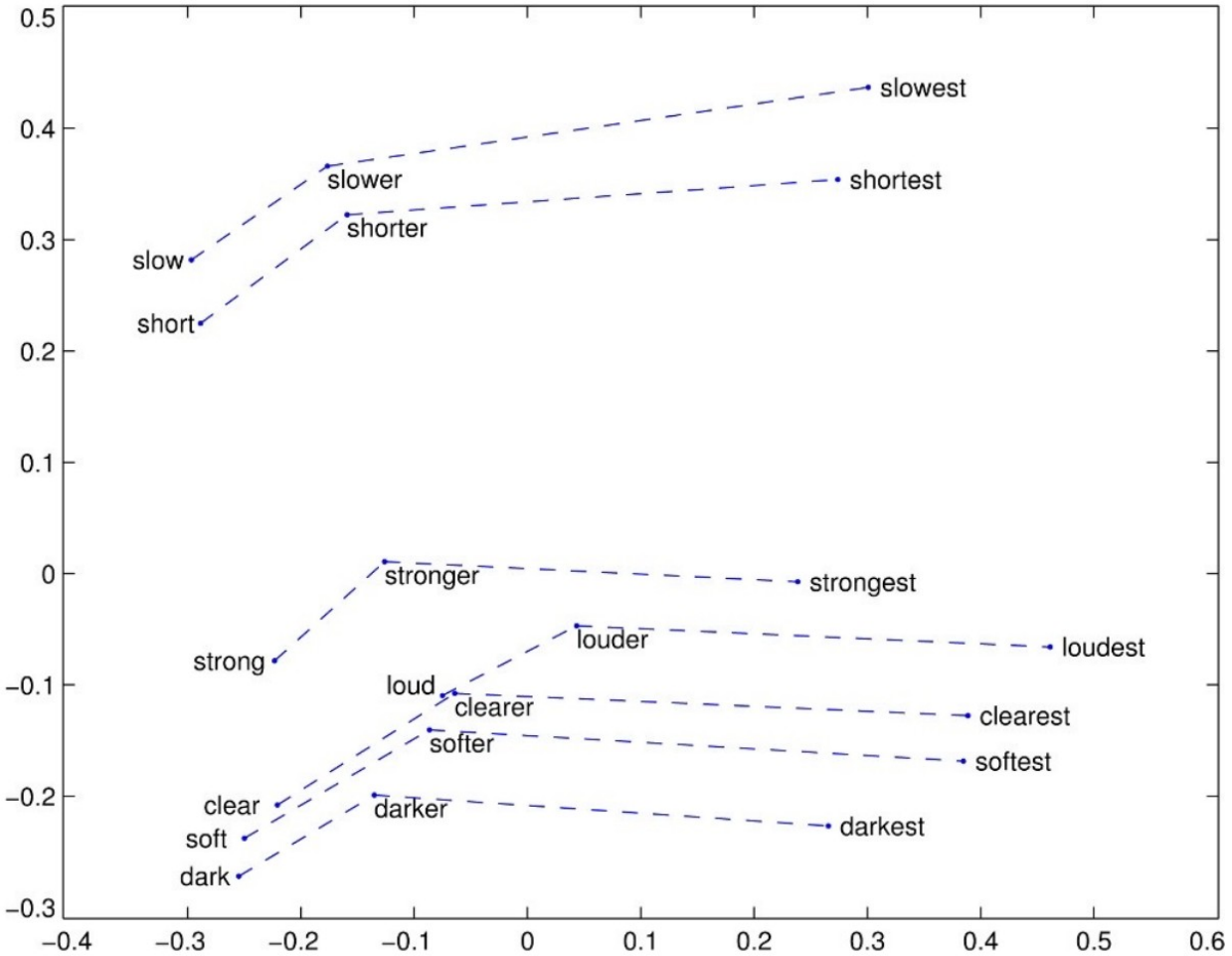
Embedding Space

Moreover, word embeddings keep similar distance between words with the same semantic relationship.



Embedding Space

Word embeddings also keep similar distances between words with the same morphological relationship.



Embedding Space

- These spatial relations can be modeled and exploited using *vector operations*:
  - [wednesday] + ([tuesday] – [monday]) = [thursday]
  - [three] + ([two] – [one]) = [four]
  - [three] + ([2] – [two]) = [3]
  - [lives] + ([knew] – [knows]) = [lived]
  - [this] + ([those] – [that]) = [these]
  - [Poland] + ([French] – [France]) = [Polish]
  - [uncle] + ([woman] – [man]) = [aunt]

Embedding evaluation

- Intrinsic Evaluation
  - Use embeddings for tasks that can be directly evaluated
    - *Similarity*:
      Find closest word to *w*: $\cos(w_1, w_2) = \frac{w_1 \cdot w_2}{|w_1||w_2|}$
    - *Analogy*:
      w_a is to w_b like w_c is to <?>
      Find w_d that is the closest to $w_c+(w_b-w_a)$
- Extrinsic Evaluation
  - Use embeddings in downstream task (translation, sentiment analysis, text classification, etc.) and evaluate impact on task performance.

Limitations

- **Compositionality:** Word embeddings do not encode composited meanings such as *not happy* ↔ *sad* or *hot dog* ↔ *sausage*, since they rely on statistical co-occurrence rather than syntactic structure or adjacency.
- **Polysemy:** Words like *bat* (animal vs. sports equipment) or *apple* (fruit vs. company) have a single vector in traditional embeddings, making it impossible to distinguish meanings. Contextual embeddings (e.g., BERT, ELMo) address this issue but require a different model structure. [see session *8.Transformers*]
- **Rare/unseen words:** Traditional embeddings assign fixed vectors to words seen during training, meaning rare or unseen words are not well-represented. Subword-based models like fastText aim to address this, but limitations still exist.
- **Hierarchical/Ontological Relationships**: Word embeddings do not naturally capture strict hierarchical structures like *dog* → *mammal* → *animal* or part-whole relationships like *wheel* → *car*.
- **Causal/Temporal Relationships:** They do not capture cause-effect relationships, such as *smoking* → *cancer* or *winter* → *cold*.

Contextual Embeddings

- The same word may have different meanings in different contexts:
  - *Please **type** everything in lowecase*
  - *What **type** of food do you like?*
- Static word embeddings provide a unique representation for each word, regardless of the context. Polysemous words get mixed representations.
- Contextual embeddings provide different representations for the same word in different contexts.

Contextual Embeddings

- *Contextual* embeddings are trained (similarly to static embeddings) on a task that requires learning word representations.
- *Static* word embedding models are simply a vector with the hidden layer weights for each word.
- Contextual word embedding models are *dynamic*, and produce on-the-fly the embeddings for each word in the sentence.
- This pre-trained model can be:
  - Straightforwardly used to compute embeddings which will be input for another task.
  - Fine-tuned to perform a different task.

Contextual Embeddings

Unlike static embeddings, contextual embeddings generate different word vectors based on surrounding context.

- **ELMo** (2018) – BiLSTM-based, captures deep contextual & morphological info.
- **BERT** (2019) – Transformer-based, deep *bidirectional* representations (MLM & NSP).
- **XLNet** (2019) – Improves BERT using permutation-based training.
- **RoBERTa** (2019) – Optimized BERT with more data & no NSP.
- **T5** (2020) – Encoder-decoder model, unifies NLP tasks in a text-to-text format.
- **GPT** (2018–2023) – Transformer decoder (*unidirectional*), excels in generative tasks.

Acknowledgements

- Slides in this session are based on ideas and images from lectures by:
  - Marta Ruiz Costa-jussà
  - José Adrián Rodríguez Fonollosa
  - Salvador Medina Herrera