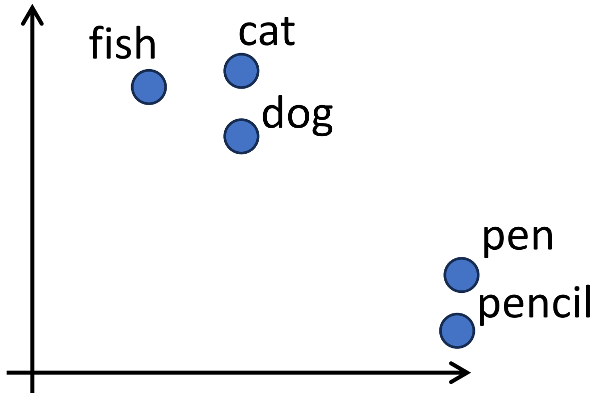
Large Language Models

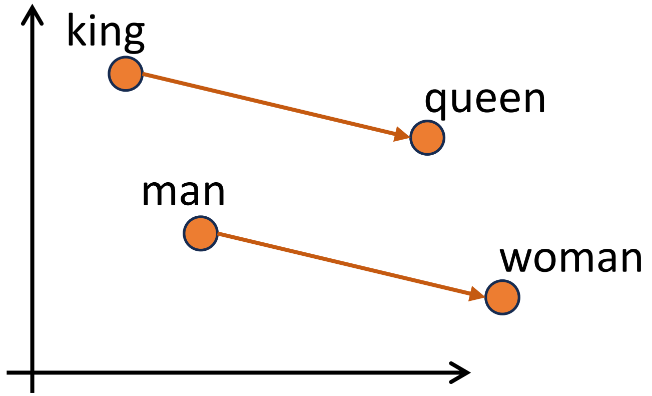
Word embeddings

Flavio Giobergia

What are word embeddings?

- Word embeddings are *dense* vector representations of words
  - (dense as opposed to sparse, e.g. one-hot encoding)
- Each word is mapped to a *vector of real numbers*
  - *High-dimensionalities (e.g. d=300 dimensions) are used to have "enough space" to represent various facets of the words)*
- Word embeddings capture *semantic meanings* and *relationships* between words
  - Words with similar meanings have similar representations
  - Words with similar connections (relationships) are linked via similar transformations





Sort of! https://gist.github.com/fgiobergia/b3a20e097f9b697d0a02fb17685cfd5a

Before word embeddings (one-hot encoding)

- *One-hot encoding* *does* allow us to build vector representations

- We assume a vocabulary W with |W| words
  - E.g., W = { dog, cat, fish, pen, pencil }, |W| = 5
  - An order can be established among words (e.g., lexicographic)

- One-hot encoding creates for each of the |W| words, a |W|-dimensional sparse vector

- For the ith word, all dimensions are set to 0 except for the ith , which is set to 1

```
1. dog    ➔ [1 0 0 0 0]
2. cat    ➔ [0 1 0 0 0]
3. fish   ➔ [0 0 1 0 0]
4. pen    ➔ [0 0 0 1 0]
5. pencil ➔ [0 0 0 0 1]
```

Problems of one-hot encoding

- The vectors are *Sparse*
  - This leads to scalability issues
    - A standard vocabulary can have 50,000+ words, implying a 50,000-dimensional vector representation
    - Vectors are too sparse in the space to be useful (curse of dimensionality)
  - The vectorial space is not used efficiently
    - For a set of words W, |W|-1 are 0, only 1 is non-0

- The vectors are *Orthogonal*
  - There is no preservation of semantic similarity, or relationships
    - (Remember, we'd like words to be closer if they are similar, distant if dissimilar)
  - Here, all pairs of words have exactly the same distance (e.g., cosine, or Euclidean)
    - $\cos(w_1, w_2) = 0 \quad \forall\, w_1, w_2 \in W,\ w_1 \neq w_2$
    - $\mathrm{L}_2(w_1, w_2) = \sqrt{2} \quad \forall\, w_1, w_2 \in W,\ w_1 \neq w_2$

Distributed Representations

- One-hot encoding is a type of *local representation*
  - Each entity is represented by a unique, “isolated” identifier
- By contrast, *distributed representations* aim to distribute the information across several dimensions
- We *let* models (e.g., neural networks) learn these representations
  - By crafting a task, and letting the model solve it

Framing the right task

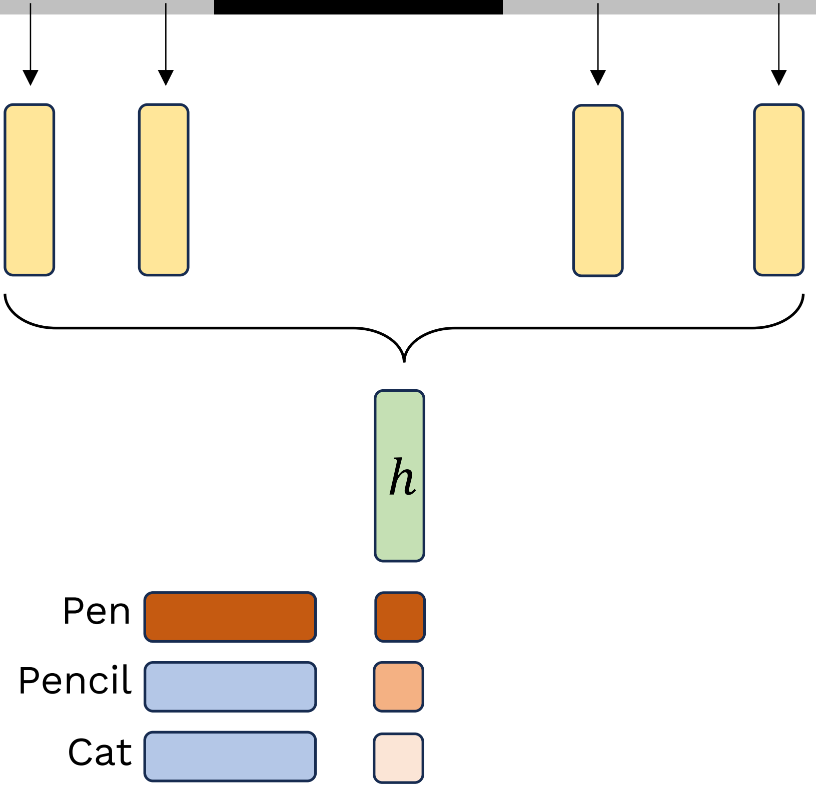
I used a [redacted] to write the essay

- *Task*: Can you fill the ***blank***, given some ***context***?

- In other words, can we estimate the probability that each word w is the correct one?
  - $\mathrm{P}(\mathrm{x_i} = w) = \mathrm{P}(\mathrm{x_i} = w | \mathrm{x_{i-1}}, \mathrm{x_{i-2}}, \dots, \mathrm{x_{i+1}}, \mathrm{x_{i+2}}, \dots)$

Solving the right task

I used a [redacted] to write the essay



Solution:

1. Assign each context word to a *vector* (random, at first!)
2. Aggregate all *vectors* into *a single one* (e.g., sum them!)
3. Assign each candidate output word to a *vector* (random too, at first!)
4. Compute the distance between the *context vector* and all possible *output words* (e.g., via dot product)
5. Find the *word* that best matches the *context*
6. Is it the *correct* word?
7. Adjust *all vectors* accordingly (via *gradient descent*)

… Rinse and repeat!

- The same process is applied to millions of sentences
- Similar words are found in similar contexts
- To solve the previous task, the word vectors of similar words must be similar!

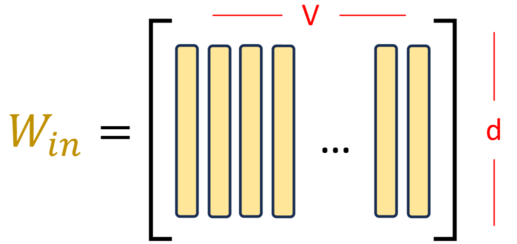
I used a pencil to write the essay

You used my pen to write a letter

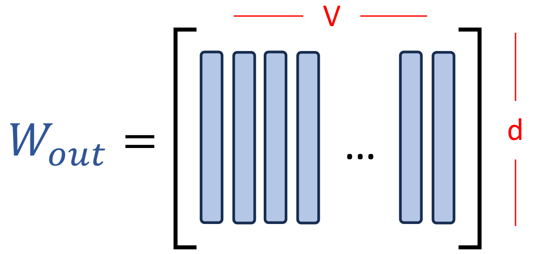
…

In terms of matrices (I)

- All vectors for input words can be stacked into a matrix, W_{in} $(d \times V)$



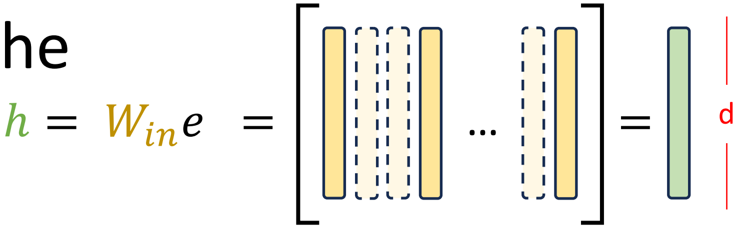
- All vectors for output words can also be stacked into a matrix, W_{out} $(d \times V)$



- The entire context can be represented a binary vector of presences, e $(V \times 1)$
  - Note: this loses the order of the word! (*bag of words* approach)

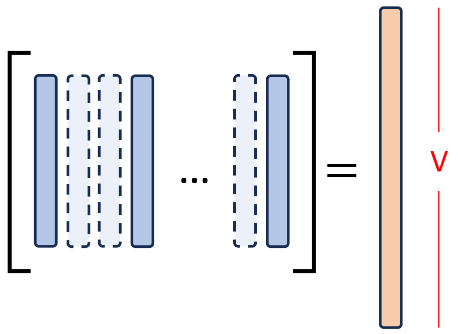
$$e_j = [\,1\;0\;0\;1\;\ldots\;0\;1\,]^T$$

- $h = W_{in} e (d \times 1)$ computes the sum of the vectors for the context words
  - e_j acts as a selector of the j^{th} column within W_{in}



In terms of matrices (II)

- Next, we search among the vectors in W_{out}, the most similar to h
  - We can use the dot product as a measure of similarity
    - ("how aligned are the vectors?")
  - $\tilde{p} = h^T W_{out}$ is a vector of similarities between h and each possible word
- We can normalize values in $\tilde{p}$ be positive and sum to 1
  - $p_i = softmax(\tilde{p})_i$
- Remember, we know what the correct target word is
  - We can use a cross-entropy loss to update W_{in}, W_{out}

$\tilde{p} = hW_{out} =$ 

Neural Language Models & word2vec

- Bengio et al [1] presented a similar approach to the previously described one in 2000
  - But with *causality* (i.e., predict next word from previous ones)

- word2vec [2] does all of the previous things, with some caveats.
  - Two possible tasks
    - *Continuous Bag of Words* (context ➔ predict middle word – as previously described)
    - *Skip-gram* (middle word ➔ predict context)
  - Workarounds to prevent computing softmax (computationally expensive!)
    - *Hierarchical softmax* (Huffman encode vocabulary, predict left/right path – $O(\log_2(V))$)
    - *Negative sampling* (predict whether a word is/is not the “correct” one)

[1] Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model." *Advances in neural information processing systems* 13 (2000).
[2] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems* 26 (2013).

Limitations of word2vec

- While word2vec addressed many problems, some still exist. Among others,
- *Inability to handle out-of-vocabulary words*
  - If a word is not in the training vocabulary, word2vec cannot generate a vector for it
- *Lack of contextualized vectors*
  - (after training,) word vectors are fixed and are not affected by context
  - For instance, the sentences “a *bat* is a mammal” and “the player swung the baseball *bat*” have very different meanings for *bat*. Word2vec doesn’t care about that.
  - When learning, word2vec “averages” all meanings of a word

FastText

- FastText addresses the out-of-vocabulary problem
- Breaking up words into subwords (e.g., tri-grams)
  - E.g., <where> ➔ <wh, whe, her, ere, re>

Note
< and > indicate beginning of word, end of word.
They can be used to assign different meanings to prefixes/suffixes.

- A vector representation is learned for each subword
- The vector for a word is given by the sum of the vectors of its subwords
- $v_{where} = v_{<wh} + v_{whe} + v_{her} + v_{ere} + v_{re>}$
- We can compose subword vectors to generate vectors for new words!

Visualizations – Semantic meanings

- 300-dimensional FastText vectors for words belonging to 3 separate categories
  - Household items
  - Mammals
  - Birds
- Reducing to 2 dimensions with Principal Component Analysis (for visualization purposes)
- The words belonging to the 3 categories are well-separated in the “compressed” embedding space
  - (they are also separated in the original latent space, but visualizing 300 dimensions is tricky)

Visualizations – Relationships

- Visualizing the vectors for *countries* and *capital cities*
- Connecting each country to its capital city
- We can see that there is a transformation (translation) that approximately connects each pair of words
- The vector of the translation can be obtained subtracting a capital city from its country
  - E.g., germany - berlin
  - Represents the relationship "capital of"
- We can apply this transformation by "adding" it to other countries
  - E.g. spain + "capital of" ➔ spain + germany - berlin