Distributed word representations

Lecture 3

Symbolic vs Distributional

- Any symbolic model (like one hot) lacks the relationship between words with related meaning
- No natural similarity relationship between words

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Look at neighbours

- We can learn a lot about the word's meaning by it's context
- "You shall know a word by the company it keeps"

J.R. Firth 1957, British Linguist

- Let's look at the context and understand what it means
- If you understand where the word fits, you know the meaning

…government debt problems turning into ***banking*** *crises as happened in 2009…*

…saying that Europe needs unified ***banking*** *regulation to replace the hodgepod*

…India has just given its ***banking*** *system a shot in the arm…*

Distributed vs Distributional

- **Distributed** representation: unlike one-hot, the meaning is around the whole vector

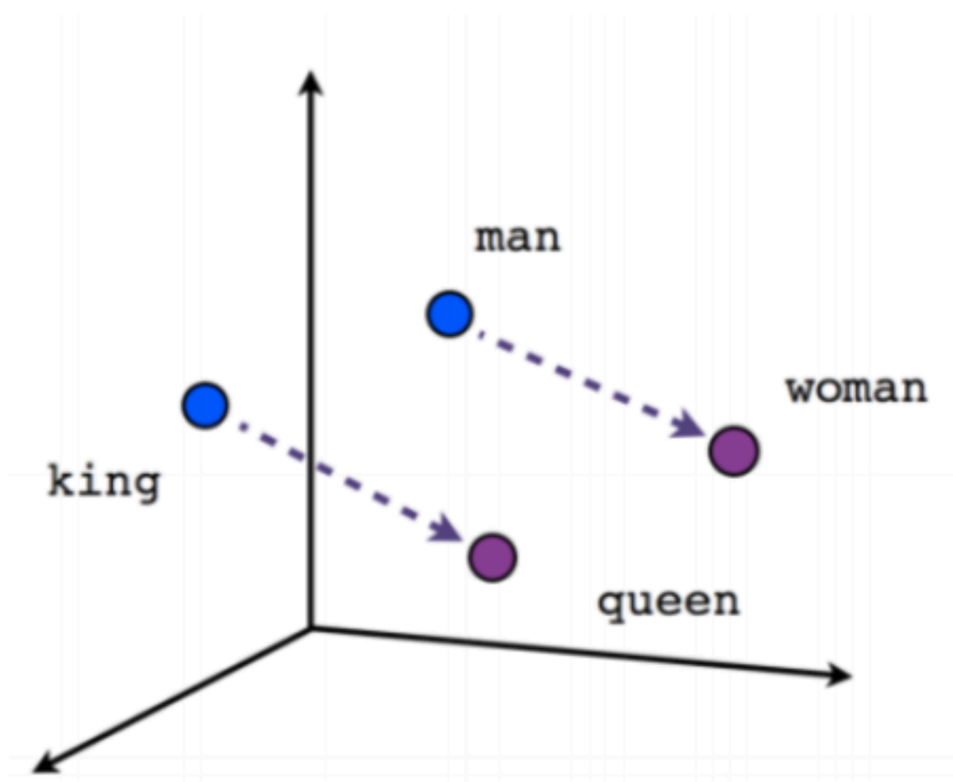
- **Distributional** model: the meaning has similarity in its nature, words with similar meanings have similar distributions

Distributed word vectors

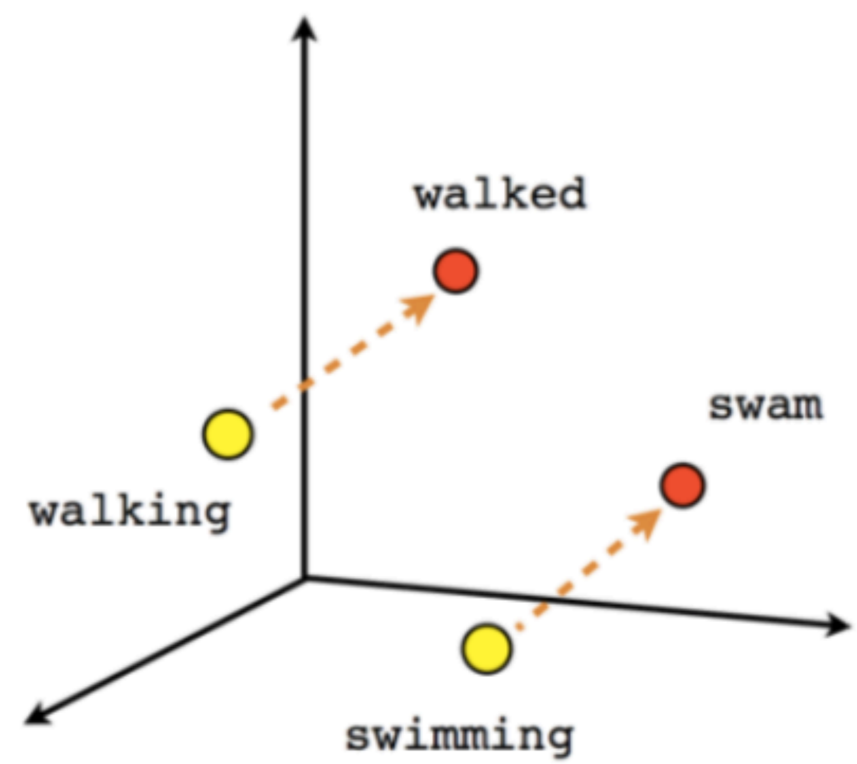
- Similar words should have similar vectors

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Interesting properties



Male-Female



Verb tense

Learning distributed word embedding

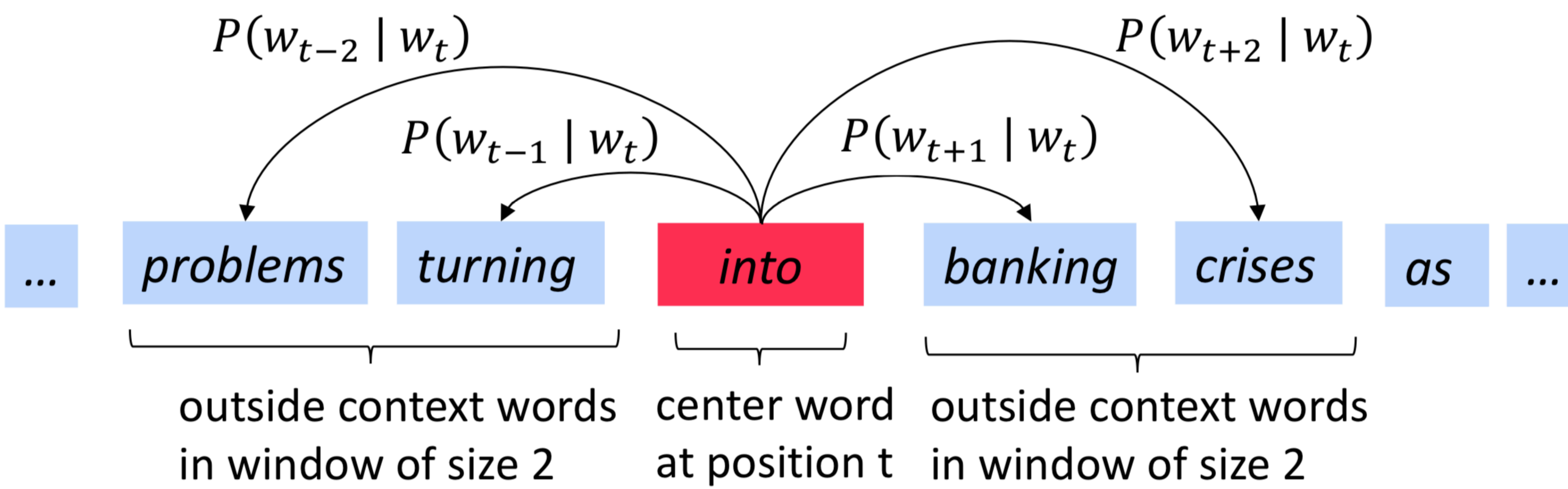
- $p\ (w_{context} \mid w_t) = \dots$
- With a loss function $J = 1 - p(w_{context}|w_t)$
- We do this iteratively and adjust the word vectors
- Result: we have really, really good word vectors

word2vec

- Two algorithms
  - Skip-grams (SG): word => context
  - Continuous Bag of Words (CBOW): context = > word
- Two training methods
  - Hierarchical softmax
  - Negative sampling

(Mikolov et al. 2013)

Skip-gram model



Credit: Stanford CS224n

Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_j.

Likelihood = $$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\ne 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- <u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

- <u>Answer:</u> We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word

- Then for a center word *c* and a context word *o*:

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

Word2vec: prediction function

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

Dot product compares similarity of *o* and *c*.
Larger dot product = larger probability

After taking exponent,
normalize over entire vocabulary

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

Training: computing gradients

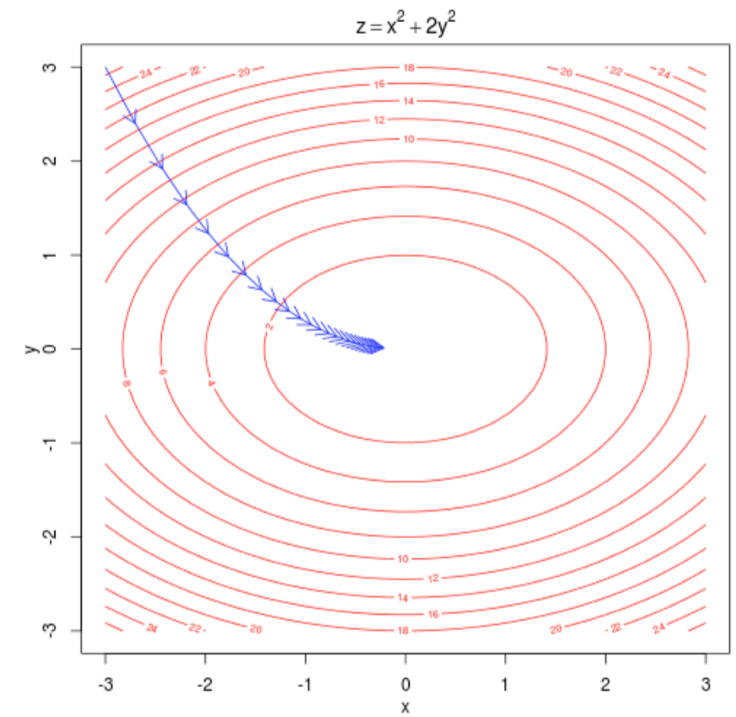
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- One long vector with all word and word context vectors
- Adjust their parameters and learn new representations
- How? Gradient Descent

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

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

Stochastic Gradient Descent



- As always, we want to adjust our parameters moving into the direction of gradient

- We don't want to compute the gradient based on each vector so we take a sample of them and use SGD

Two models

- Skip-gram: works better for small corpuses of the training data, represents even some rare words

- CBOW: several times faster, a bit better accuracy for frequent words

- For skip-gram, each pair of focus word/context word is a unique example, for CBOW they become a single instance

Negative Sampling

- Let’s not look all the word’s embedding on each step
- Let’s calculate softmax only on the a sample of words

- Let’s use frequent examples as our negative sample

Exercise: Let’s train the vectors!

Dimensionality reduction with t-SNE

- Word vectors are still quite big: 128 coordinates
- How do we see what's going on inside?
- We need to make a projection to a space with fewer number of dimensions (2 is best)

Dimensionality reduction with t-SNE

- There are many dimensionality reduction techniques, most common: PCA, Primary Component Analysis
- It tries to find some new coordinate space, with axis as combinations of original ones, trying to explain the most variance of the data
- Computable analytically, but takes a lot of time on large data and is limited in the way it can reduce the dimensions
- More on it in other lectures

Dimensionality reduction with t-SNE

- t-SNE: one of unsupervised way to learn a dimensionality reduction model, non-linear, unlike PCA. Developed by Laurens van der Maaten

- The general idea is to make a map between old points and new ones such that the distance between close points will be close, and distance between distant would be distant

- t-SNE stands for t Distributed Stochastic Neighbor Embedding. So there is t-Distribution

- Can't easily include new data into it, needs to relearn

Two similarities matrixes

Original

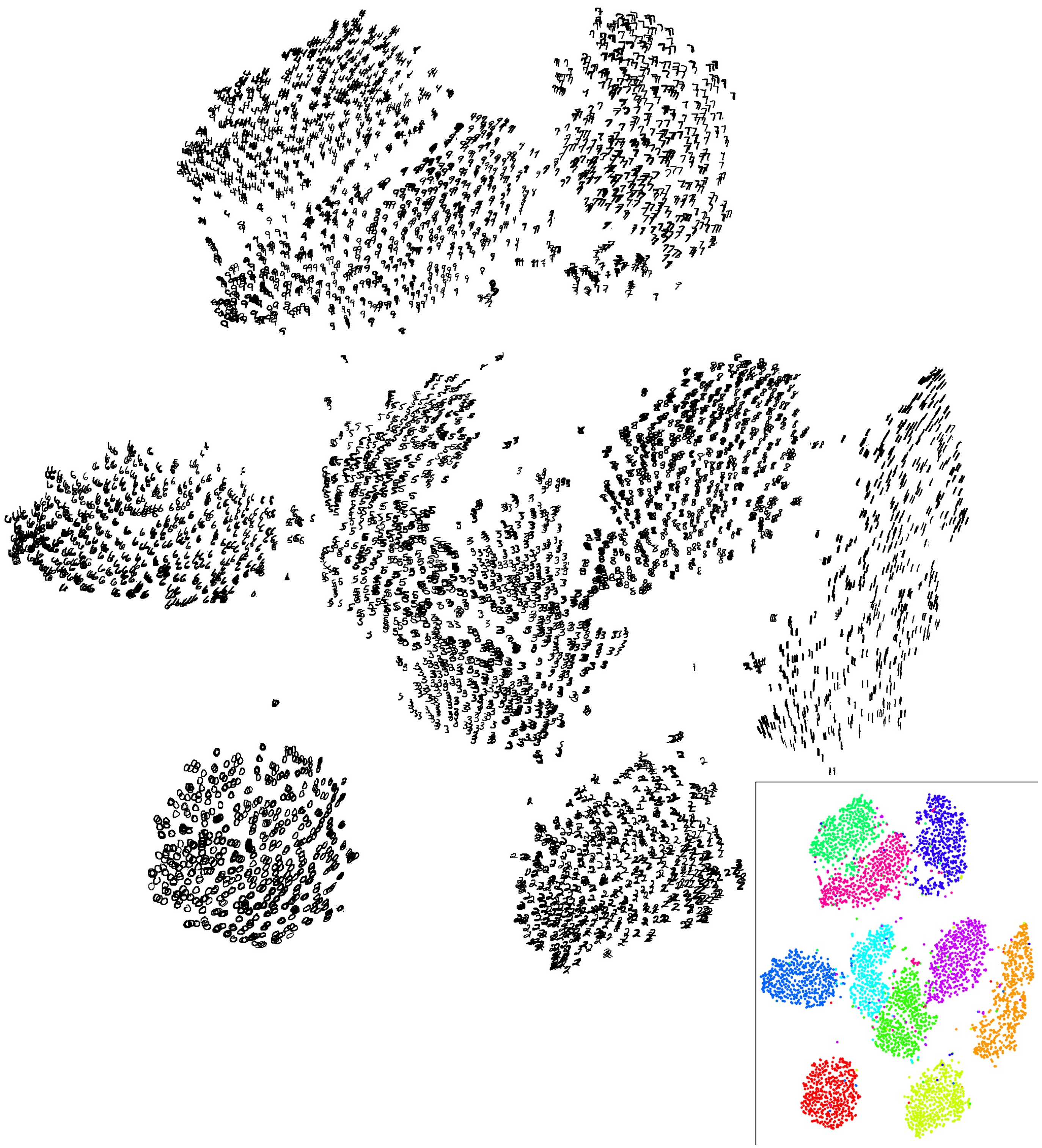
$$p_{j|i} = \frac{\exp\left(-|x_i - x_j|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-|x_i - x_k|^2 / 2\sigma_i^2\right)}$$

- We want to make them similar

On a map

$$q_{ij} = \frac{f(|x_i - x_j|)}{\sum_{k \neq i} f(|x_i - x_k|)}$$

How it looks



word2vec applications

- Word and document clustering
- Any classification problem: sentiment, fraud
- Any ML tasks that takes vectors as inputs