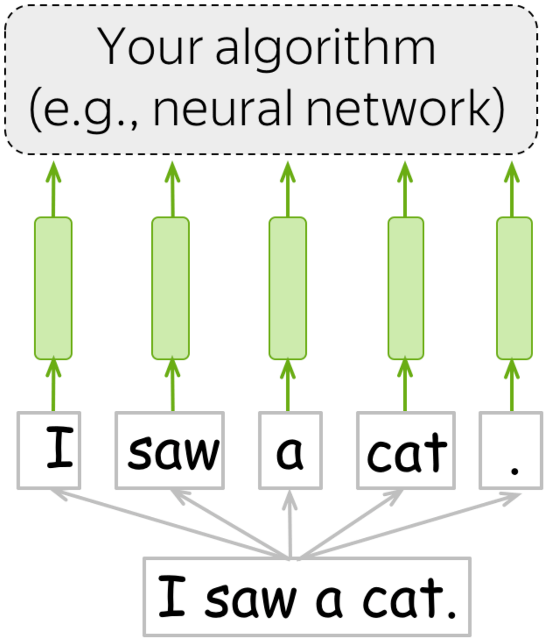
Foundations for Natural Language Processing

Neural Embeddings

Ivan Titov
(with graphics/materials from Elena Voita)

School of informatics

Neural models and word embeddings



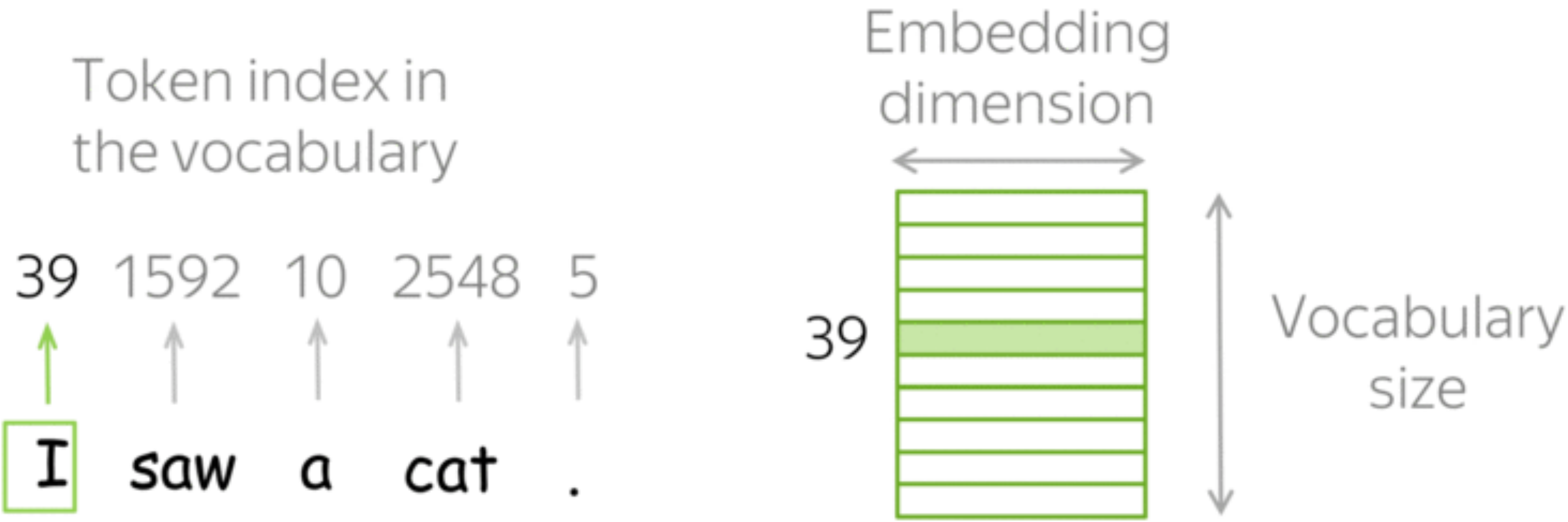
Any algorithm for solving a task

Word representation - vector
(input for your model/algorithm)

Sequence of tokens

Text (your input)

Neural models and word embeddings



One-hot vectors as word representations

One is 1, the rest are 0

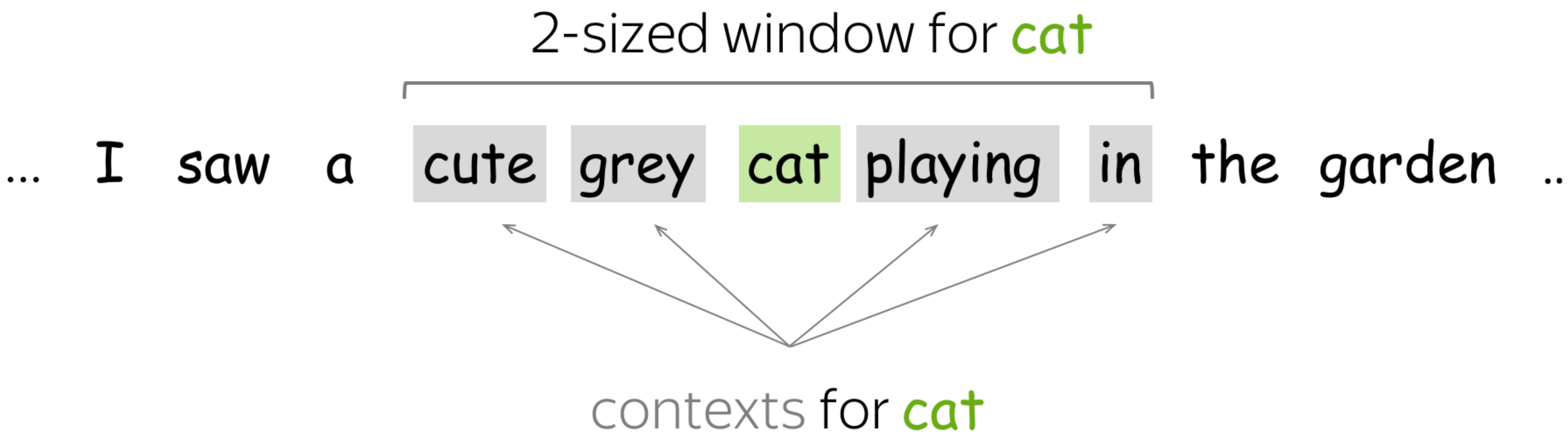
dog	0...0...010....0...0
cat	0...010...0....0...0
table	0...0...0....0010...

Embedding dimension = vocabulary size

Issues:

- very high dimensional
- do not capture semantic similarity between words (recall last lecture)

Recap: latent semantic analysis



Context:

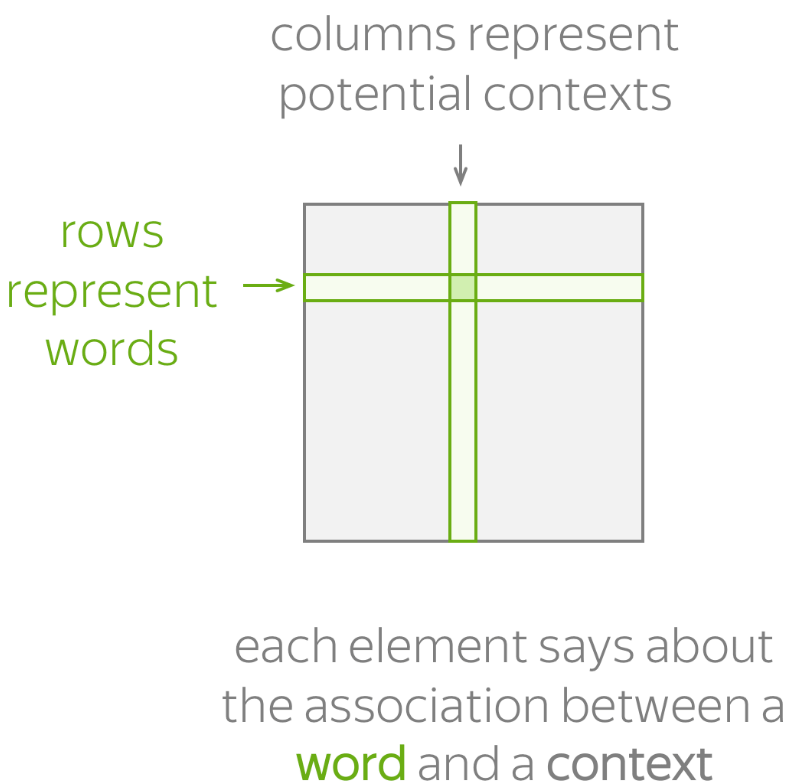
- surrounding words in a L-sized window

Matrix element:

- N(w, c) – number of times word w appears in context c

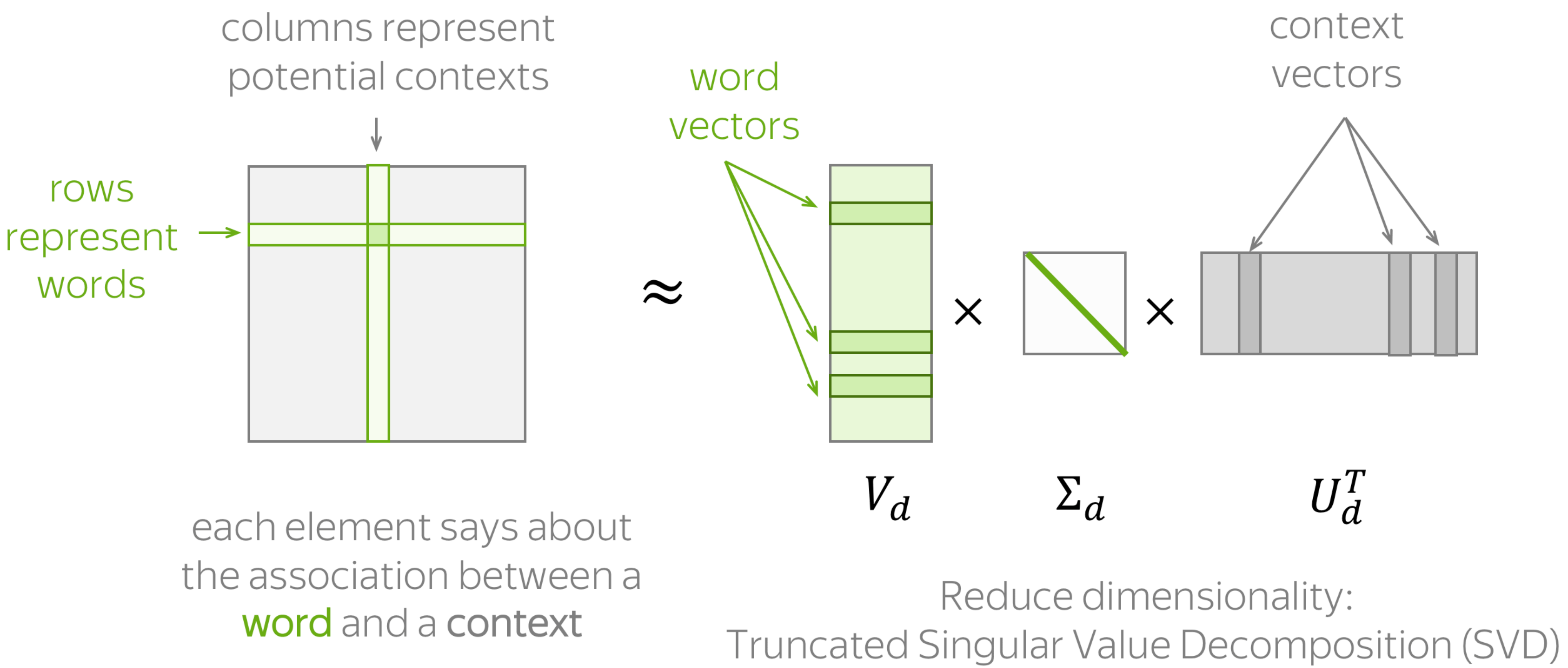
Recall: to make it work reasonably well, you need something more sophisticated (e.g., PMI)

Latent Semantic Analysis



This is either the 'raw' co-occurrence matrix N, or its transformations (e.g., PMI)

Latent Semantic Analysis



Hard to scale, any alternative?

Prediction-based (aka neural) methods

$P(w_{t-2}|w_t)$ $P(w_{t-1}|w_t)$ $P(w_{t+1}|w_t)$ $P(w_{t+2}|w_t)$

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

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

context words | central word | context words

Prediction-based (aka neural) methods

$P(w_{t-2}|w_t)$ $P(w_{t-1}|w_t)$ $P(w_{t+1}|w_t)$ $P(w_{t+2}|w_t)$

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

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

context words | central word | context words

Prediction-based (aka neural) methods

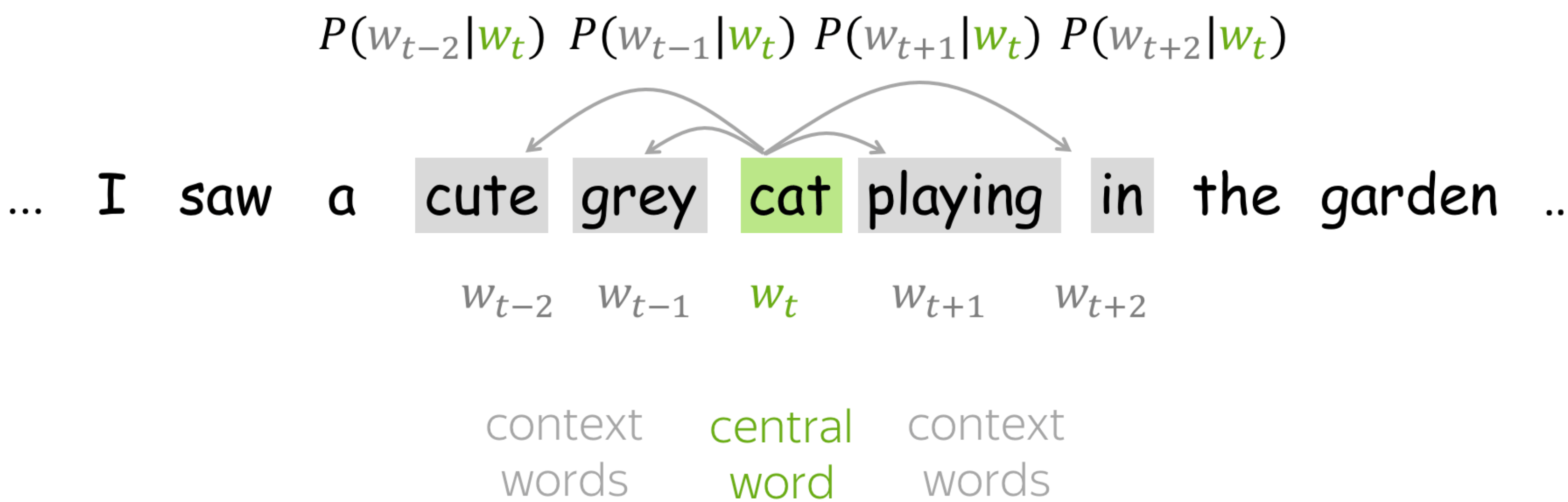
$P(w_{t-2}|w_t)$ $P(w_{t-1}|w_t)$ $P(w_{t+1}|w_t)$ $P(w_{t+2}|w_t)$

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

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

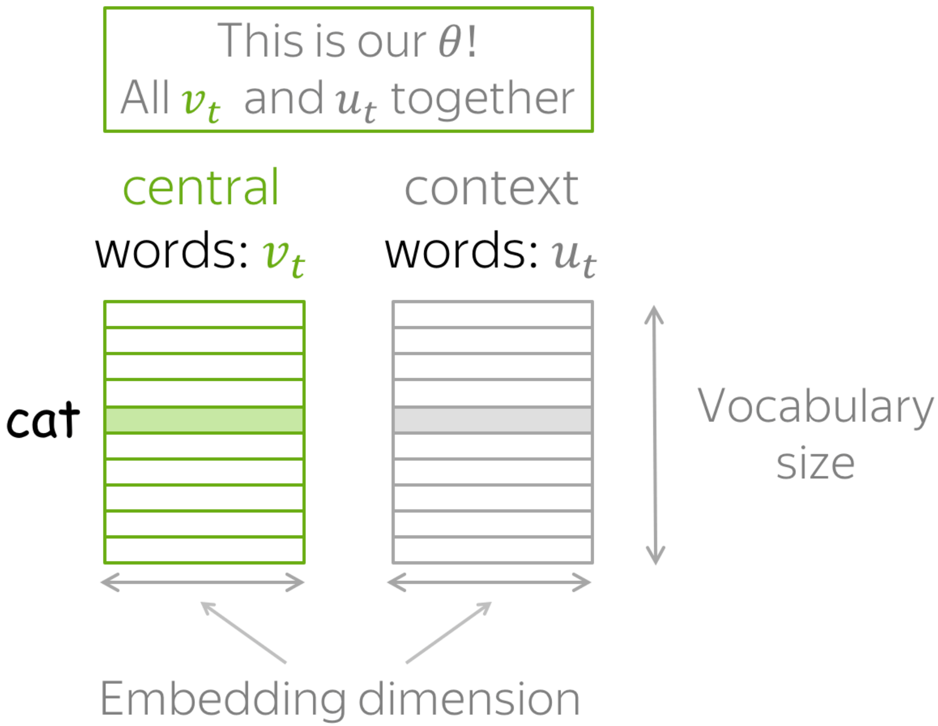
context words | central word | context words

Prediction-based (aka neural) methods



What we discuss is Mikolov's Skipgram but there several variations on this idea

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



For each word w we have two vectors:

- when it is a central word
- when it is a context word

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

The probability of the context word o given the central word c is

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

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

Normalize over entire vocabulary
to get probability distribution

This is the softmax function

Back to our example

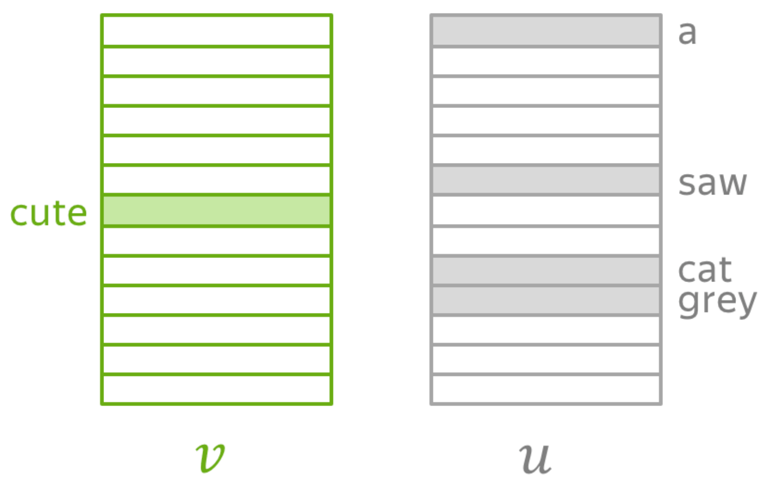
$P(u_I|v_a)$ $P(u_{saw}|v_a)$ $P(u_{cute}|v_a)$ $P(u_{grey}|v_a)$

Back to our example

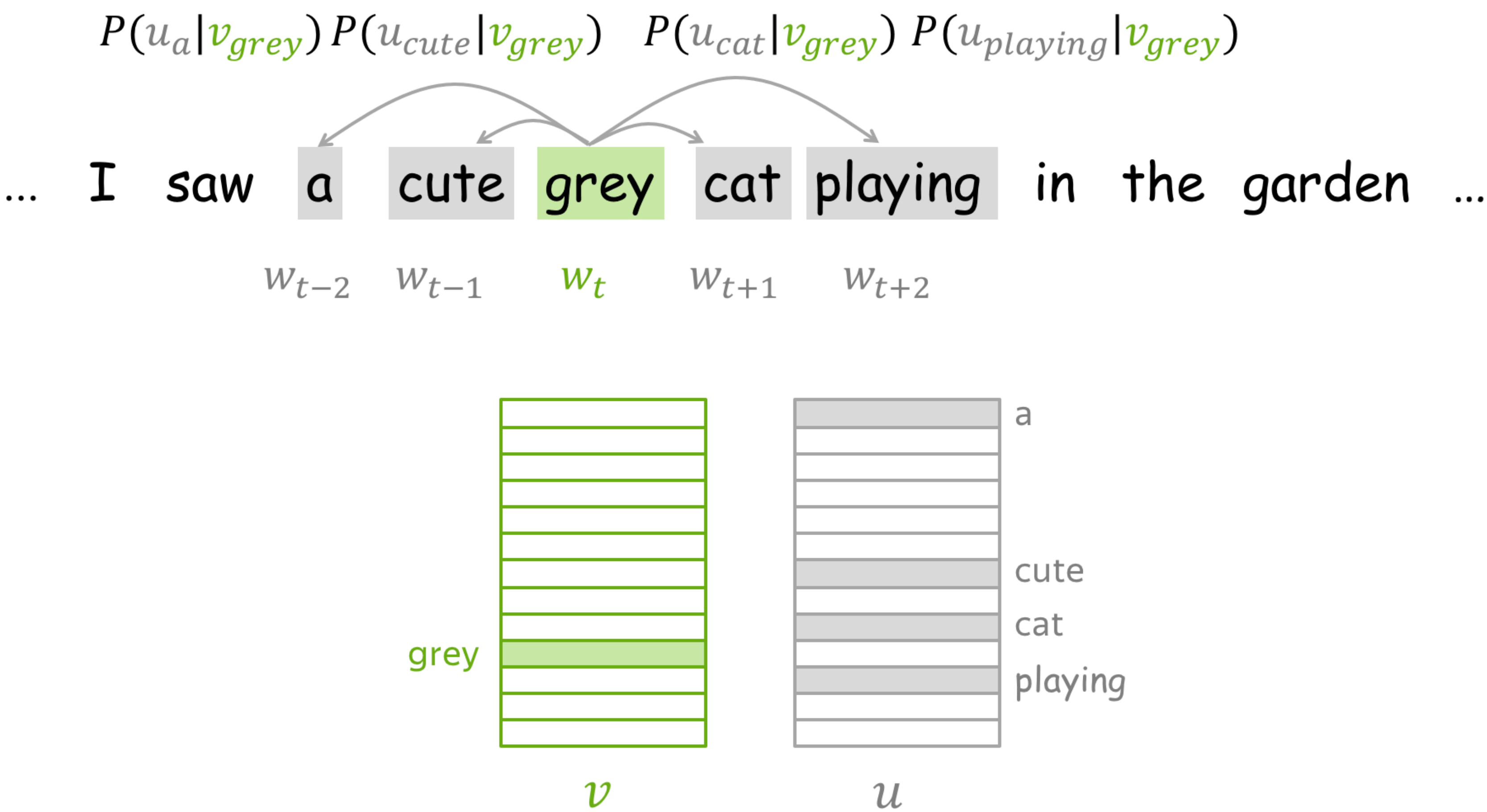
$P(u_{saw}|v_{cute})$ $P(u_{a}|v_{cute})$ $P(u_{grey}|v_{cute})$ $P(u_{cat}|v_{cute})$

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

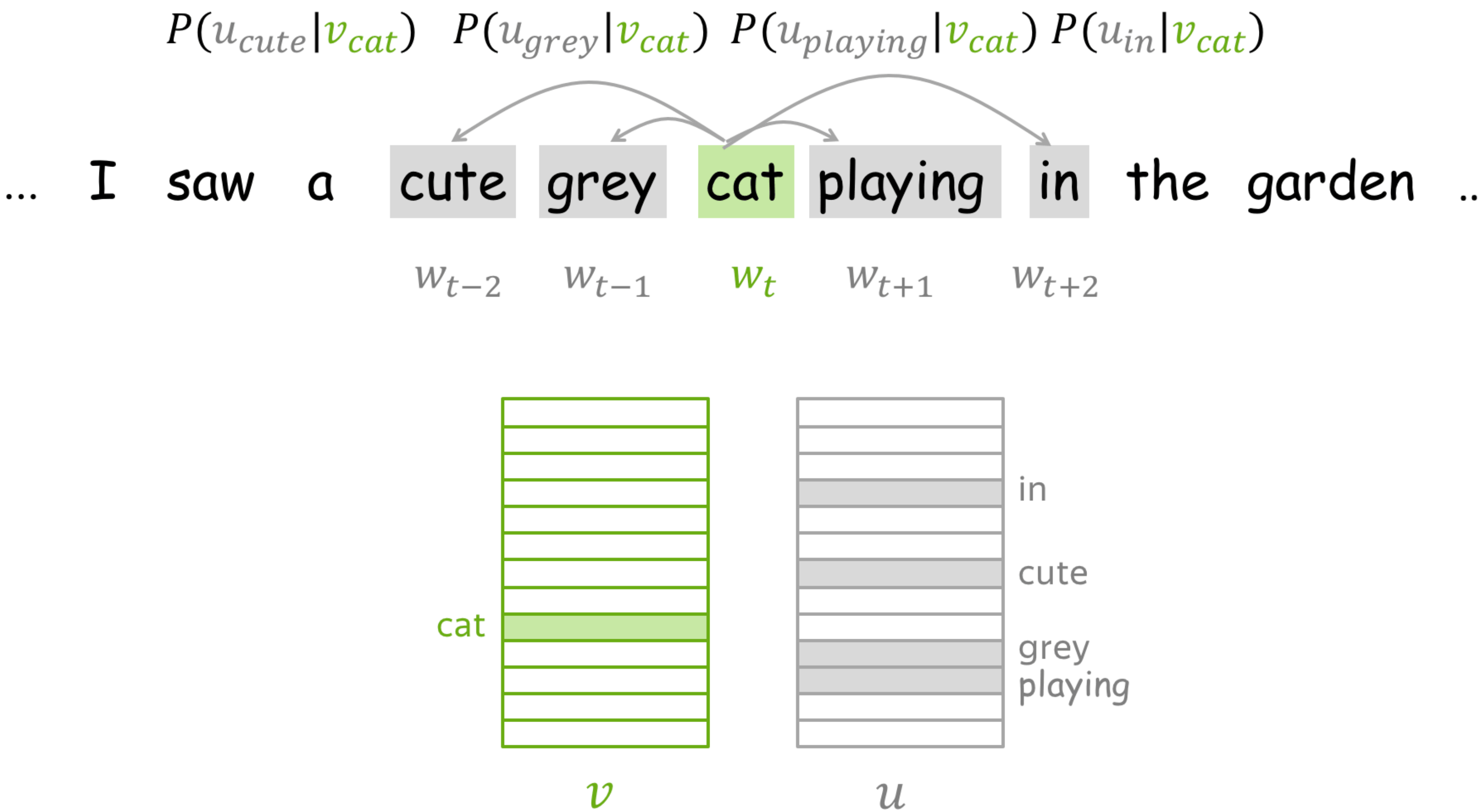
w_{t-2} w_{t-1} w_{t} w_{t+1} w_{t+2}



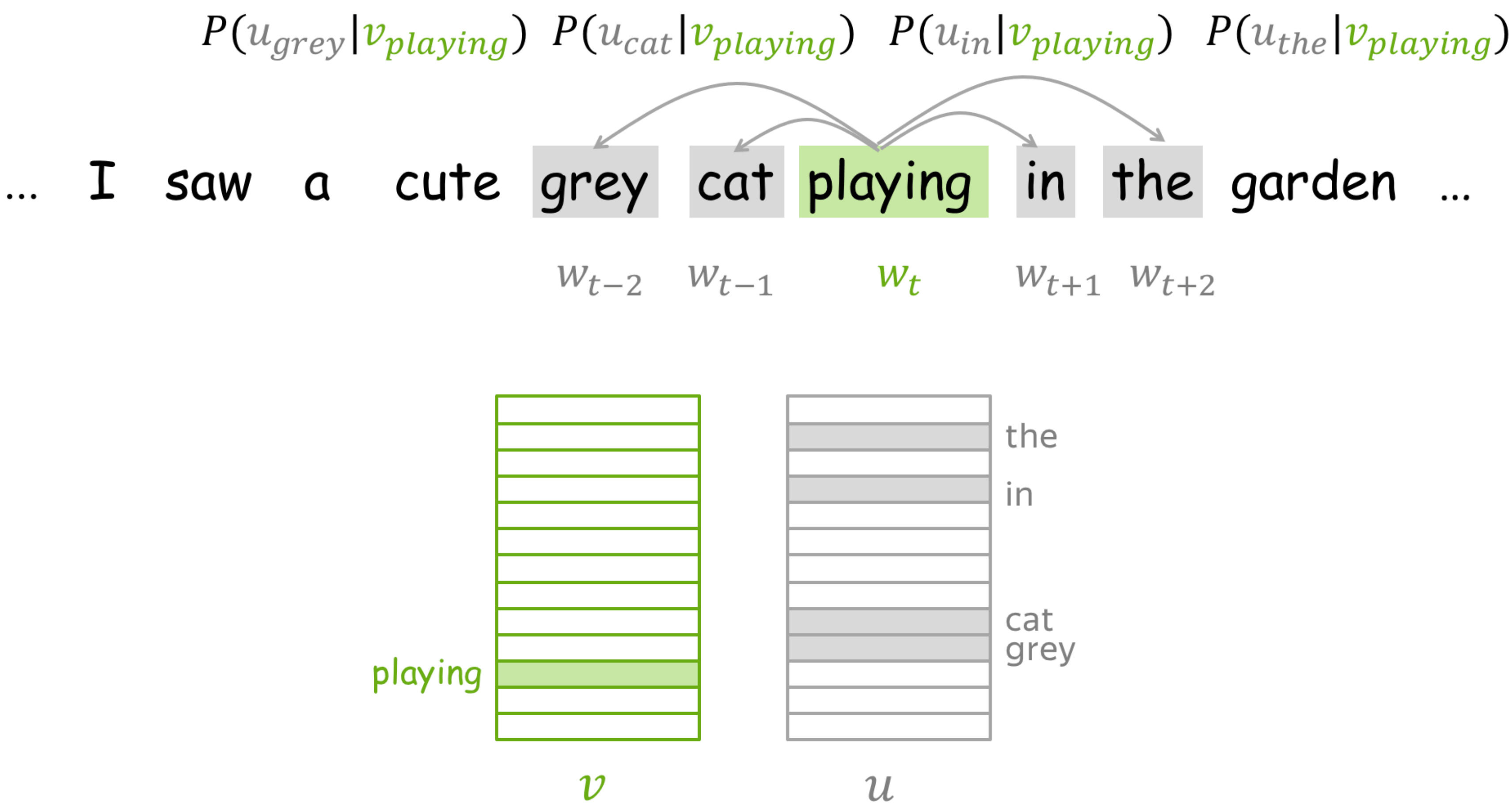
Back to our example



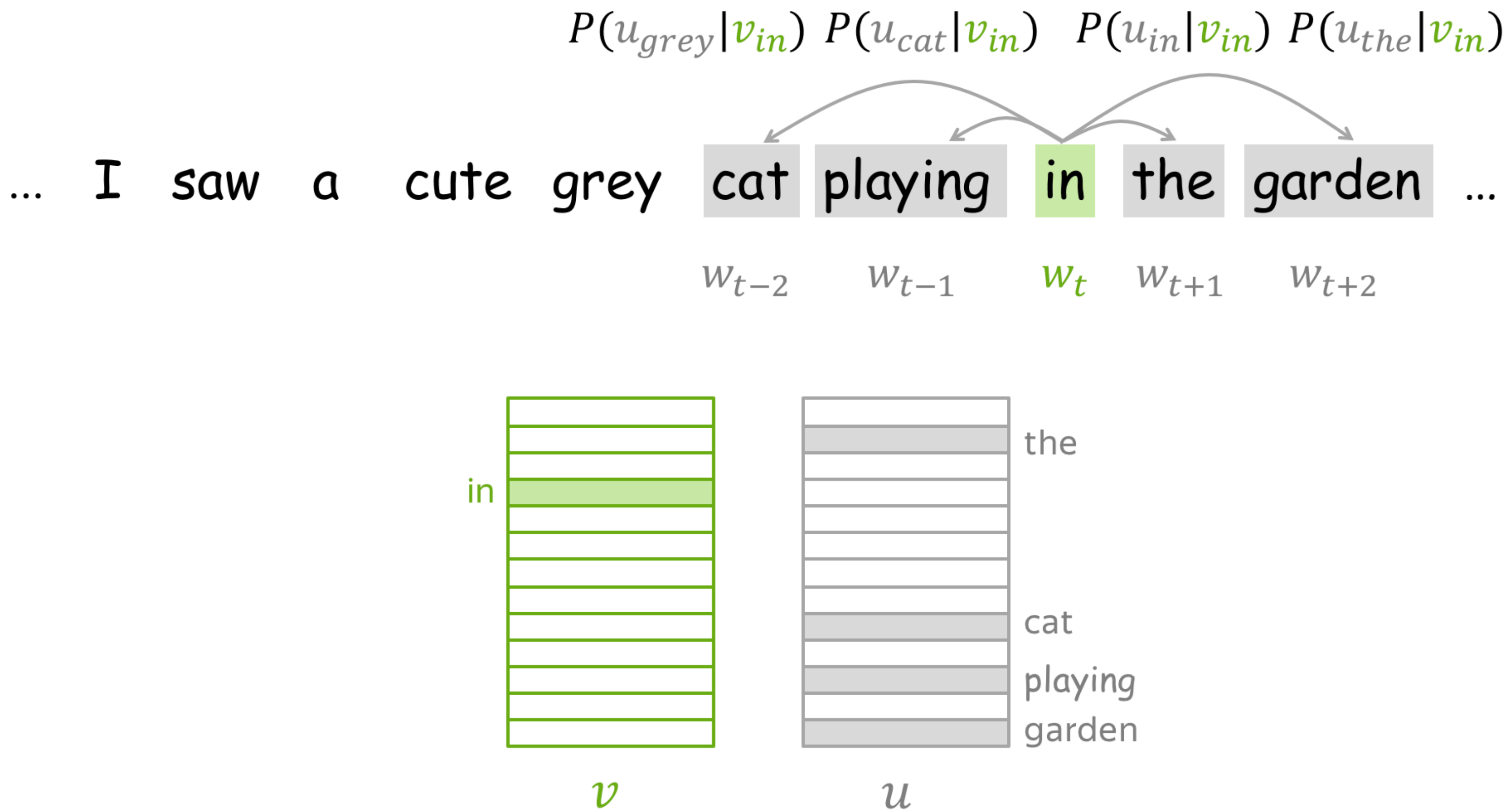
Back to our example



Back to our example



Back to our example



What do we optimize?

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

agrees with our plan above $\longmapsto$ go over text

with a sliding window

compute probability of the context word given the central

How do we optimize the model?

We rely on gradient descent (recall the Logistic Regression lecture)

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

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 optimize the model?

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

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

How do we optimize the model?

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

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

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

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 optimize the model?

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

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

make an update

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

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

How do we optimize the model?

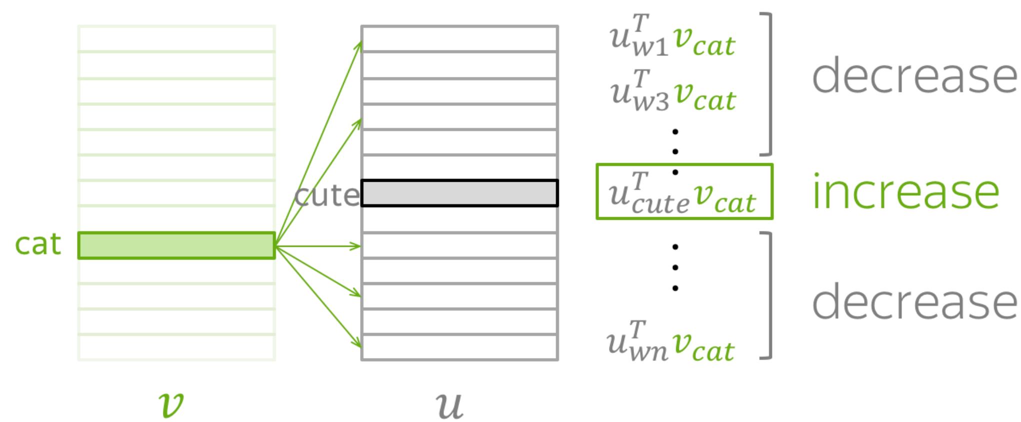
… I saw a cute grey cat playing in the garden …

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

make an update

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

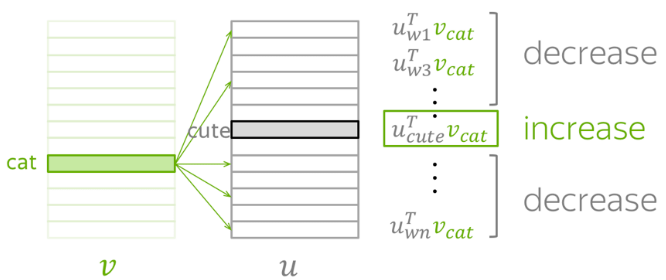
$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \; \forall\, w \in V$$



Making learning more efficient

Dot product of v_{cat}:

- with u_{cute} - increase,
- with <u>all other</u> u - decrease



Parameters to be updated:

- v_{cat}
- u_w for all w in the vocabulary

|V| + 1 vectors

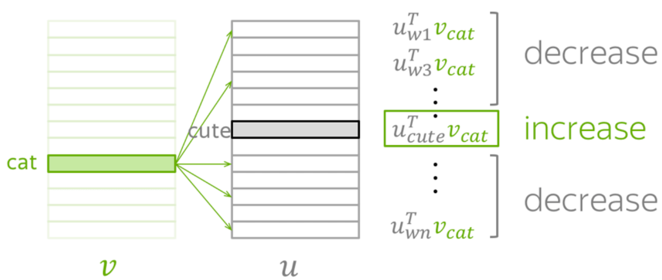
Making learning more efficient

Dot product of v_{cat}:

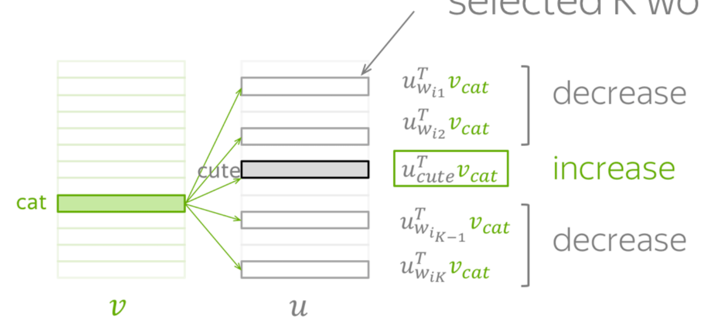
- with u_{cute} - increase,
- with **all other** u - decrease

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

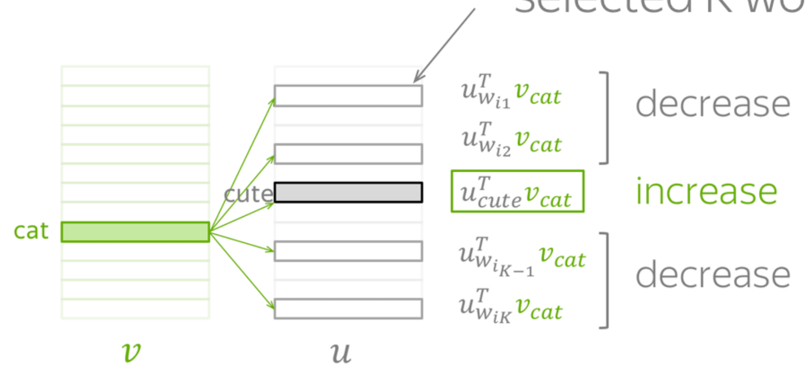
|V| + 1 vectors

Parameters to be updated:

- v_{cat}
- u_{cute} and u_w for w in K negative examples

K + 2 vectors

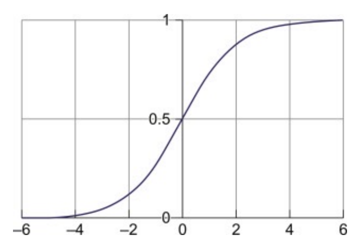
Negative sampling



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

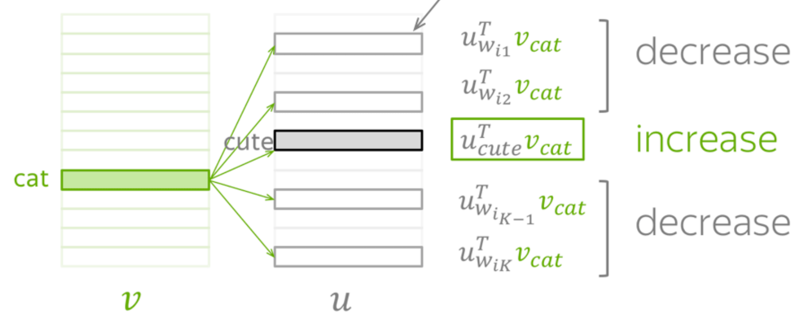
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

the logistic sigmoid function



Negative sampling

Negative samples: randomly selected K words



Converted to binary classification: predict "+" for (central, real context) pairs, and "-" for (central, random context)

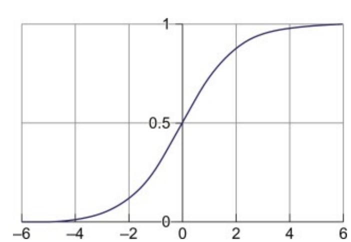
"Positive examples"

"Negative examples"

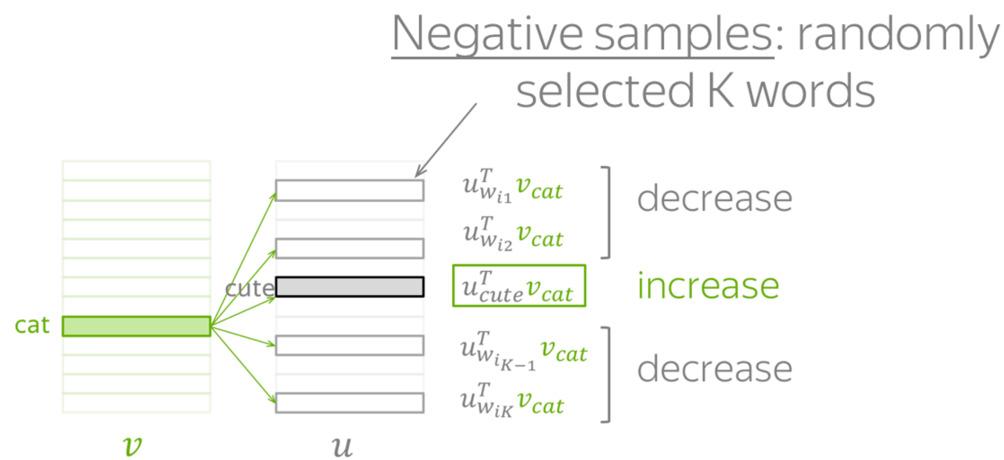
$$J_{t,j}(\theta) = -\log \sigma(u_{cute}^T v_{cat}) - \sum_{w \in \{w_{i_1}, \ldots, w_{i_K}\}} \log(1 - \sigma(u_w^T v_{cat}))$$

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

the logistic sigmoid function

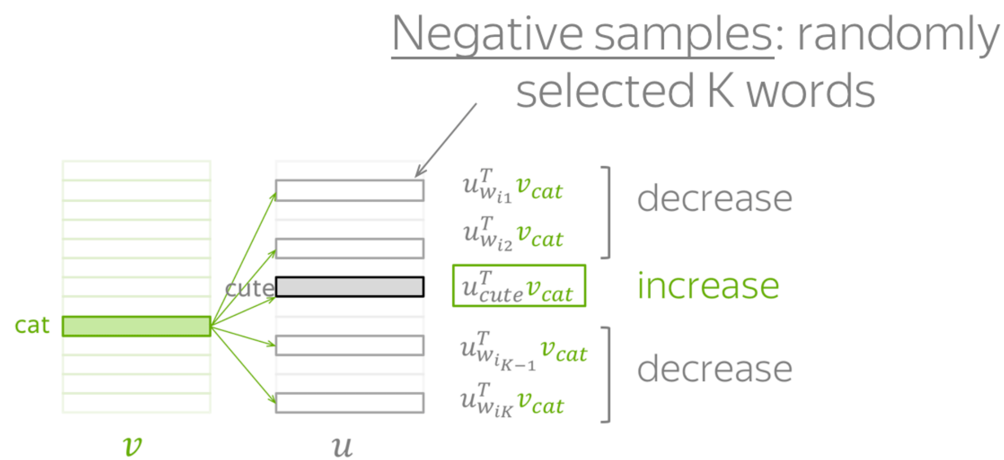


How to select negative samples?



The basic idea is to select random words based on their (unigram) frequency in a corpus, what are the issues and how can this be improved?

How to select negative samples?

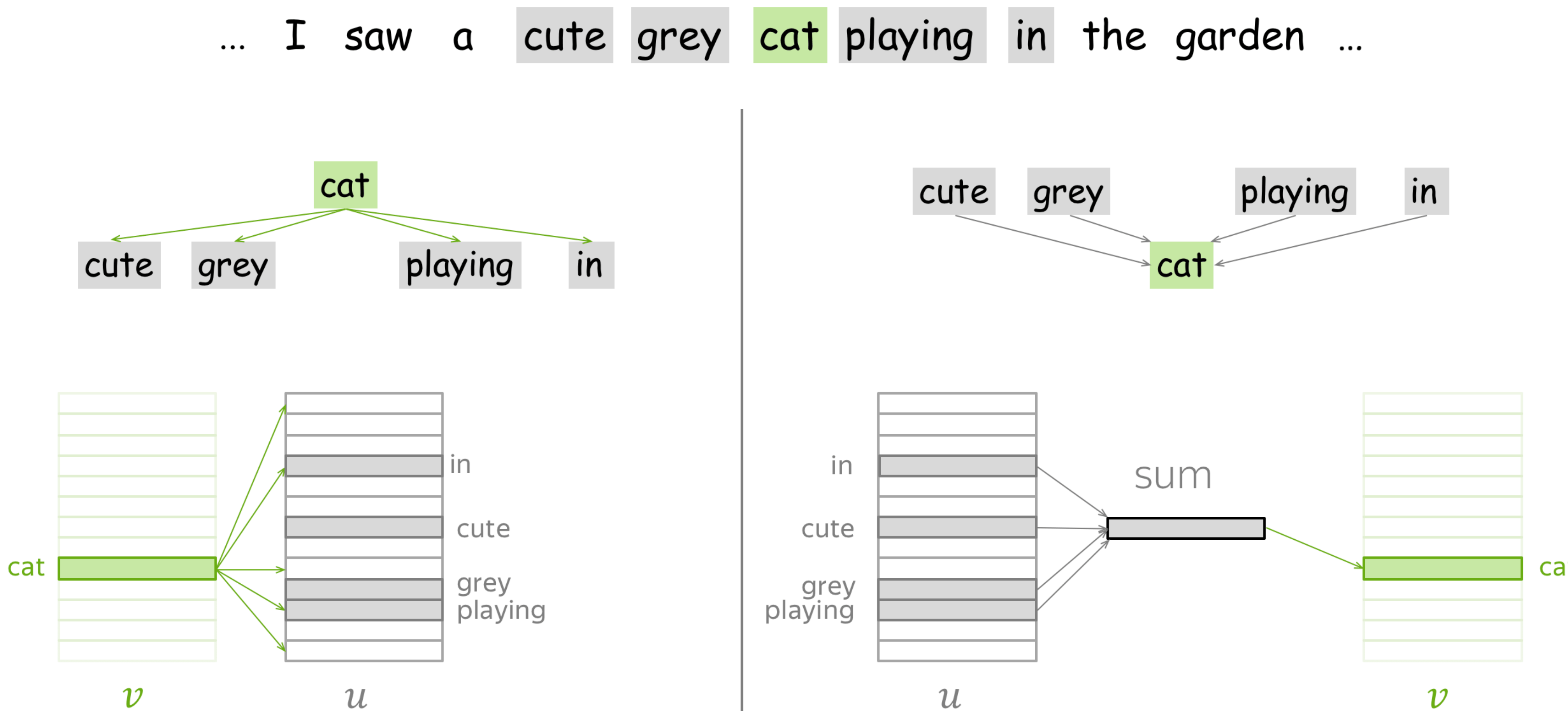


The basic idea is to select random words based on their (unigram) frequency in a corpus, what are the issues and how can this be improved?

- "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

Negative sampling is an important technique used in many contexts in NLP/ML, often with these mods

Variations of Word2Vec



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

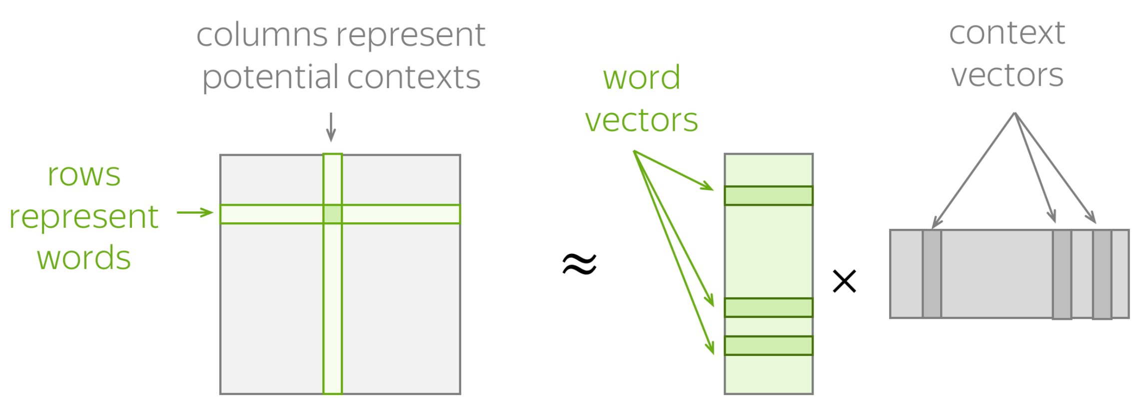
CBOW: from sum of context predict central

Generally, word2vec was far from the first idea of learning embedding, its success is largely due to simplicity (+ the efficient implementation and stability)

Relation to LSA

It is possible to show that optimizing the skipgram objective (with the negative sampling modification) corresponds to factorizing PMI matrix (Levy & Goldberg 2014)

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



Evaluation (~ recap)

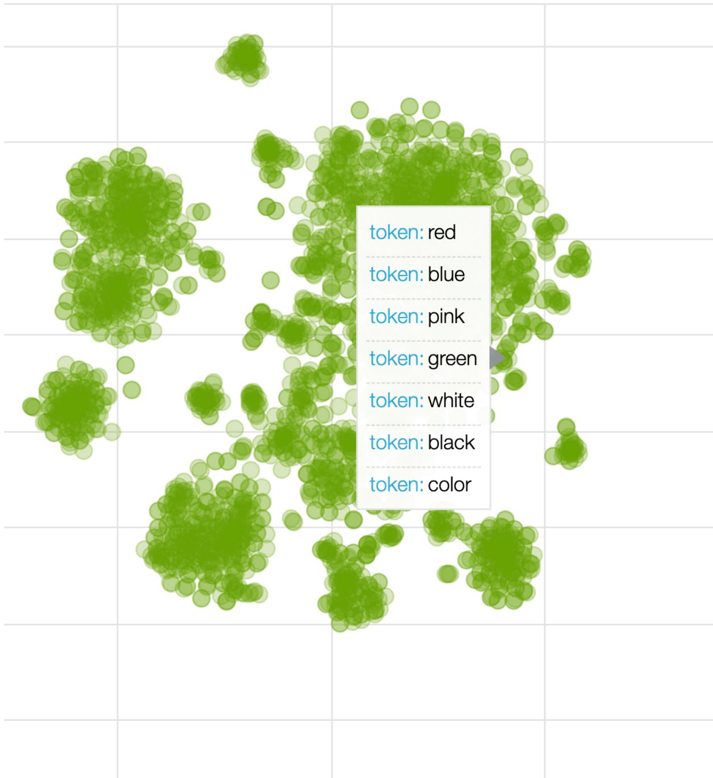
As we will see soon the low-dimensional vectors produced by neural methods are much 'friendlier' to downstream applications than sparse counts

Intrinsic

Use embeddings to represent tokens within models for downstream tasks (e.g., sentiment classification, question answering, …)

Extrinsic

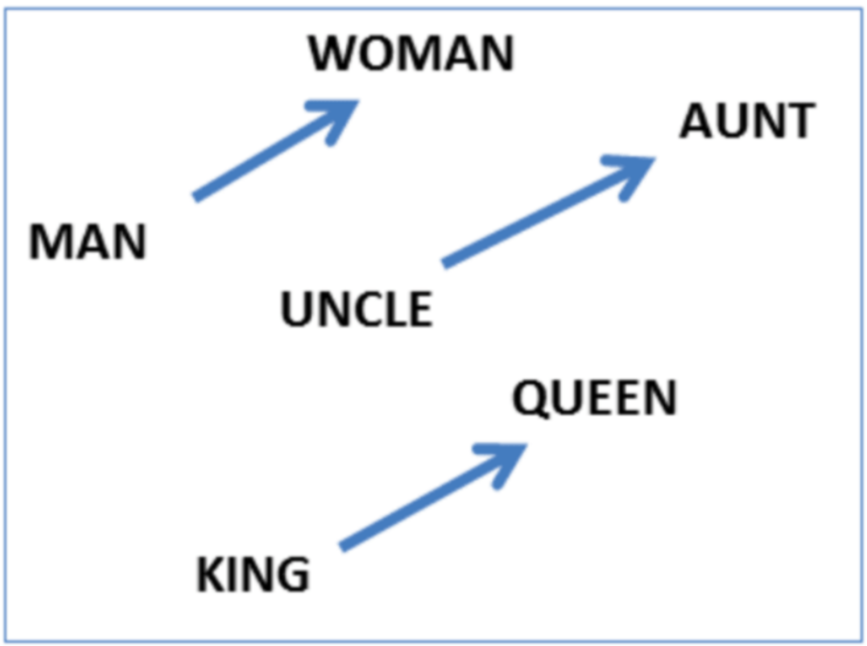
- Relatedness
- Word associations
- Analogy

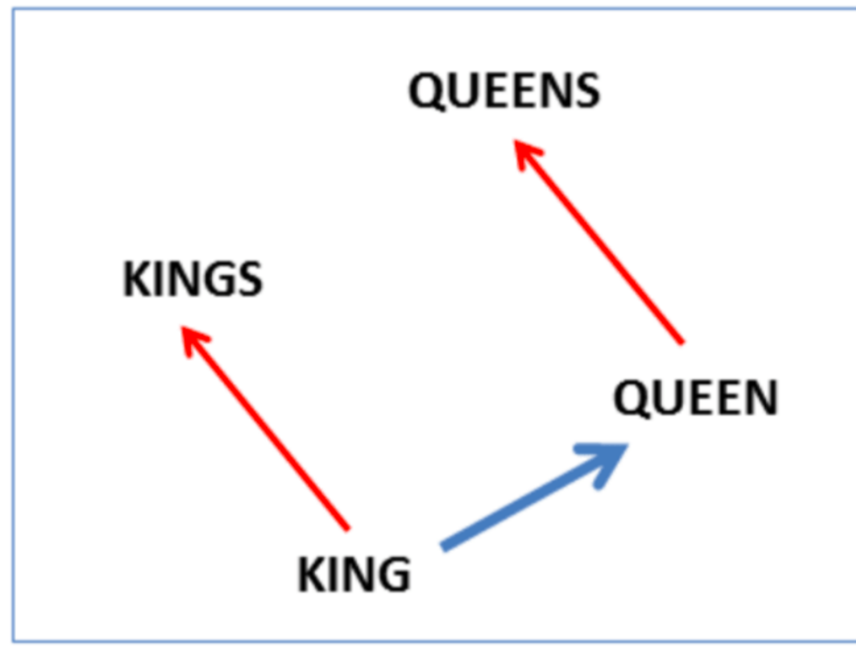


Analogy

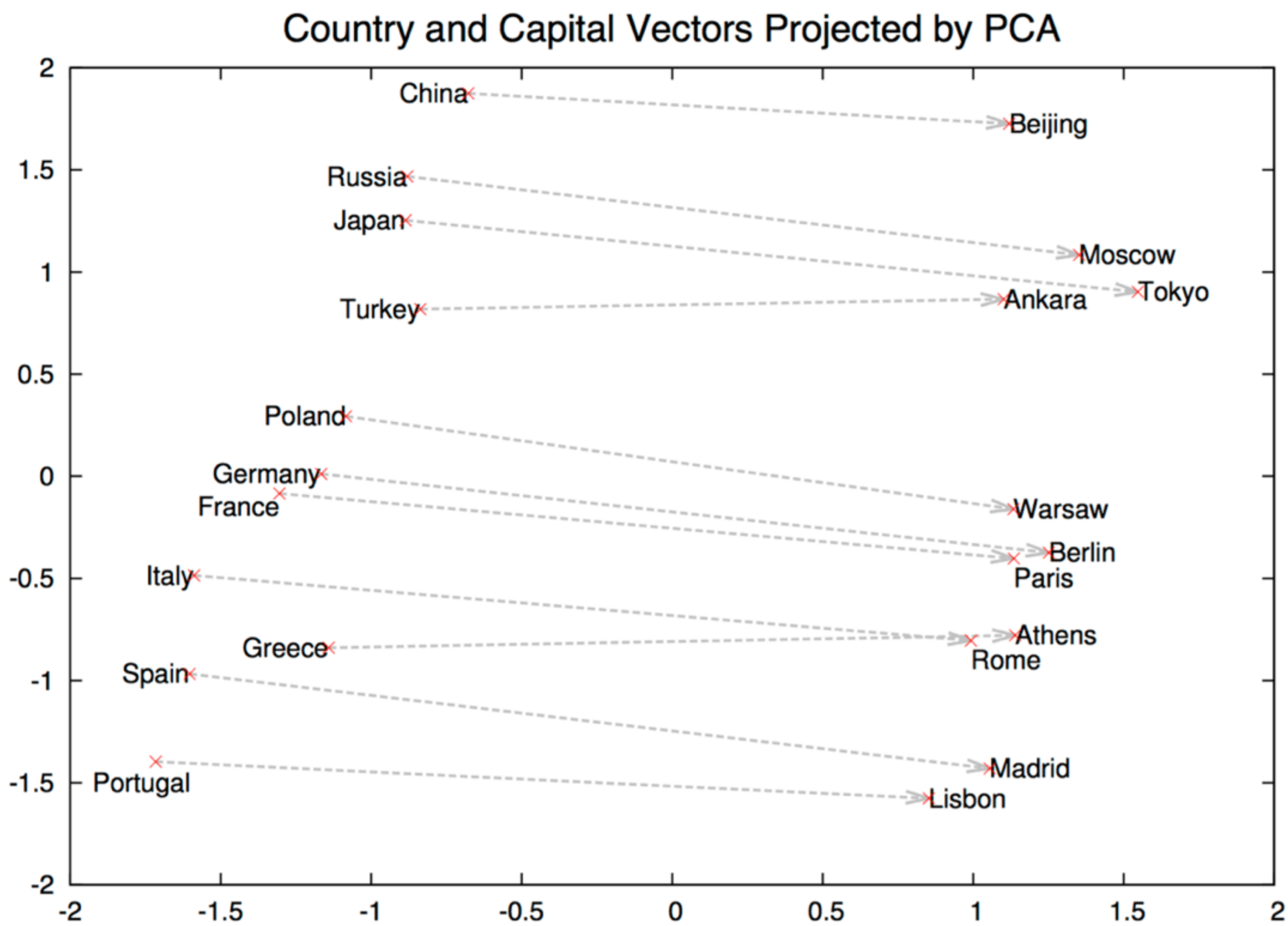
semantic: $v(\textbf{king}) - v(\textbf{man}) + v(\textbf{woman}) \approx v(\textbf{queen})$

syntactic: $v(\textbf{kings}) - v(\textbf{king}) + v(\textbf{queen}) \approx v(\textbf{queens})$

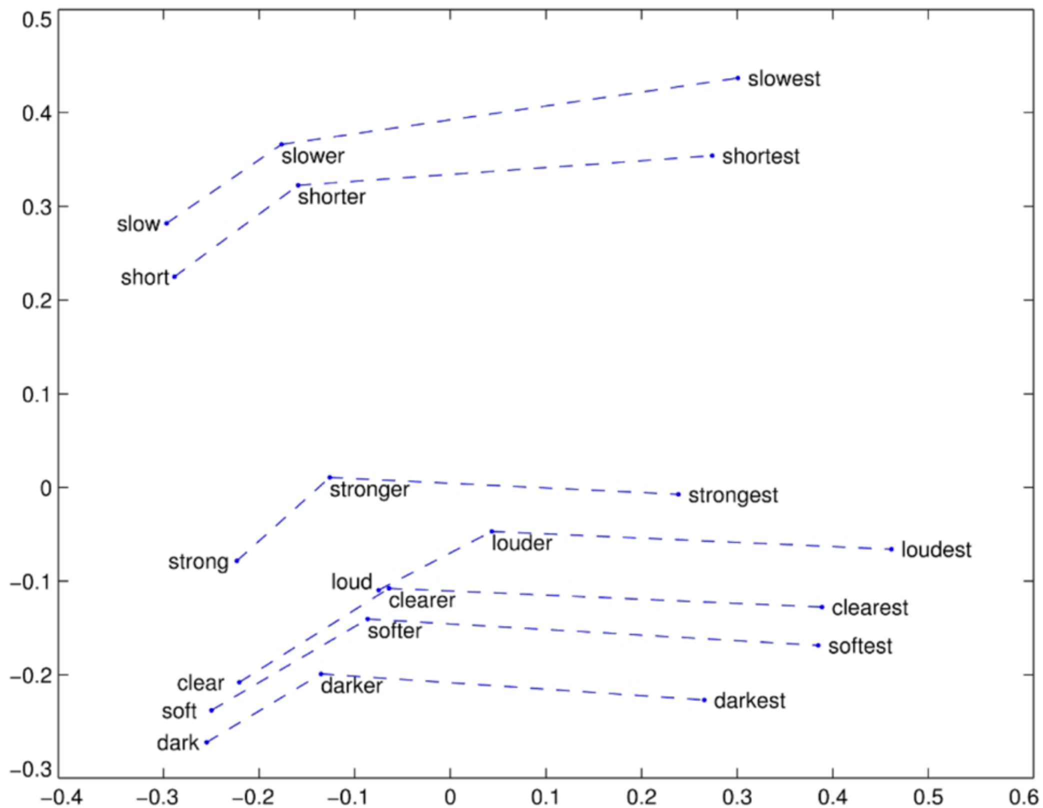




Analogy: semantic relations



Analogy: syntactic relations



Why makes word2vec represent relations in such linear way? See a paper by an Edinburgh student: [Analogies Explained: Towards Understanding Word Embeddings, Allen & Hospidales, ICML 2019]

There are some important caveats about the analogy benchmarks and visulaizations: [*https://aclanthology.org/N18-2039/*]

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