Foundations of Natural Language Processing

Lecture 10: Word Embeddings

Mirella Lapata
School of Informatics
University of Edinburgh
mlap@inf.ed.ac.uk

Slides based on content from: Philipp Koehn, Alex Lascarides, Sharon Goldwater, Shay Cohen, Khalil Sima'an, Ivan Titov, Lena Voita

Recap: Constructing Vector Spaces

Informal algorithm for constructing vector spaces:

- Select a corpus
- Select n target words which will be represented as vectors in the space;
- Select k dimension words (they are found around target word in the context window)
- compute $k \times n$ cooccurrence matrix
- Compute (PPMI): weighted cooccurrence matrix
- Compute similarity of any two focus words as the cosine of their vectors

Embedding Learning Algorithms

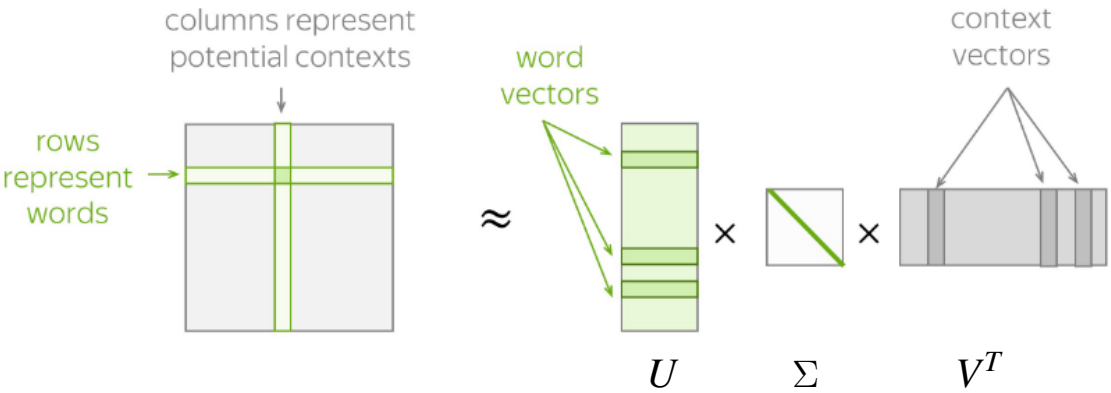
Singular Value Decomposition (SVD)

- Also called Latent Semantic Indexing
- Factorization of cooccurrence matrix
- Least squares objective optimized by power method

Embedding Learning Algorithms

Singular Value Decomposition (SVD)

- Also called Latent Semantic Indexing
- Factorization of cooccurrence matrix
- Least squares objective optimized by power method



Embedding Learning Algorithms

Singular Value Decomposition (SVD)

- Also called Latent Semantic Indexing
- Factorization of cooccurrence matrix
- Least squares objective optimized by power method

Word2Vec

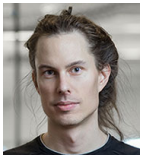
- Models used to learn word embeddings
- Optimized by gradient descent
- Skip-gram model predicts surrounding words (context) given a target word

Embedding Learning Algorithms

Word2Vec

- Models used to learn word embeddings
- Optimized by gradient descent
- Skip-gram model predicts surrounding words (context) given a target word

Word2Vect and Friends



Mikolov et al, 2013: Distributed Representations of Words and Phrases and their Compositionality

Input: a large text corpus, V, d

- V: a predefined vocabulary
- d: dimension of word vectors (e.g., 300)
- W: embedding matrix of size $V \times d$
- Text corpora: Wikipedia + Gigaword 5 (6B), Twitter (27B), Common Crawl (840B)

Output: $f : V \to \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Representing words as discrete symbols

How can we represent words which are discrete symbols as vectors?

Words can be represented by **one-hot** vectors:

$$\begin{aligned} \text{motel} &= [0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0] \\ \text{hotel} &= [0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0] \end{aligned}$$

- **Vector dimension** is the number of words in the vocabulary (e.g., 500,000).
- Skip-gram treats one-hot vector as an **index**.
- When a one-hot vector is multiplied by the embedding matrix W, it effectively extracts the corresponding row from W

One-hot vector example

One-hot vector: $\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

Embedding matrix: $W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix}$

One-hot vector example

One-hot vector: $\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

Embedding matrix: $W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix}$

Multiplication:

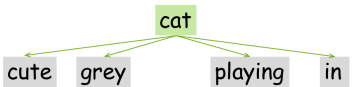
$$\mathbf{x}^T W = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix}$$

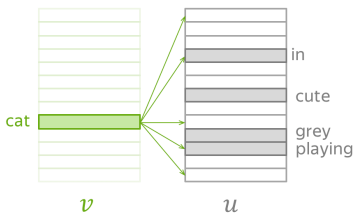
$$= \begin{bmatrix} w_{31} & w_{32} & w_{33} \end{bmatrix}$$

Result: The third row of W is selected as the word embedding.

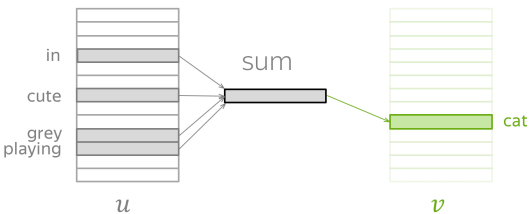
Word2Vec and Friends

… I saw a cute grey cat playing in the garden …





Skip-Gram: from central predict context (one at a time)



CBOW: from sum of context predict central

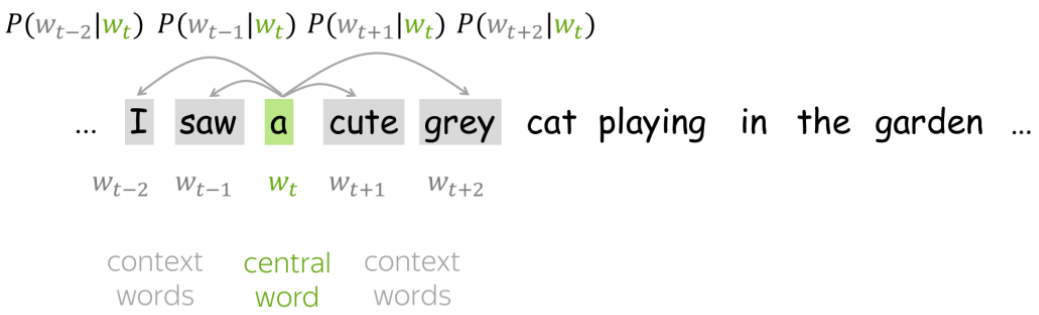
Skip-gram: Main Idea

Word2Vec is an iterative method. Its main idea is as follows:

- take a huge text corpus;
- go over the text with a sliding window, moving one word at a time. At each step, there is a central word $\in V$ word and context $\in V$;
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

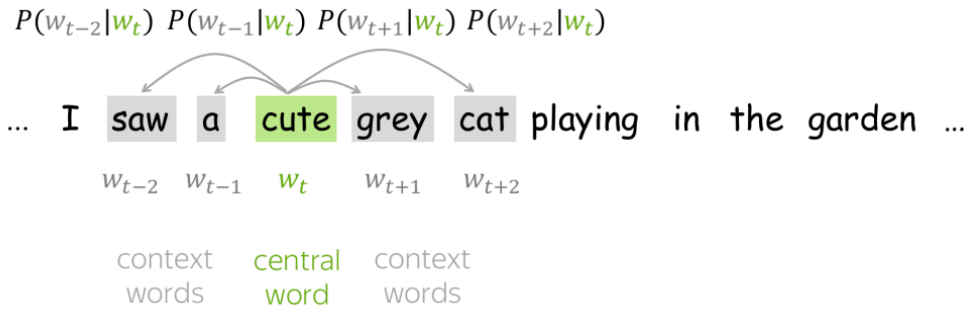
Skip-gram

The idea: we want to use words to predict their context words.



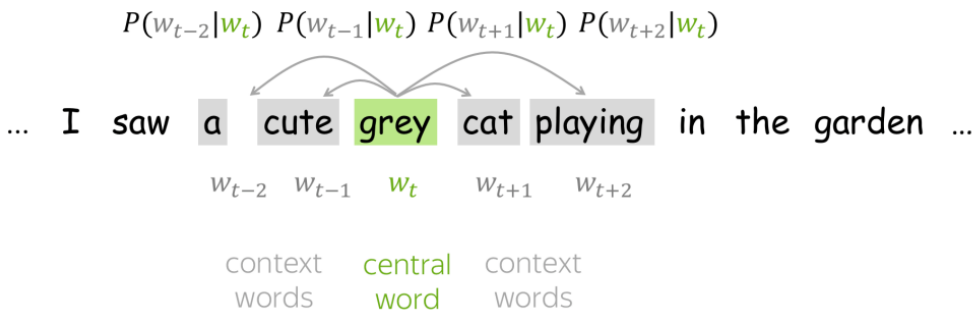
Skip-gram

The idea: we want to use words to predict their context words.



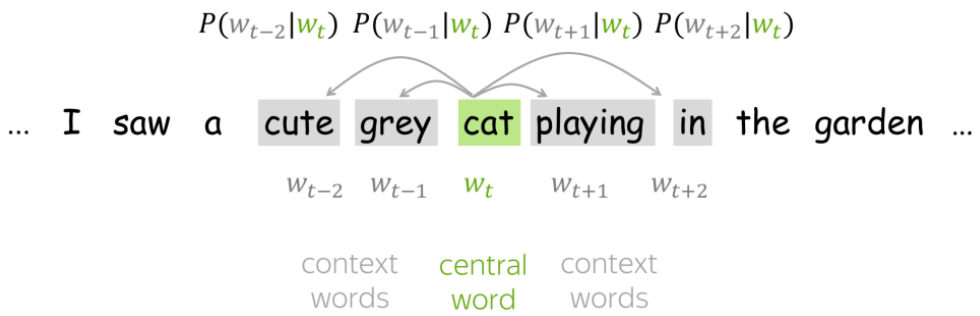
Skip-gram

The idea: we want to use words to predict their context words.



Skip-gram

The idea: we want to use words to predict their context words.



How do we calculate the probabilities $P(w_{t+j}|w_t, \theta)$?

- We have two sets of vectors for each word in the vocabulary

  $\mathbf{v}_c \in \mathbb{R}^d$: *embedding for target word* c

  $\mathbf{u}_o \in \mathbb{R}^d$: *embedding for context word* o

- Use dot product to measure the probability of context word o given center word c:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product: measures similarity of o and c
Larger dot product = larger probability

Normalize over entire vocabulary
to get probability distribution

How do we calculate the probabilities $P(w_{t+j}|w_t, \theta)$?

- We have two sets of vectors for each word in the vocabulary

  $\mathbf{v}_c \in \mathbb{R}^d$: *embedding for target word* c

  $\mathbf{u}_o \in \mathbb{R}^d$: *embedding for context word* o

- Use dot product to measure the probability of context word o given center word c:

This is the softmax function!

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product: measures similarity of o and c
Larger dot product = larger probability

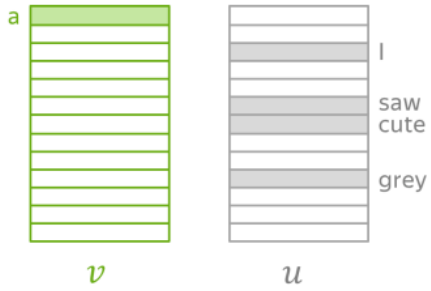
Normalize over entire vocabulary
to get probability distribution

Back to our example

$P(u_I|v_a)$ $P(u_{saw}|v_a)$ $P(u_{cute}|v_a)$ $P(u_{grey}|v_a)$

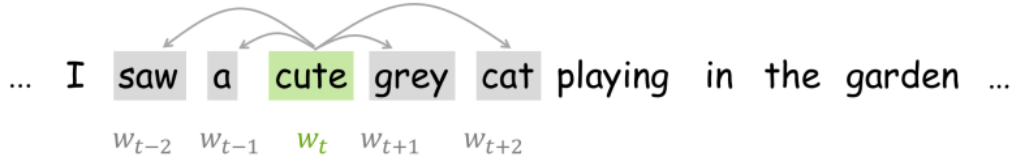
... I saw a cute grey cat playing in the garden ...

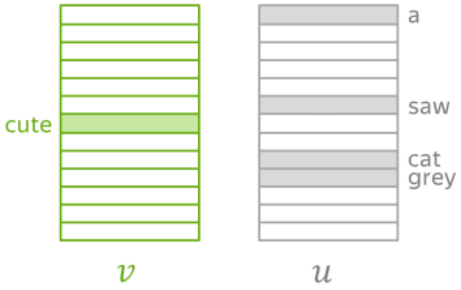
w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}



Back to our example

$P(u_{saw}|v_{cute})$ $P(u_a|v_{cute})$ $P(u_{grey}|v_{cute})$ $P(u_{cat}|v_{cute})$



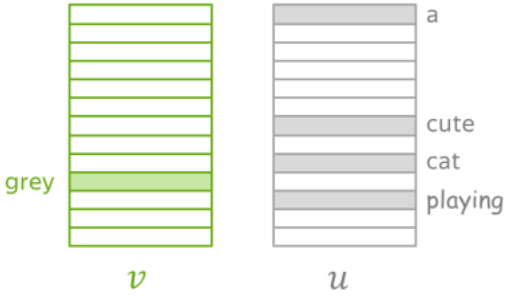


Back to our example

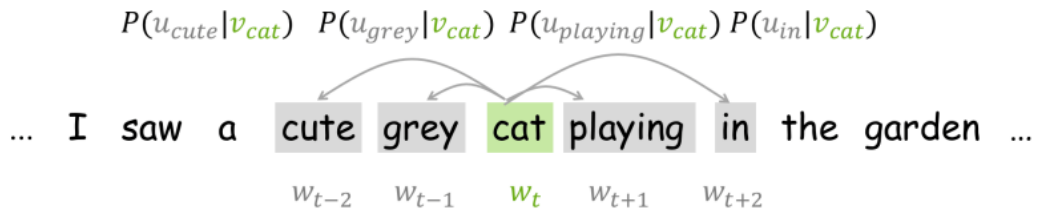
$P(u_a|v_{grey})$ $P(u_{cute}|v_{grey})$ $P(u_{cat}|v_{grey})$ $P(u_{playing}|v_{grey})$

… I saw a cute grey cat playing in the garden …

w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}



Back to our example

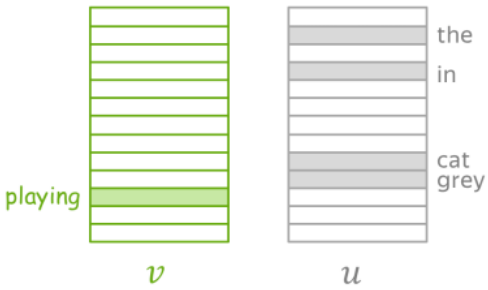


Back to our example

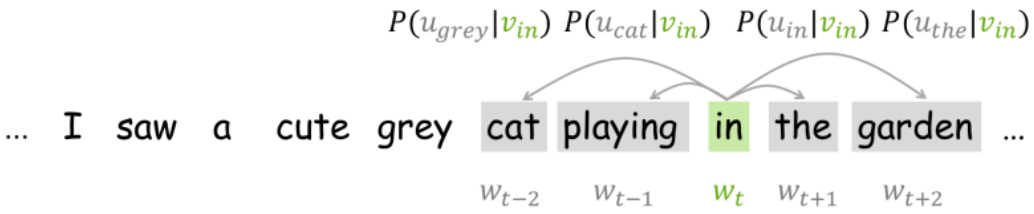
$P(u_{grey}|v_{playing})$ $P(u_{cat}|v_{playing})$ $P(u_{in}|v_{playing})$ $P(u_{the}|v_{playing})$

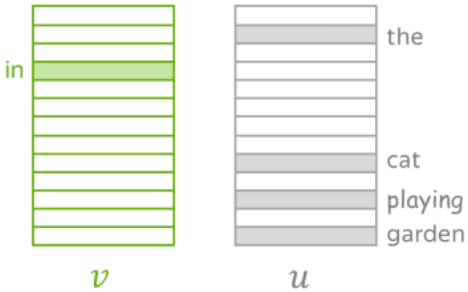
… I saw a cute grey cat playing in the garden …

w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}



Back to our example





How do we train the Skip-gram model?

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \neq 0} P(w_{t+j} | w_t, \theta),$$

$$\text{Loss} = J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \neq 0}} \log P(w_{t+j} | w_t, \theta)$$

agrees with our plan above ⟼ go over text

with a sliding window

compute probability of the context word given the central

T is number of words in the training corpus.

How do we train the Skip-gram model?

We rely on gradient descent (recall the lecture on logistic regression).

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta).$$

In practice, we optimize **one word at a time**:

$$-\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P(w_{t+j}|w_t,\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} J_{t,j}(\theta)$$

How do we train the Skip-gram model?

$$J_{t,j}(\theta) \quad = \quad -\log P(cute|cat) \quad =$$

Note which parameters are present at this step:

- from vectors for central words, only v_{cat}
- from vectors for context words, all u_w for all words in the vocabulary.

How do we train the Skip-gram model?

$$J_{t,j}(\theta) = -\log P(cute|cat) = -\log\left(\frac{\exp u_{cute}^T v_{cat}}{\sum_{w \in Voc} \exp u_w^T v_{cat}}\right)$$

Note which parameters are present at this step:

- from vectors for central words, only v_{cat}
- from vectors for context words, all u_w for all words in the vocabulary.

How do we train the Skip-gram model?

$$J_{t,j}(\theta) = -\log P(cute|cat) = -\log\left(\frac{\exp u_{cute}^T v_{cat}}{\sum_{w \in Voc} \exp u_w^T v_{cat}}\right)$$

$$-\left(\log \exp u_{cute}^T v_{cat} - \log \sum_{w \in Voc} \exp u_w^T v_{cat}\right)$$

Note which parameters are present at this step:

- from vectors for central words, only v_{cat}
- from vectors for context words, all u_w for all words in the vocabulary.

How do we train the Skip-gram model?

$$J_{t,j}(\theta) = -\log P(cute|cat) = -\log\left(\frac{\exp u_{cute}^T v_{cat}}{\sum_{w \in Voc} \exp u_w^T v_{cat}}\right)$$

$$-\left(\log \exp u_{cute}^T v_{cat} - \log \sum_{w \in Voc} \exp u_w^T v_{cat}\right)$$

Since $\log \exp(x) = x$

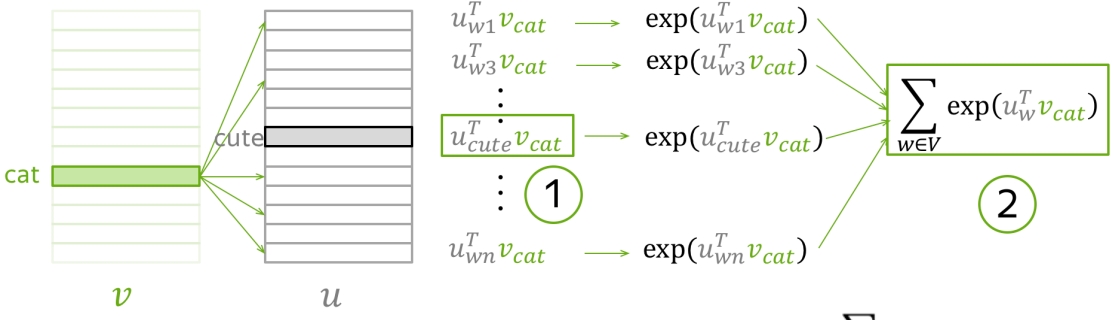
$$-u_{cute}^T v_{cat} + \log \sum_{w \in Voc} \exp u_w^T v_{cat}$$

Note which parameters are present at this step:

- from vectors for central words, only v_{cat}
- from vectors for context words, all u_w for all words in the vocabulary.

How do we train the Skip-gram model?

1. Take dot product of v_{cat} with **all** u 2. exp 3. sum all



$$J_{t,j}(\theta) = \underbrace{-u_{cute}^T v_{cat}}_{1} + \underbrace{\log \sum_{w \in V} \exp(u_w^T v_{cat})}_{2}$$

How do we train the Skip-gram model?

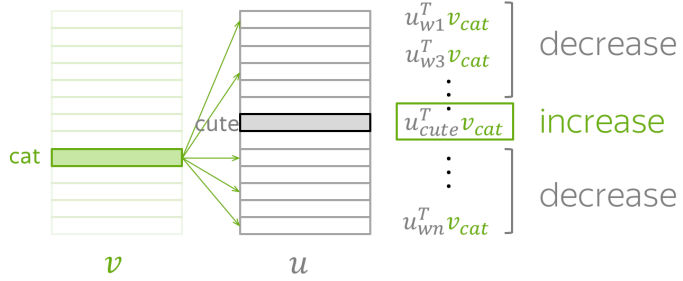
Get loss for this one step

$$J_{t,j}(\theta) = \underbrace{-u_{cute}^T v_{cat}}_{1} + \underbrace{\log \sum_{w \in V} \exp(u_w^T v_{cat})}_{2}$$

evaluate the gradient, make an update

$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$

$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \quad \forall w \in V$$

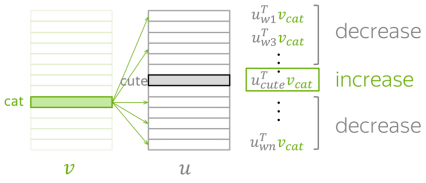


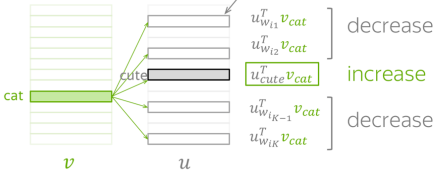
Negative Sampling: Making learning more efficient

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **all other** u - decrease

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **a subset of other** u - decrease

Negative samples: randomly selected K words





Parameters to be updated:
- v_{cat}
- u_w for all w in the vocabulary

|V| + 1 vectors

Parameters to be updated:
- v_{cat}
- u_{cute} and u_w for w in K negative examples

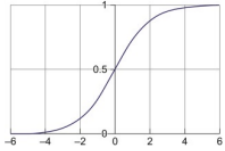
K + 2 vectors

Negative Sampling: Making learning more efficient

$$J_{t,j}(\theta) = -\log \sigma(u_{cute}^T v_{cat}) - \sum_{w \in \{w_{i_1}, \ldots, w_{i_K}\}} \log(1 - \sigma(u_w^T v_{cat}))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

the logistic sigmoid function



We have now converted negative sampline to binary classification:
predict "+" for (central, real context) pairs, and "−" for (central, random context).

Negative Sampling: Making learning more efficient

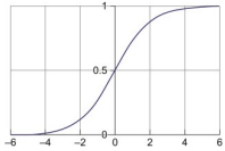
Positive Examples

Negative Examples

$$J_{t,j}(\theta) = -\log \sigma(u_{cute}^T v_{cat}) - \sum_{w \in \{w_{i_1}, \ldots, w_{i_K}\}} \log(1 - \sigma(u_w^T v_{cat}))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

the logistic sigmoid function



We have now converted negative sampline to binary classification:
predict "+" for (central, real context) pairs, and "−" for (central, random context).

How do we select negative samples?

The basic idea is to select random words based on their (unigram) frequency in a corpus. But can we do better?

- “Flatten” the unigram distribution to make sure infrequent words get sampled (recall Zipf’s distribution)
- Don’t generate compatible contexts
- Make sure you generate “hard” negative examples, to learn informative word representations

What happens when training has finished, what are the embeddings?

How do we select negative samples?

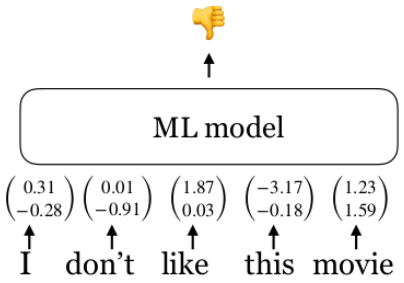
The basic idea is to select random words based on their (unigram) frequency in a corpus. But can we do better?

- "Flatten" the unigram distribution to make sure infrequent words get sampled (recall Zipf's distribution)
- Don't generate compatible contexts
- Make sure you generate "hard" negative examples, to learn informative word representations

What happens when training has finished, what are the embeddings?
We learn two embeddings per word, $\mathbf{v}_c$, $\mathbf{u}_o$, we can just add them or throw away the context embeddings $\mathbf{u}_o$.

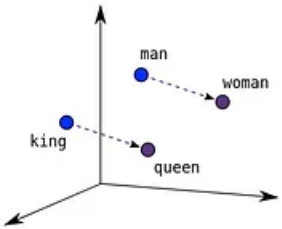
Evaluation

Extrinsic Evaluation: Let's plug these word embeddings into a real NLP system and see whether this improves performance. Could take a long time but still the most important evaluation metric

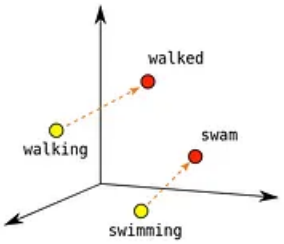


Intrinsic Evaluation: Evaluate on a specific/intermediate subtask which is fast to compute and not clear if it really helps the downstream task.

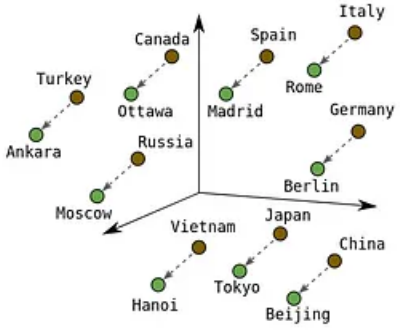
Intrinsic Evaluation: Word Analogy



Male-Female



Verb Tense



Country-Capital

semantic: $v(king) - v(man) + v(woman) \approx v(queen)$
syntactic: $v(walking) - v(swimming) + v(swam) \approx v(walked)$

More examples at `http://download.tensorflow.org/data/questions-words.txt`

Summary for today

- Neural embeddings can be efficiently learned from large collections of unannotated texts ('self-supervision')
- Many algorithms, and important hyperparameter choices (windows sizes, numbers of negatives samples)
- Useful in practice and have some intriguing properties
- Preferable over raw count-based method but:
  - *how do we handle multiple senses? (ambiguity)*
  - *how do we encode longer spans of text? (compositionality)*

Check out Lena Voita's online resource – NLP class for you:
`https://lena-voita.github.io/nlp_course.html`