

Embeddings for textual data


Why embed textual data?

- Finding a good metric
 - Good for ranking
- Group/Cluster Data
 - Discover topics
- Solve word analogies
 - King : Queen :: Man : ?
- Do classification!

How to embed textual data?

Several methodologies exist:

- Latent Semantic Analysis (LSA)
- Latent Dirichlet Allocation (LDA)
- Word2Vec
- BERT and ELMo
 - Bidirectional Encoder Representations from Transformer
 - Deep contextualized word representations
- Hyperbolic Embeddings



Euclidean
Embedding

Latent Semantic Analysis/Indexing (LSA/LSI)

Given a collection of documents, the goal is to group/cluster similar documents together

How to proceed?

Assume the **Distributional Hypothesis**

*items that are used and occur in the same contexts
tend to purport similar meanings*

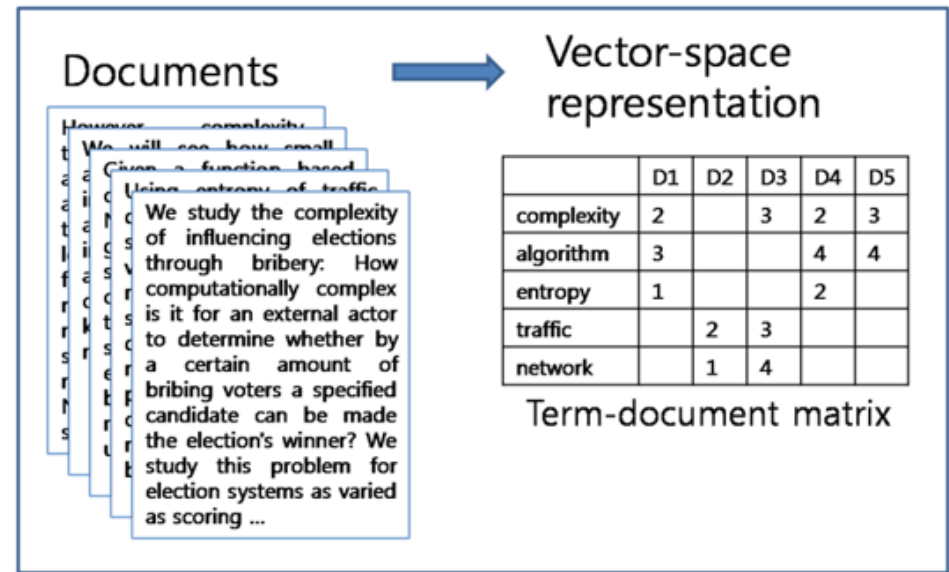
Idea: Two documents are similar, if similar words occur!

So, represent each document as the frequency count vector of words that occur and proceed from there!

Latent Semantic Analysis/Indexing (LSA/LSI)

[1980's]

- Create a term-document matrix $D = |\text{Vocab}| \times |\text{\#docs}|$
 - slightly process it with tf-idf



- We can cluster this matrix, but because of a very large and sparse vector representation usually results in “noisy vectors”
- Goal is to find a **low rank approx** of the term-document matrix M .

$$\min_{M_r} \|M - M_r\|_F$$

(where M_r is of at most rank r)

how to minimize this?

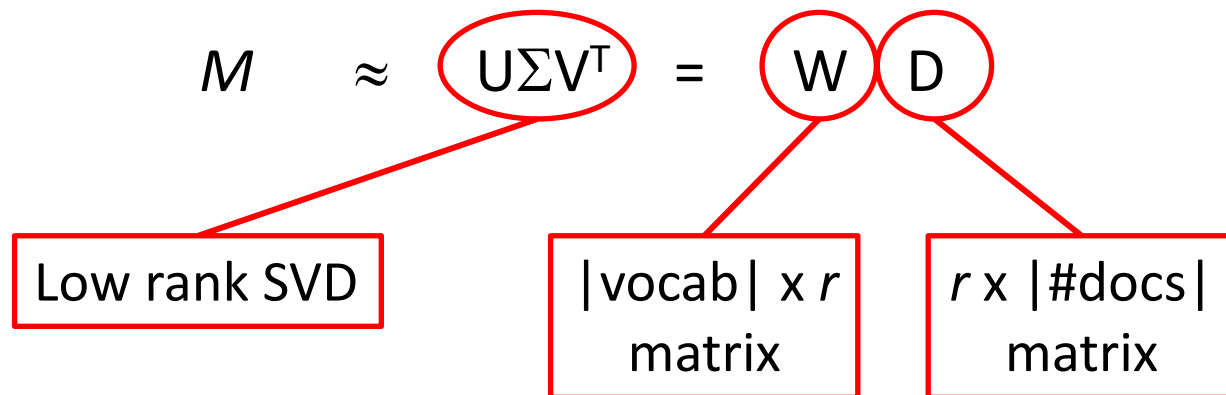
Latent Semantic Analysis/Indexing (LSA/LSI)

$$\min_{M_r} \|M - M_r\|_F \quad s.t. \quad \text{rank}(M_r) \leq r$$

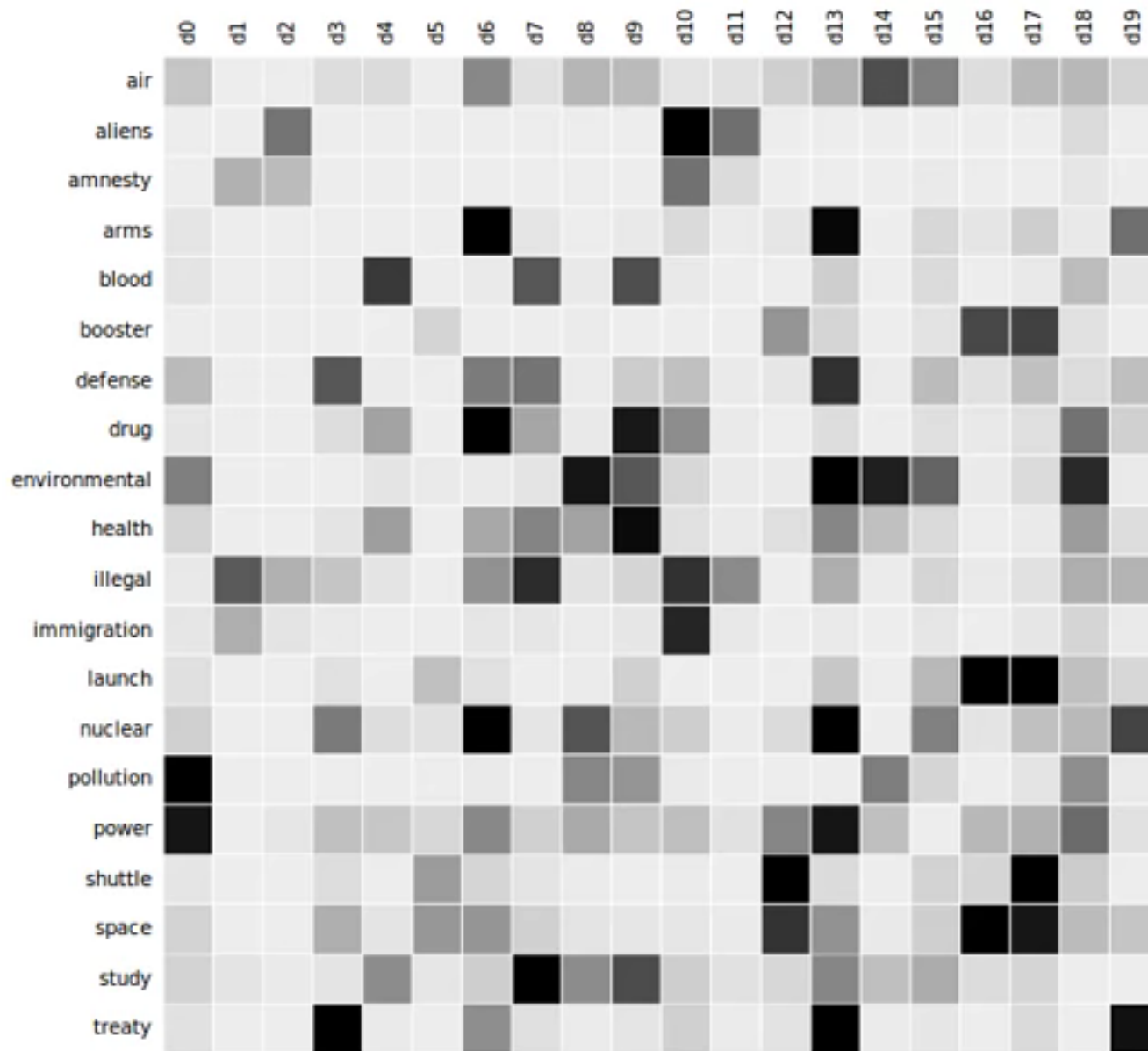
Solution: SVD and keep the left/right singular vectors/values corresponding to top r singular values! (Eckart–Young Thm.)

Looks familiar?

- We are doing **matrix factorization**!



LSI in action



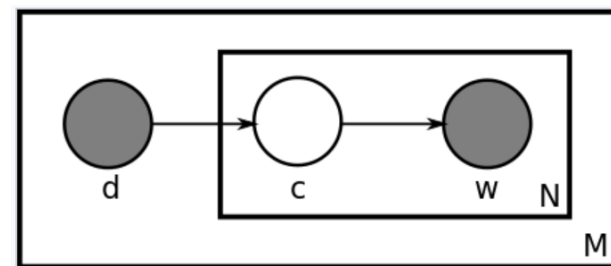
Probabilistic Model for LSI (aka pLSI)

- Each (word, document) co-occurrence is modelled as a mixture of conditionally independent distributions controlled by the topic (class). Specifically:

$$P(w, d) = \sum_c P(c)P(d|c)P(w|c) \neq P(d) \sum_c P(c|d)P(w|c)$$

each conditional distribution
is modelled as a multinomial.

let's focus on this

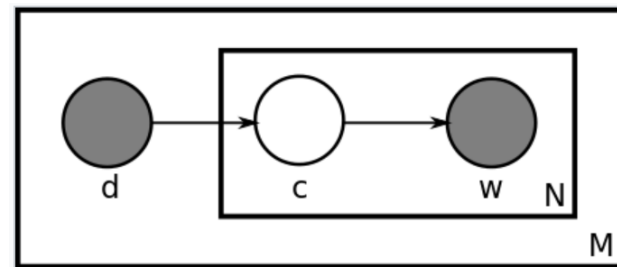


In probabilistic graphical model plate notation:

- For each document, draw N words (in the document) as follows:
 - draw a topic according to the multinomial $P(c|d)$
 - draw a word (related to the topic) according to the multinomial $P(w|c)$
- draw M documents by repeating the process above

pLSI Model Parameter Learning

$$P(w, d) = P(d) \sum_c P(c|d)P(w|c)$$



- w and d are observed, c is unobserved (latent)

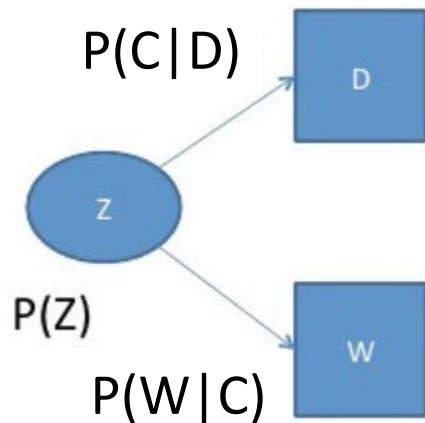
So standard way of learning the model parameters?

- EM (Expectation Maximization)!
 - Of course the optimization is highly nonconvex and the solution yielded by EM algorithm is suboptimal
- Can use variational techniques
 - Similar issues of optimality
- One can use Method of Moments! (details later)

pLSI Model Alternative View

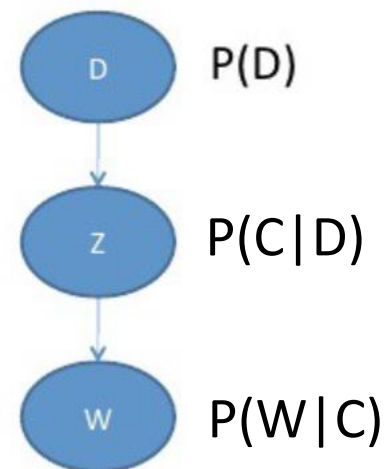
$$P(w, d) = \sum_c P(c)P(d|c)P(w|c) = P(d) \sum_c P(c|d)P(w|c)$$

- Start with topic



*asserts the same
(conditional)
independencies!*

- Start with document



Interpretation:

$$P(w, d) = \sum_c \underbrace{P(c)}_{\text{red}} \underbrace{P(d|c)}_{\text{green}} \underbrace{P(w|c)}_{\text{blue}}$$

$$M \approx \underbrace{U}_{\text{green}} \underbrace{\Sigma}_{\text{red}} \underbrace{V^T}_{\text{blue}}$$

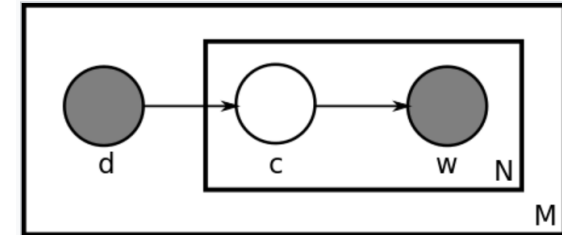
Σ is prior probabilities of topics

U is prob. dist. of document given topic

V is prob. dist. of words given topic

Generalizing the pLSI Model (towards LDA)

Some issues with the pLSI model:



- Because we have **no parameters to model** $P(D)$, we don't know how to assign probabilities to new documents (ie cannot generate new documents)
- The number of parameters for pLSI **grows linearly** with the number of documents we have, so it is prone to overfitting

Solution: Latent Dirichlet Allocation (LDA)!

- LDA **uses dirichlet priors** for the document-topic and word-topic distributions, lending itself to better generalization
- Can be viewed as a **Bayesian version** of pLSI

Latent Dirichlet Allocation (LDA)

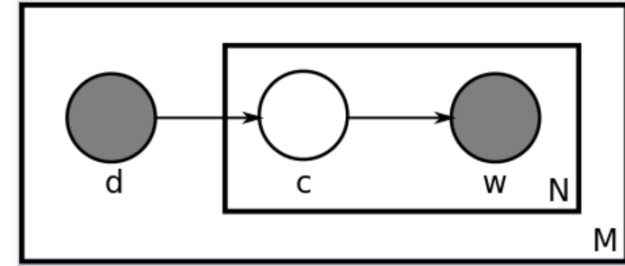
Motivation:

- Let's say the corpus we are looking at has documents from **very different** subject areas
- To model this, we need a distribution that heavily weights one specific topic, and doesn't give much weight to others
- Example:
 - Mixture X: 90% topic A, 5% topic B, 5% topic C
 - Mixture Y: 5% topic A, 90% topic B, 5% topic C
 - Mixture Z: 5% topic A, 5% topic B, 90% topic C

A Dirichlet distribution (as a prior over topics in a document) enables us to control for this!

Latent Dirichlet Allocation (LDA)

In pLSA, we sample a document,
then a topic based on that document,
then a word based on that topic



In LDA,
(distribution over words)

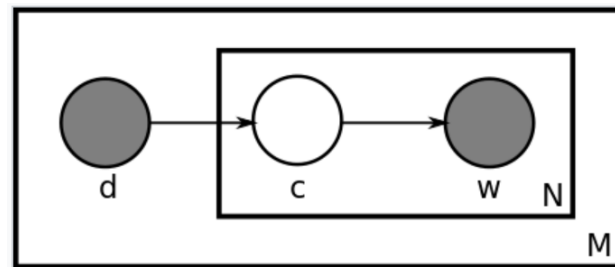
- From Dirichlet distribution $\text{Dir}(\beta)$, we select a random sample representing the word distribution of the topic Z
- We do this K times to get K different topic specific word distribution ϕ

(for generating M documents, repeat the following procedure M times)

- From a Dirichlet distribution $\text{Dir}(\alpha)$, draw a random sample representing the topic distribution or topic mixture, of a particular document. This topic distribution is θ .
- To generate a word for the document, from θ , we select a particular topic Z based on the distribution. A word is then drawn from word distribution is ϕ for that Z . Do this N times to generate N words.

Latent Dirichlet Allocation (LDA)

In pLSA, we sample a document,
then a topic based on that document,
then a word based on that topic

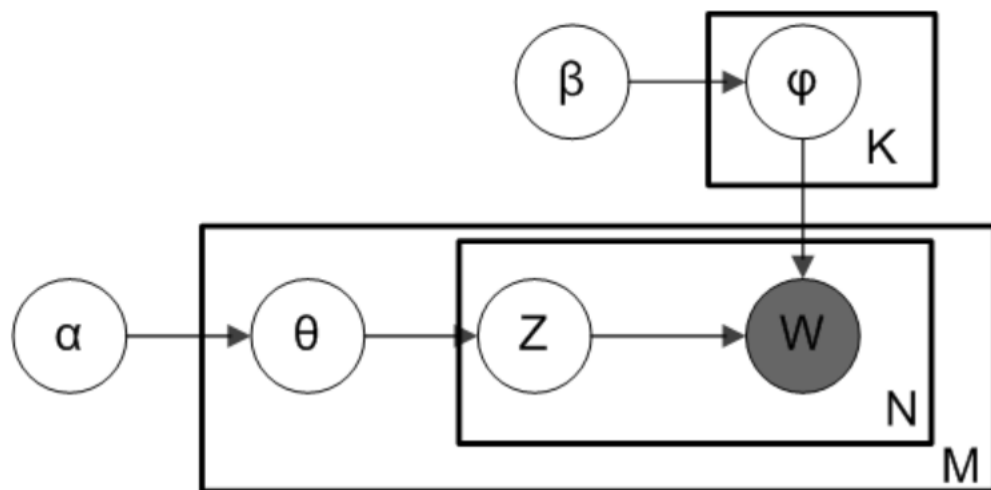


In LDA,
(distribution over words)

- From Dirichlet
- the word di
- We do this

(for generating

- From a Diric
- the topic di
- distribution
- To generate



le representing

tribution ϕ

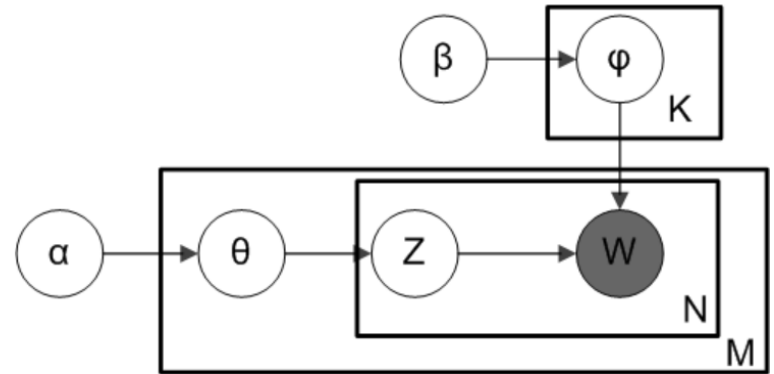
M times)

representing
ment. This topic

articular topic

Z based on the distribution. A word is then drawn from word distribution is ϕ for that Z . Do this N times to generate N words.

LDA Advantages



Some advantages:

- LDA typically works better than pLSA because it **can generalize to new documents** easily (it has a generative model to generate new documents!)
- LDA tends **to extract human-interpretable topics** from a document corpus, where each topic is characterized by the words they are most strongly associated with.

so... how to we learn the parameters?

Learning Parameters a Different Way

Consider a simple mixture model (GMM) in R, with two components

- Component 1: $N(\mu_1, \sigma_1^2)$ with probability p
- Component 2: $N(\mu_2, \sigma_2^2)$ with probability $1 - p$

How many parameters to learn?

And how do you estimate those parameters??

An alternate (to EM) view to learn the parameters:

There are basically **five parameters**, if we can come up with **five equations** that provide a relationship between these to these five parameters, then perhaps we can analytically solve this system!

That is basically the method of moments!

Method of Moments

$$\text{Mixture Density} = p N(\mu_1, \sigma_1^2) + (1 - p) N(\mu_2, \sigma_2^2)$$

Moment Generating Function:

$$p \exp\left(\mu_1 t + \frac{1}{2} \sigma_1^2 t^2\right) + (1 - p) \exp\left(\mu_2 t + \frac{1}{2} \sigma_2^2 t^2\right)$$

The first few moments are:

$$m_1 = p\mu_1 + (1 - p)\mu_2$$

$$m_2 = p(\mu_1^2 + \sigma_1^2) + (1 - p)(\mu_2^2 + \sigma_2^2)$$

$$m_3 = p(\mu_1^3 + 3\mu_1\sigma_1^2) + (1 - p)(\mu_2^3 + 3\mu_2\sigma_2^2)$$

$$m_4 = p(\mu_1^4 + 6\mu_1^2\sigma_1^2 + 3\sigma_1^4) + (1 - p)(\mu_2^4 + 6\mu_2^2\sigma_2^2 + 3\sigma_2^4)$$

$$m_5 = p(\mu_1^5 + 10\mu_1^3\sigma_1^2 + 15\mu_1\sigma_1^4) + (1 - p)(\mu_2^5 + 10\mu_2^3\sigma_2^2 + 15\mu_2\sigma_2^4)$$

So just solve this system!

Ummm...

Issues to consider:

- How to solve the (non-linear) system?
- Cannot efficiently approximate the higher order moments!
- What about higher dimensional problems?

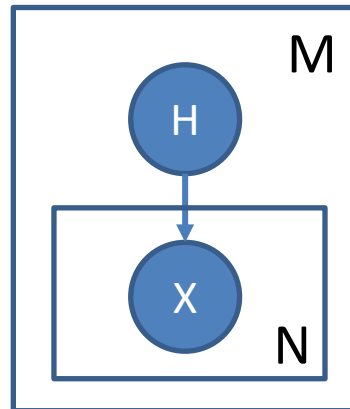
Saving grace:

*most latent variable models in the literature
only require estimation of the first three or four moments!!!*

A Concrete Example

Single topic Model:

- Generative Mechanism:
For each of the M documents
 - Draw a topic (out of K topics) from the topic distribution
 - Draw N words according to the topic(note: each document only has one topic!)



A Concrete Example

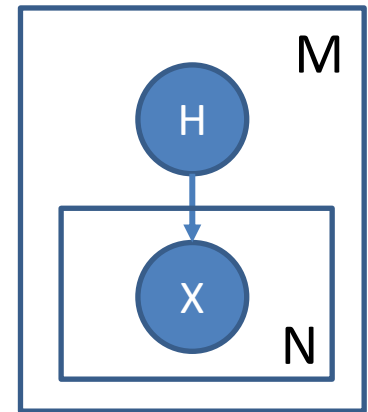
Topic distribution: (w_1, \dots, w_k)

$$\Pr[h = j] = w_j, \quad j \in [k]$$

If each word is a one hot encoding (vocab size d), then

$x_t = e_i$ if the t -th word in the document is the word i

Let $\mu_1, \dots, \mu_k \in \mathbb{R}^d$ be the distribution over the words for each topic



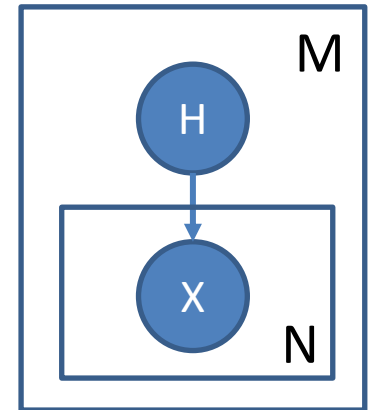
want to recover w 's and μ 's

Learning the Parameters (using MoM)

$$\begin{aligned}\mathbb{E}[x_1 \otimes x_2] &= \sum_{1 \leq i, j \leq d} \Pr[x_1 = e_i, x_2 = e_j] e_i \otimes e_j \\ &= \sum_{1 \leq i, j \leq d} \Pr[\text{1st word} = i, \text{2nd word} = j] e_i \otimes e_j,\end{aligned}$$

$$\begin{aligned}\mathbb{E}[x_t | h = j] &= \sum_{i=1}^d \Pr[t\text{-th word} = i | h = j] e_i \\ &= \sum_{i=1}^d [\mu_j]_i e_i = \mu_j, \quad j \in [k]\end{aligned}$$

$$\mathbb{E}[x_1 \otimes x_2 | h = j] = \mathbb{E}[x_1 | h = j] \otimes \mathbb{E}[x_2 | h = j] = \mu_j \otimes \mu_j$$



*want to recover
w's and μ 's*

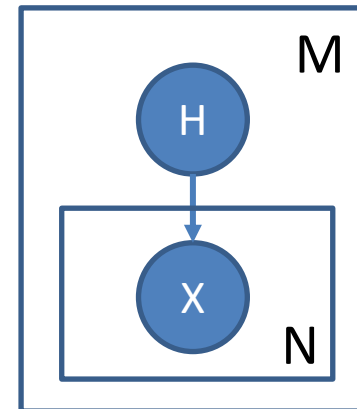
Learning the Parameters (using MoM)

$$\begin{aligned} M_2 &:= \mathbb{E}[x_1 \otimes x_2] \\ &= \sum_{i=1}^k w_i \mu_i \otimes \mu_i \end{aligned}$$

Looks familiar?

Eigen decomposition?

Not an eigen decomposition



*want to recover
w's and μ 's*

$$\begin{aligned} M_3 &:= \mathbb{E}[x_1 \otimes x_2 \otimes x_3], \\ &= \sum_{i=1}^k w_i \mu_i \otimes \mu_i \otimes \mu_i. \end{aligned}$$

seems to have the same issue

Learning the Parameters (using MoM)

$$M_2 := \mathbb{E}[x_1 \otimes x_2] = \sum_{i=1}^k w_i \mu_i \otimes \mu_i \quad [\text{Anandkumar, Ge, Hsu, Kakade, Telgarsky '12}]$$

Can perhaps 'orthogonalize' M_2

Find a linear transform W such $W^\top M_2 W = I$

*whitening
the data*

$$\tilde{\mu}_i := \sqrt{w_i} W^\top \mu_i$$

$$M_2(W, W) = \sum_{i=1}^k W^\top (\sqrt{w_i} \mu_i) (\sqrt{w_i} \mu_i)^\top W = \sum_{i=1}^k \tilde{\mu}_i \tilde{\mu}_i^\top = I$$

$$\tilde{M}_3 := M_3(W, W, W) \in \mathbb{R}^{k \times k \times k}$$

$$\tilde{M}_3 = \sum_{i=1}^k w_i (W^\top \mu_i)^{\otimes 3} = \sum_{i=1}^k \frac{1}{\sqrt{w_i}} \tilde{\mu}_i^{\otimes 3}$$

*orthogonal
decomposition of tensor!*

Word Embeddings via Neural Networks

LSI and LDA models don't use the power of neural networks (ie having highly flexible non-linear relationships between the modelling variables) to learn the embeddings

How can we incorporate neural networks and potentially develop even more powerful models?

Word2Vec

Assuming that the ‘distributional hypothesis’ is true

Why don’t we learn to predict what words are more likely to occur near a given word (in a text corpus)?

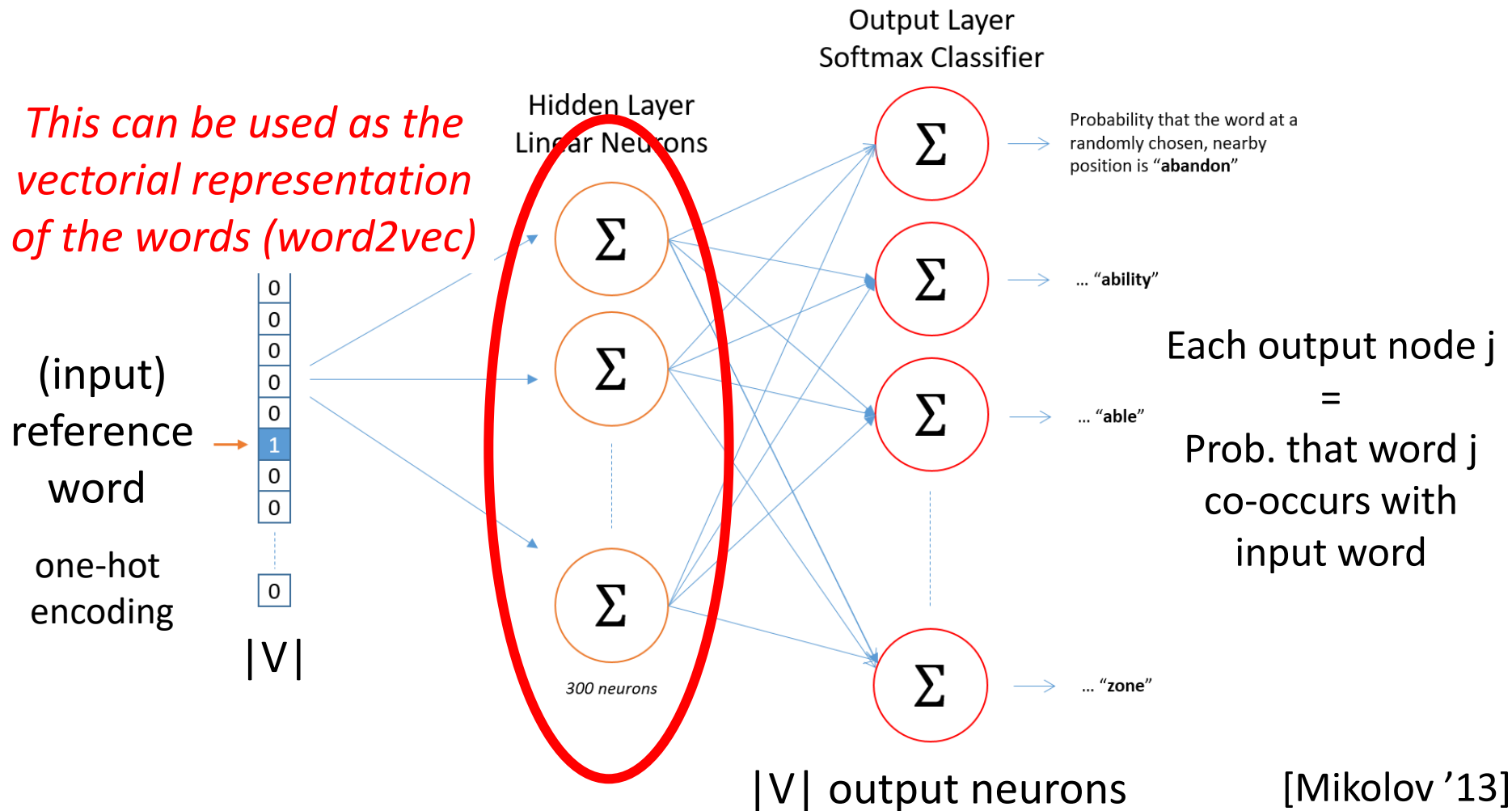
Example:

- if a reference word is “Soviet”, what words do we expect to co-occur **probably** related words like “Russia” or “Union” and **not** unrelated words like “duck” or “apple”

So how can we train a NN to predict the probability of a word co-occurring given a reference word?

Word2Vec

Let each word in the vocabulary be a one-hot encoding and consider the following network:



Word2Vec Training

Source Text

Training Samples

<div>The quick brown fox jumps over the lazy dog.</div>	→	(the, quick) (the, brown)
<div>The quick brown fox jumps over the lazy dog.</div>	→	(quick, the) (quick, brown) (quick, fox)
<div>The quick brown fox jumps over the lazy dog.</div>	→	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
<div>The quick brown fox jumps over the lazy dog.</div>	→	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Word2Vec Advantages

- Simple and intuitive
- Has been shown to work well on downstream text related tasks
- Surprising observation: can do word analogies very well!
man : king :: woman :: queen

Word2Vec: why analogies work?

Consider the analogy

man : king :: woman : queen

There is a semantic concept -- “royal”ty -- which when applied to “man” yields “king”, and the same exact semantic concept is applied to “woman” to get “queen”

royal (man) = king (semantic composability)

Key question: when does semantic **composability** becomes **additive**?

ie, how can we get: man + royal = king

Word2Vec: why analogies work?

It can be shown provably that:

Under some uniformity conditions (on the distribution of words)
a skip-gram model yields additive composability!

*it explains the additive nature of solving
word analogies by word2vec embeddings*

Additive Analogies in Word2Vec: Proof Sketch

One way to mathematically model compositionality is to say that a **set of context words** c_1, \dots, c_m have the same meaning as a **single context word** c if for every other word w

$$p(w|c_1, \dots, c_m) = p(w|c)$$

e.g. for all w

$$p(w \mid \text{man, royal}) = p(w \mid \text{king})$$

*[in reality, it will be a close approximation
which complicates the analysis]*

we say that the word “king” *paraphrases* the set “man, royal”

Additive Analogies in Word2Vec: Proof Sketch

Fact 1. If c paraphrases the set $C = \{c_1, \dots, c_m\}$, then

$$\sum_{w \in V} p(w|c) \mathbf{v}_w = \sum_{w \in V} p(w|C) \mathbf{v}_w$$

for \mathbf{v}_w is the skipgram representation of word w in the vocabulary V .

*(even when the paraphrase is
a close approximation)*

Fact 2. If words are uniformly distributed (in the vocabulary), then for the skipgram model the paraphrase of $C = \{c_1, \dots, c_m\}$ is

$$c_1 + c_2 + \dots + c_m$$

*(paraphrased words are in the local
'context' in the skipgram model)*

Additive Analogies in Word2Vec: Proof Sketch

Using the facts, we get

$\text{man} + \text{royal} = \text{king}$

$\text{woman} + \text{royal} = \text{queen}$

Therefore:

$(\text{woman} - \text{man}) + \text{king} = \text{queen}$

