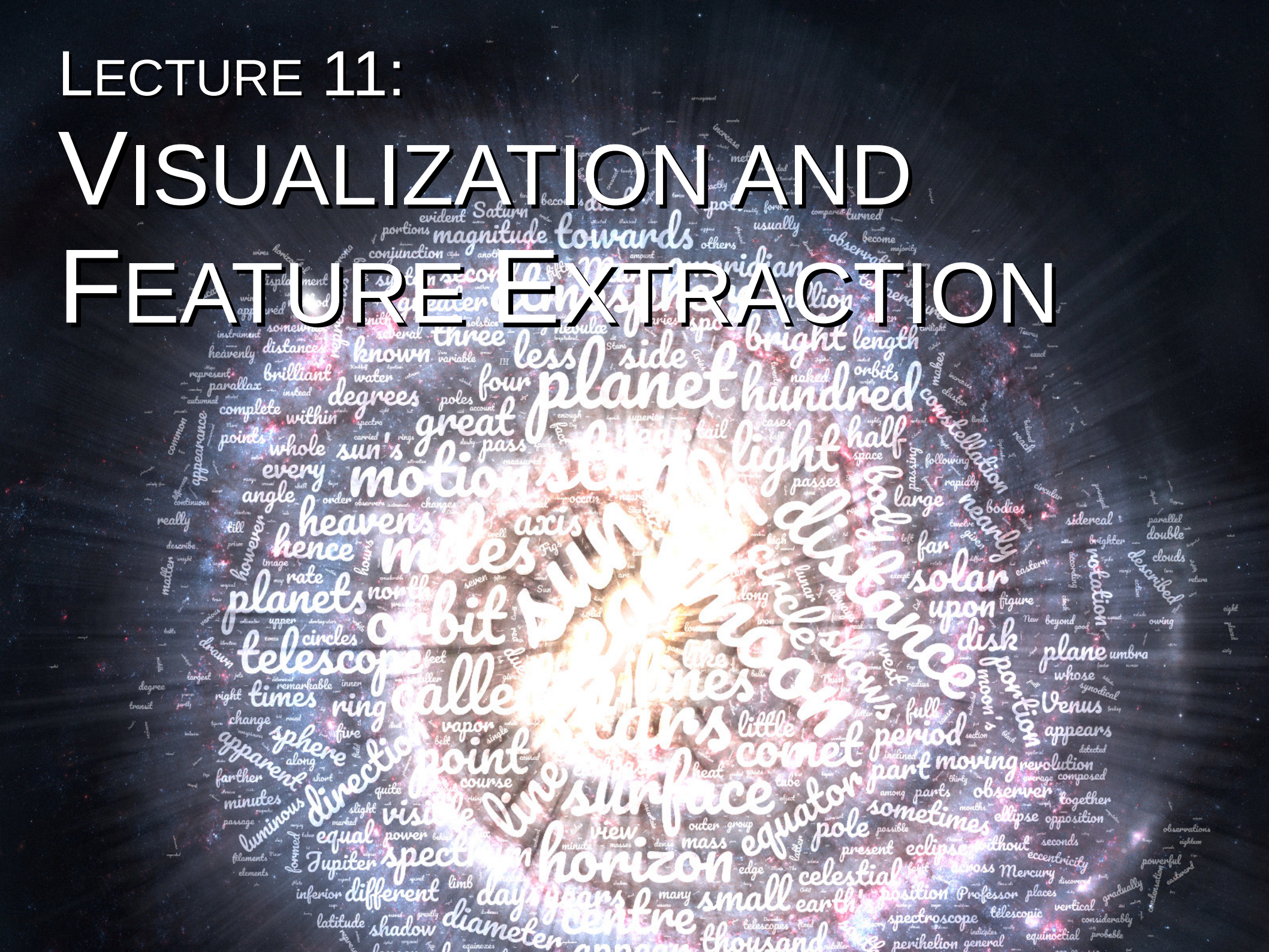
Lecture 11:

Visualization and Feature Extraction



Visualization of document collections

- Visualization is a very useful stage in data analysis: it is much more than just creating attractive artwork or simple infographics.
- Statistical visualization helps bring human analysts into the data analysis loop, and helps check for
  - clusters and subclusters and their closeness relationships
  - density and sparsity of different areas of data
  - trends, divergences from different concentrations of data
  - outliers
  - consistency with previous expectations
  - consistency with modeling assumptions
  - suitability for further processing with different predictive models
  - Can be integrated into interactive exploratory tools
  - Human conclusions about the data can be the main outcome of research

Visualization of document collections

- See the course **DATA.STAT.770 Dimensionality Reduction and Visualization** (periods III-IV) for greater coverage of dimensionality reduction and visualization methods, here we only mention a few
- The vector space model (TF-IDF) representation of documents associates each document with high-dimensional vectors, and represents a corpus as a document-term feature value matrix.
- The matrix can be used to visualize the corpus in different ways.

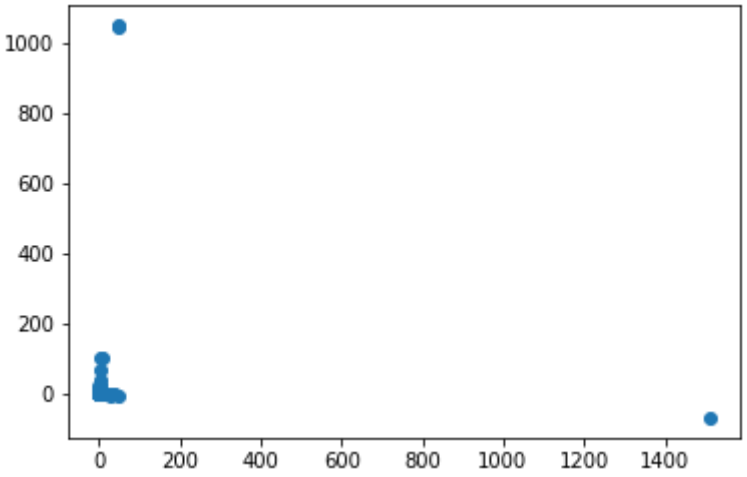
Principal component analysis

- **Principal component analysis (PCA)**
  - Linear projection method: creates low-dimensional (e.g. 2D) coordinates $\boldsymbol{y}$ for data items $\boldsymbol{x}$ as $\boldsymbol{y} = \boldsymbol{W}^T \boldsymbol{x}$ where $\boldsymbol{W}$ is a projection matrix
  - PCA extracts **orthogonal directions containing maximal variance** in vectorial data.
  - Equivalently: orthogonal projection directions (low-dimensional coordinates) that yield **smallest error in linear reconstruction** of the high-dimensional vectors from the low-dimensional coordinates.
  - Can be solved by eigenvalue decomposition of the covariance matrix of the data vectors. PCA projection directions = eigenvectors of maximal eigenvalues.
  - Or by singular value decomposition of the data matrix.

Principal component analysis

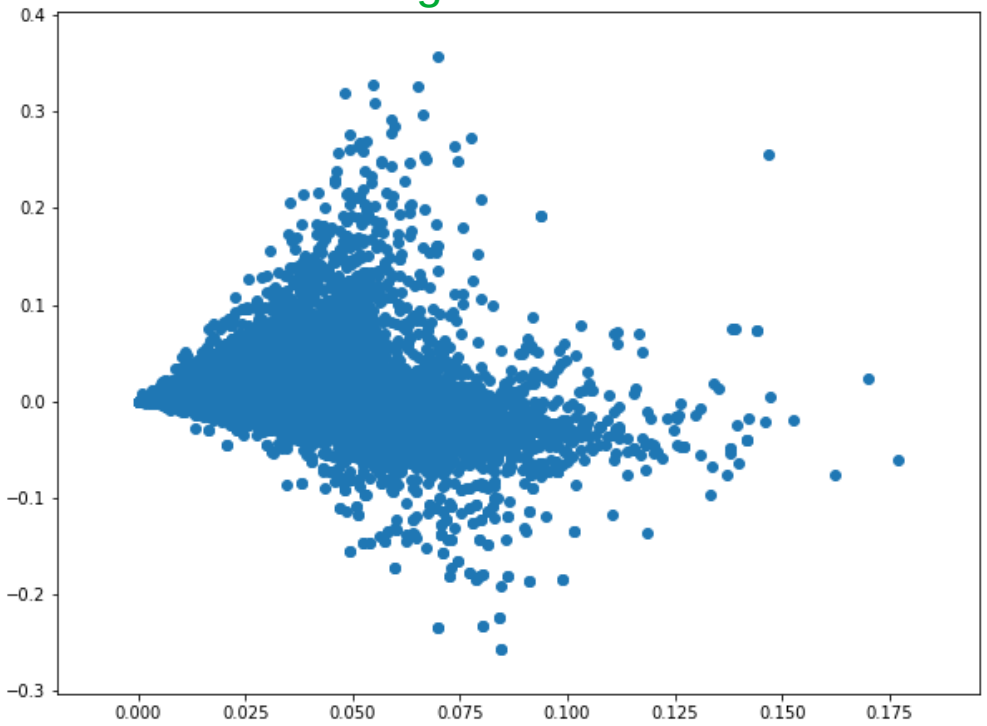
```
import sklearn
import sklearn.decomposition
X=tfidfmatrix.copy()
# Normalize tf-idf vector norms
for k in range(n_docs):
    print(k)
    X[k,:]=X[k,:]/numpy.sqrt(numpy.sum(X[k,:].multiply(X[k,:]),axis=1)[0]+0.0000000001)
# Plot projected documents
svdmodel=sklearn.decomposition.TruncatedSVD(n_components=2, n_iter=70, random_state=42)
documentplot = svdmodel.fit(X).transform(X)
import matplotlib.pyplot
%matplotlib auto
myfigure, myaxes = matplotlib.pyplot.subplots();
myaxes.scatter(documentplot[:,0],documentplot[:,1]);
```

Result is not what we hoped for:
this is because of sparsity.



Without normalizing norms

After normalizing norms

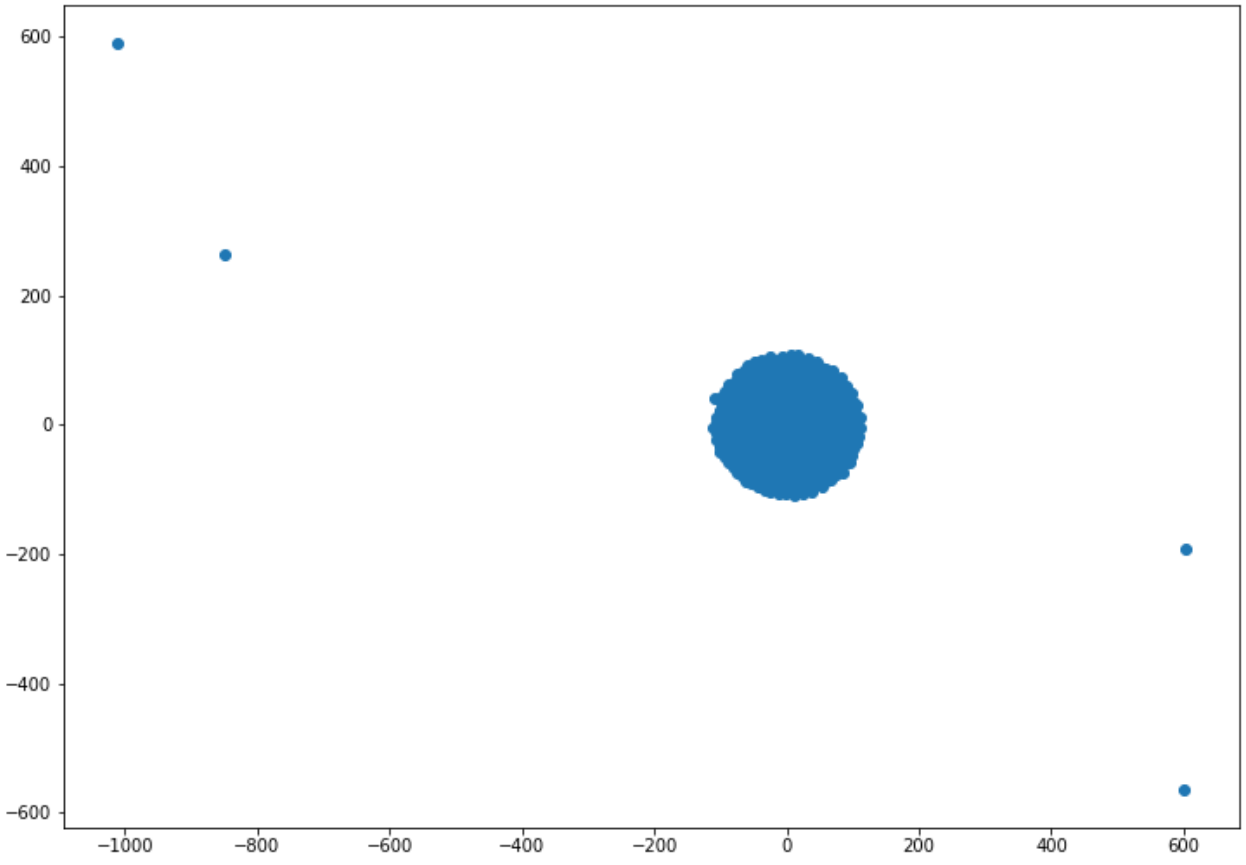


t-distributed Stochastic Neighbor Embedding

- **t-distributed Stochastic Neighbor Embedding (t-SNE)**
  - Nonlinear projection method: creates low-dimensional (e.g. 2D) coordinates $\boldsymbol{y}$ for data items $\boldsymbol{x}$.
  - The t-SNE method **does not create a mapping**: it only finds low-dimensional coordinates for the specific points in the training data set.
  - t-SNE extracts low-dimensional coordinates that try to **preserve a neighborhood distribution of data** points to be similar to an original high-dimensional neighborhood distribution.
  - The neighborhood is a joint distribution of point pairs
  - Solved by iterative nonlinear optimization (gradient descent algorithms) to minimize the cost.
  - Tree-based approximations are used to avoid quadratic O(N^2) cost per iteration, approximation takes only O(NlogN) time per iteration. However, the scikit-learn implementation does not seem to use them.

t-distributed Stochastic Neighbor Embedding

```
# Create a 1000-sample subset to avoid long running times
subsetindices=numpy.random.permutation(numpy.shape(X)[0])
Xsmall=X[subsetindices[0:1000],:].toarray()
# Run t-SNE
import sklearn.manifold
tsnemodel = sklearn.manifold.TSNE(n_components=2, verbose=1, perplexity=20, n_iter=400)
tsneplot = tsnemodel.fit_transform(Xsmall)
# Plot the result
import matplotlib.pyplot
%matplotlib auto
myfigure, myaxes = matplotlib.pyplot.subplots();
myaxes.scatter(tsneplot[:,0],tsneplot[:,1]);
```

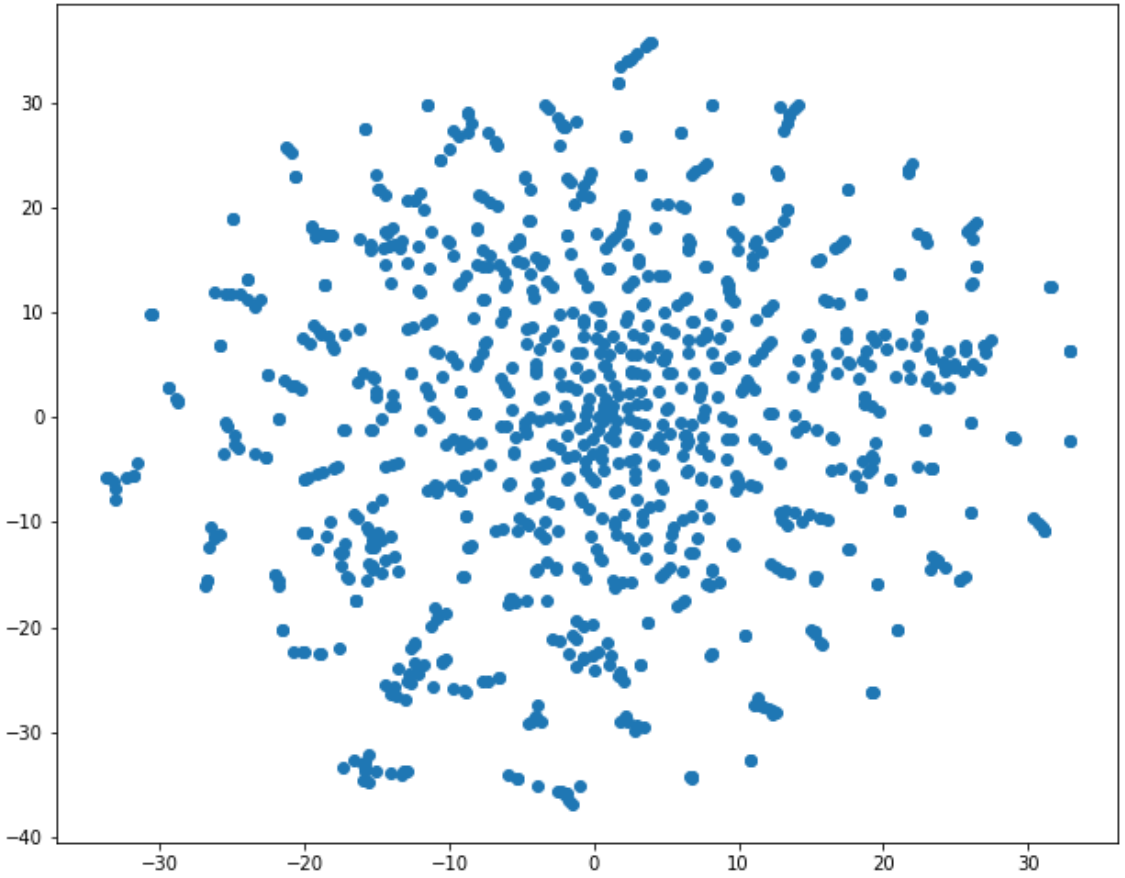


Why does the result look like this?

Answer: our vocabulary pruning has been so agressive, most documents in our subset have no words in common ----> maximum distance
(only 25182 pairs of documents have words in common out of 499500, about 5%)

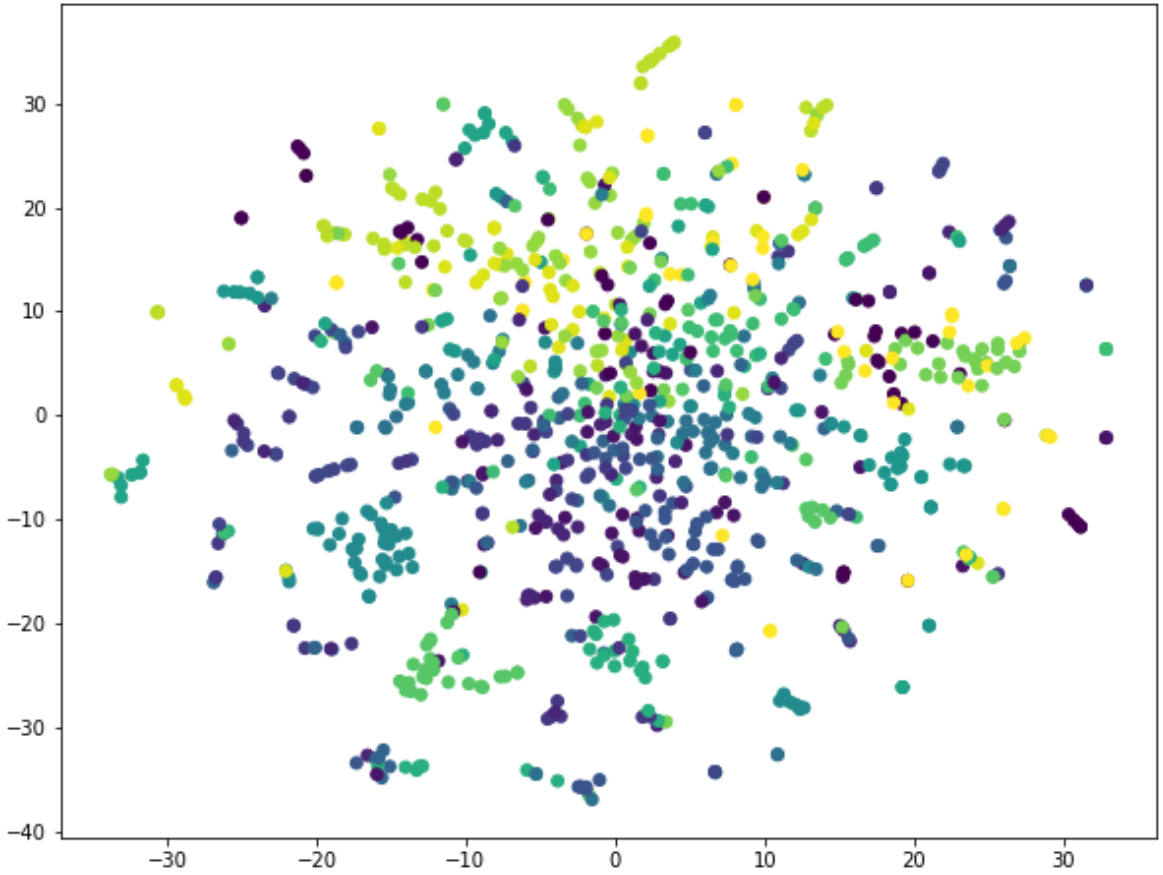
t-distributed Stochastic Neighbor Embedding

```
#%% PCA followed by t-SNE
# Find a 100-dimensional PCA projection
svdmodel=sklearn.decomposition.TruncatedSVD(n_components=100, n_iter=70,
random_state=42)
pcaplot = svdmodel.fit(X).transform(X)
Xsmaller=pcaplot[subsetindices[0:1000],:]
# Run t-SNE on the 100-dimensional PCA-projected data
tsneplot2 = tsnemodel.fit_transform(Xsmaller)
import matplotlib.pyplot
%matplotlib auto
myfigure, myaxes = matplotlib.pyplot.subplots();
myaxes.scatter(tsneplot2[:,0],tsneplot2[:,1]);
```



t-distributed Stochastic Neighbor Embedding

```
#%% PCA followed by t-SNE
# Find a 100-dimensional PCA projection
svdmodel=sklearn.decomposition.TruncatedSVD(n_components=100, n_iter=70,
random_state=42)
pcaplot = svdmodel.fit(X).transform(X)
Xsmaller=pcaplot[subsetindices[0:1000],:]
# Run t-SNE on the 100-dimensional PCA-projected data
tsneplot2 = tsnemodel.fit_transform(Xsmaller)
import matplotlib.pyplot
%matplotlib auto
myfigure, myaxes = matplotlib.pyplot.subplots();
myaxes.scatter(tsneplot2[:,0],tsneplot2[:,1],c=directoryindices[subsetindices[0:1000]]);
```



Plotted with colors corresponding to the known newsgroup indices

Word embedding

- The methods we have discussed so far have treated words only as indices into a vocabulary, and derived probability distributions based on those indices only.
- However, this can make it hard to make simplifying assumptions to generalize between words and predict new content
  - For example, words that are (near) synonyms like 'coding' and 'programming' may behave similarly even though they have different vocabulary indices.
- **Word embedding** is a set of approaches that try to derive vectorial feature representations for words, so that words with similar feature vectors would have similar behavior. If the features are well built, predictive models can make use of them.

Word embedding

- **Word2vec** is a highly influential word embedding method. We will discuss three variants: **continuous bag of words**, **skip-gram**, and **"negative sampling" based skip-gram.**
- Word2vec references:
  - Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. ArXiV, 2013. https://arxiv.org/abs/1301.3781
  - Tomas Mikolov, 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 (NIPS 2013). https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html
  - Yoav Goldberg and Omer Levy. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. ArXiV, 2014. https://arxiv.org/abs/1402.3722
- **GloVE** ("global vectors for word representation")
- GloVE references:
  - https://nlp.stanford.edu/projects/glove/

Word embedding

- The hope is that a successful word embedding model can be used to study word semantic relationships and used in predictive models.
- Example from word2vec:

picture from
Tomas Mikolov, 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 (NIPS 2013).

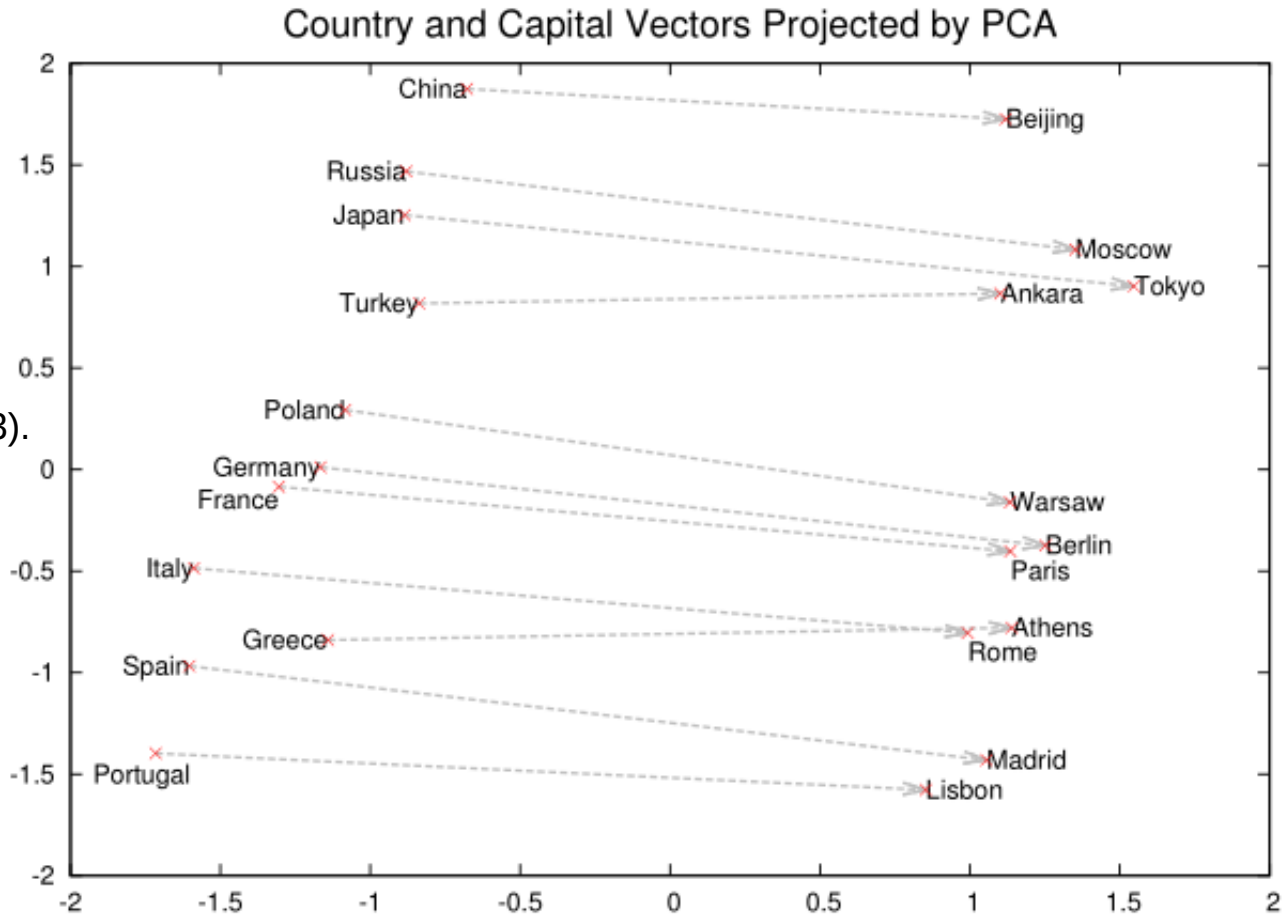


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Continuous bag of words (CBOW) model

- **Continuous bag of words (CBOW) model**: a word2vec model for conditional probabilities of words w given a set of their **contexts** c.
- For each word w appearing in some document, the contexts are a set $\boldsymbol{C}(w)$ of surrounding words in a window ("context-words").
  - Mikolov et al.: $\boldsymbol{C}(w)$ contains the $R=4$ words to the left of w and the $R=4$ words to the right of w, in total $2R$ words.
- The probability of word w given the context is written as a softmax:

$$p\left(w|\boldsymbol{C}\left(w\right);\boldsymbol{\theta}\right)=\frac{e^{\text{affinity}\left(w,\boldsymbol{C}\left(w\right)\right)}}{\sum_{v=1}^{V} e^{\text{affinity}\left(v,\boldsymbol{C}\left(w\right)\right)}}$$

where $\text{affinity}\left(w,\boldsymbol{C}\left(w\right)\right)$ is any function of w and $\boldsymbol{C}(w)$, can take positive/negative/zero values. Higher affinity yields larger conditional probability.

- This part is not yet an assumption: any finite discrete probability distribution over V choices can be written in a similar way as

$$p_w=\frac{p_w}{\sum_{v=1}^{V} p_v}=\frac{e^{\log p_w}}{\sum_{v=1}^{V} e^{\log p_v}}=\frac{e^{\log p_w+Const}}{\sum_{v=1}^{V} e^{\log p_v+Const}}$$

where Const is any constant.

Continuous bag of words (CBOW) model

- The CBOW model assumes each word has a feature representation as a d-dimensional vector $\boldsymbol{\phi}_w$, and each context-word has a feature representation as a d-dimensional vector $\boldsymbol{\psi}_c$.
- The affinity an inner product of the word vector with the averaged vector of the $2R$ context-words:

$$\text{affinity}\left(w, \boldsymbol{C}(w)\right) = \boldsymbol{\phi}_w^T \left(\frac{1}{2R} \sum_{c \in \boldsymbol{C}(w)} \boldsymbol{\psi}_c \right)$$

- The vectors of all words and all context-words in their vocabularies are parameters of the CBOW model: $\boldsymbol{\theta} = \left(\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_V\}, \{\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_U\} \right)$
- The "probability" of a corpus with M documents is then:

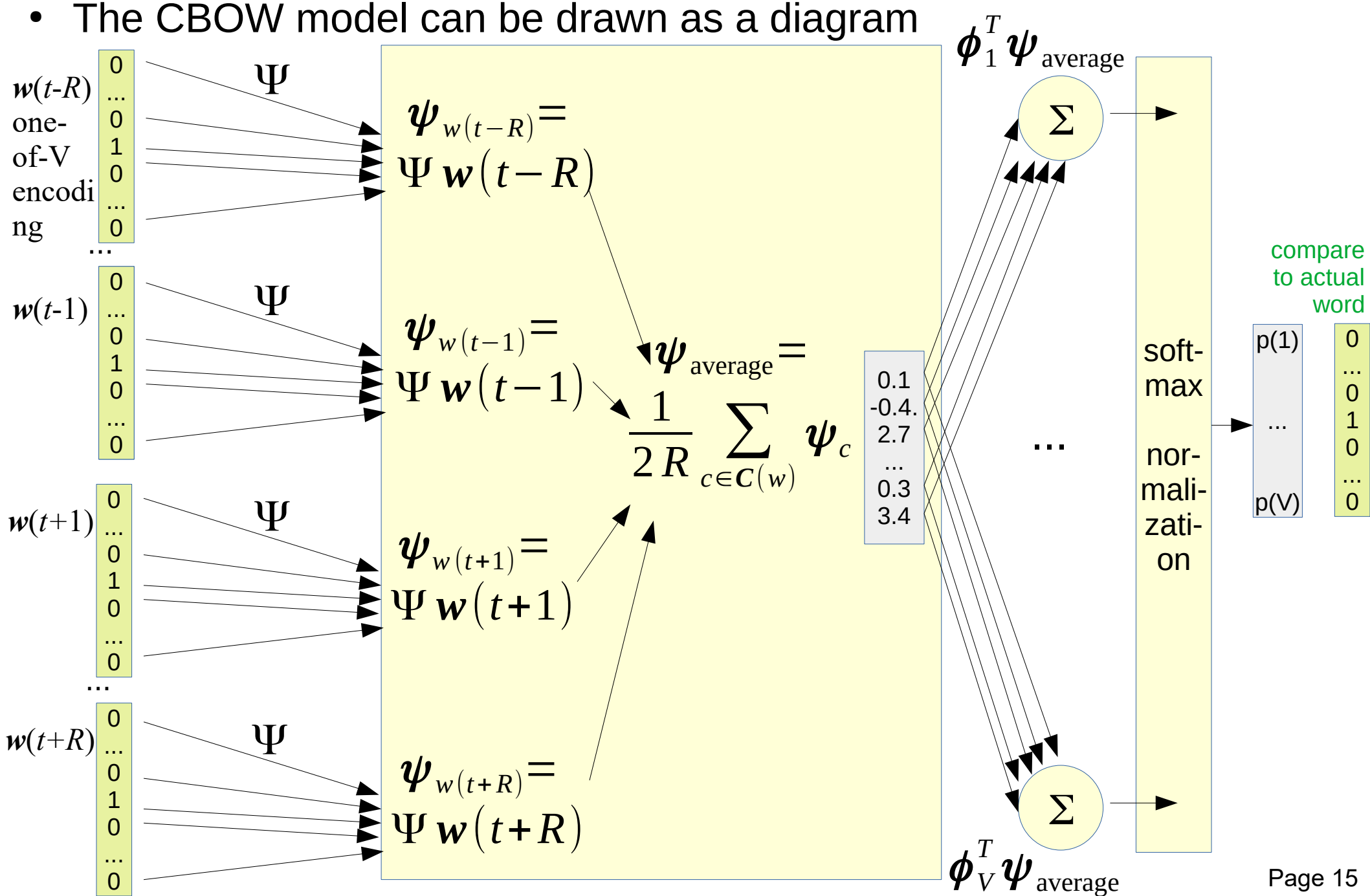
$$\prod_{s=1}^{M} \prod_{i=1}^{N^{(s)}} p\left(w_i^{(s)} | \boldsymbol{C}\left(w_i^{(s)}\right); \boldsymbol{\theta}\right)$$

The parameters $\boldsymbol{\theta}$ are found by maximizing this corpus "probability"

- Somewhat similar idea to an n-gram model but:
  - considers words both before and after the current one.
  - reuses context-words in ways that do not correspond to a proper decomposition of a joint probability into conditional probabilities

Continuous bag of words (CBOW) model

- The CBOW model can be drawn as a diagram



Skip-gram model

- **Skip-gram model**: a word2vec model for probabilities of words w and their contexts c.
- For each word w appearing in some document, the contexts are a set $\boldsymbol{C}(\boldsymbol{w})$ of surrounding words in a window ("context-words").
  - Mikolov et al.: choose a window size R uniformly from 1,...,$\mathrm{R}_{\mathrm{Max}}$ (they use $\mathrm{R}_{\mathrm{Max}}$=10), then $\boldsymbol{C}(\boldsymbol{w})$ contains the R words to the left of w and the R words to the right of w.
- The context-words $c \in \boldsymbol{C}(w)$ are treated as *different kinds of objects* than the words w
- The skip-gram models the probability of an individual context-word c given the word w : $p(c|w;\boldsymbol{\theta})$
- The contexts $c \in \boldsymbol{C}(w)$ are assumed to be independent given w

Skip-gram model

- The "probability" of all contexts in a corpus with M documents is written as

$$\prod_{s=1}^{M}\prod_{i=1}^{N^{(s)}}\prod_{c_j\in C\left(w_i^{(s)}\right)} p\left(\boldsymbol{c_j}|w_i^{(s)};\boldsymbol{\theta}\right)$$

where $w_i^{(s)}$ is the word at position i in document s

- Somewhat similar idea to an n-gram model but:
  - considers words both before and after the current one.
  - contexts of words can be overlapping, so some contexts have several probabilities in the above equation, corresponding to different central words w. This is not really a correct treatment of observations...
- The parameters $\boldsymbol{\theta}$ are found by maximizing this corpus "probability"
- First we have to define the context-word probability

Skip-gram model

- Suppose the context-words c are indices into a discrete vocabulary of U possible context-words: $c \in \{1, \ldots, U\}$. This can be the same vocabulary as the vocabulary of the words: $w \in \{1, \ldots, V\}$, or a separate one.
- The conditional probability of the occurrence of a particular context-word can be modeled as a "softmax" normalization:

$$p(c|w;\boldsymbol{\theta}) = \frac{e^{\text{affinity}(w,c)}}{\sum_{c'=1}^{U} e^{\text{affinity}(w,c')}}$$

where $\text{affinity}(w,c)$ is any function of the word *w* and the context-word *c*

- The affinity can be any function that takes any (positive, negative or zero) values. Higher affinity yields larger conditional probability.
- This part is not yet an assumption: any finite discrete probability distribution over U choices can be written in a similar way as

$$p_c = \frac{p_c}{\sum_{c'=1}^{U} p_{c'}} = \frac{e^{\log p_c}}{\sum_{c'=1}^{U} e^{\log p_{c'}}} = \frac{e^{\log p_c + Const}}{\sum_{c'=1}^{U} e^{\log p_{c'} + Const}}$$

where Const is any constant.

Skip-gram model

- The skip-gram model assumes each word has a feature representation as a d-dimensional vector $\boldsymbol{\phi}_w$, and each context-word has a feature representation as a d-dimensional vector $\boldsymbol{\psi}_c$, and the affinity is $\text{affinity}(w,c)=\boldsymbol{\phi}_w^T\boldsymbol{\psi}_c$
- The vectors of all words and all context-words in their vocabularies are the skip-gram parameters: $\boldsymbol{\theta}=\left(\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_V\},\{\boldsymbol{\psi}_1,\ldots,\boldsymbol{\psi}_U\}\right)$
- The parameters are found by maximising the corpus probability:

$$\boldsymbol{\theta}=\text{argmax}_{\boldsymbol{\theta}}\prod_{s=1}^{M}\prod_{i=1}^{N^{(s)}}\prod_{c_j\in\boldsymbol{C}(w_i^{(s)})}p\left(\boldsymbol{c_j}|w_i^{(s)};\boldsymbol{\theta}\right)$$

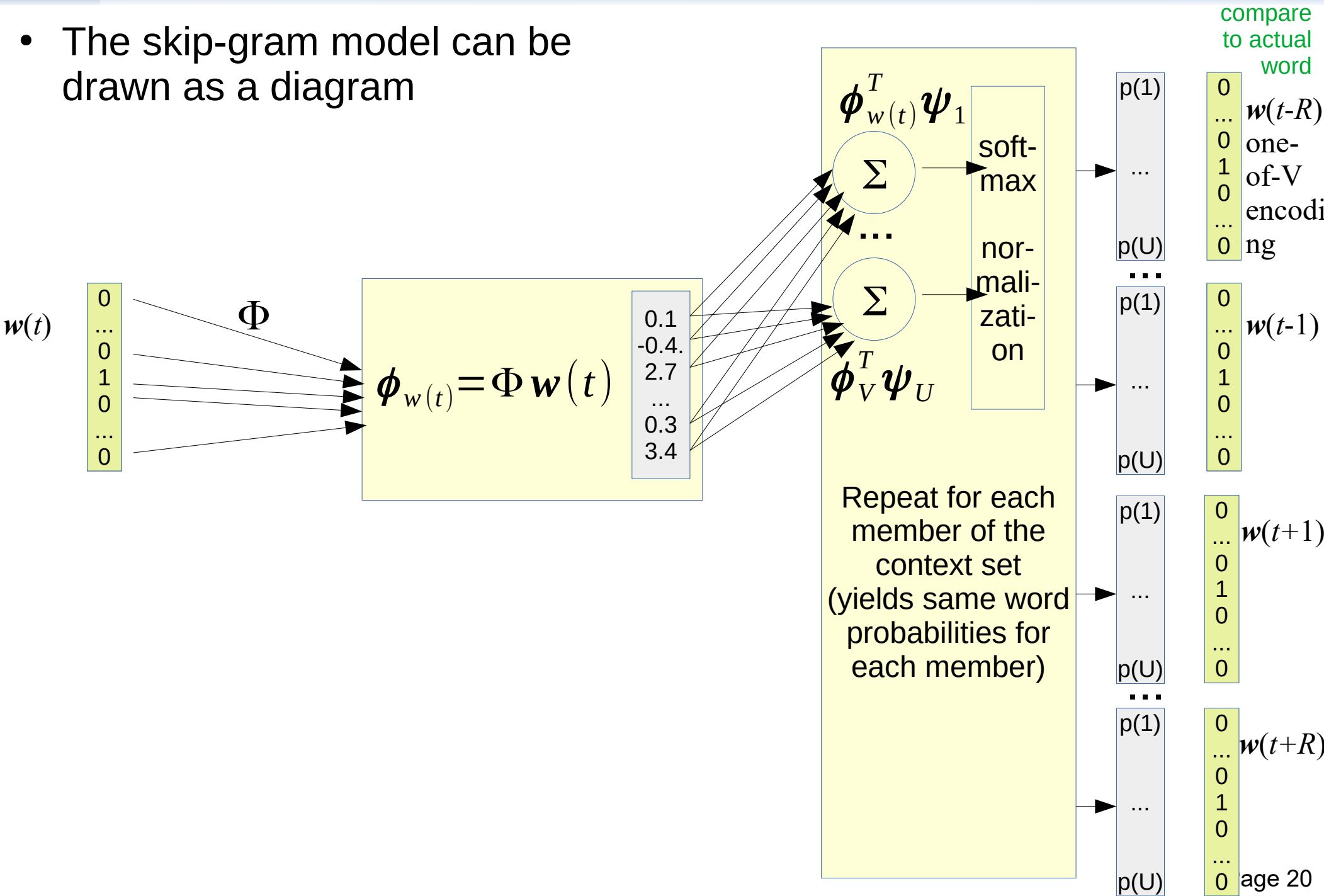
$$=\text{argmax}_{\boldsymbol{\theta}}\prod_{s=1}^{M}\prod_{i=1}^{N^{(s)}}\prod_{c_j\in\boldsymbol{C}(w_i^{(s)})}\frac{e^{\boldsymbol{\phi}_w^T\boldsymbol{\psi}_c}}{\sum_{c'=1}^{U}e^{\boldsymbol{\phi}_w^T\boldsymbol{\psi}_{c'}}}$$

- Or usually its logarithm:

$$\boldsymbol{\theta}=\text{argmax}_{\boldsymbol{\theta}}\sum_{s=1}^{M}\sum_{i=1}^{N^{(s)}}\sum_{c_j\in\boldsymbol{C}(w_i^{(s)})}\left(\boldsymbol{\phi}_w^T\boldsymbol{\psi}_c-\log\left(\sum_{c'=1}^{U}e^{\boldsymbol{\phi}_w^T\boldsymbol{\psi}_{c'}}\right)\right)$$

Skip-gram model

- The skip-gram model can be drawn as a diagram



"Negative sampling" based Skip-gram model

- The previous skip-gram equations require going over all context-word possibilities for every occurrence: can be slow for large vocabularies.
- The "negative sampling" approach is an alternative formulation (not really the same model, although it can be seen as an approximation!).
  Tomas Mikolov, 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 (NIPS 2013). https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html
- The approach is an example of "noise contrastive estimation", which supposes that a good model shoult be able to tell apart data from noise.
- For a pair (w,c) of a word and a context-word, consider a binary classification task: what is the probability that the pair **arose from the same distribution as the training data** (class = "Corpus") vs. that it **arose from noise** (i.e. class = "Noise").
- We only have positive examples (pairs that exist in the corpus), however, we can suggest a distribution for noise.

"Negative sampling" based Skip-gram model

- Denote the conditional probability that a pair (w,c) belongs to the "Corpus" class as $p(\text{Corpus}|(w,c);\boldsymbol{\theta})$
- The model assumes that the word and context-word have feature vectors, and assumes that the class probability is a logistic probability based on their inner product:

$$p(\text{Corpus}|(w,c);\boldsymbol{\theta}) = \frac{1}{1+\exp(-\boldsymbol{\phi}_w^T \boldsymbol{\psi}_c)}$$

Larger inner product yields larger probability the pair is from the corpus

- The probability of the other class (noise) is then

$$p(\text{Noise}|(w,c);\boldsymbol{\theta}) = 1 - p(\text{Corpus}|(w,c);\boldsymbol{\theta}) = \frac{1}{1+\exp(\boldsymbol{\phi}_w^T \boldsymbol{\psi}_c)}$$

- Suppose we can define a generative distribution of "noise" pairs (w,c), so for each word w we can generate "noise" contexts c'.

"Negative sampling" based Skip-gram model

- Suppose for each positive pair (w,c) in the corpus, we generate K pairs (w,c') from the noise distribution. The **expected log-likelihood** (=mean over possible noise samples) of this augmented data set is

$$\sum_{(w,c)\in Corpus}\left(\log p(\text{Corpus}|(w,c);\boldsymbol{\theta})+\sum_{i=1}^{K}E_{c'\sim p_{Noise}(c'|w)}\log p(\text{Noise}|(w,c');\boldsymbol{\theta})\right)$$

$$=\sum_{(w,c)\in Corpus}\left(-\log(1+\exp(-\boldsymbol{\phi}_w^T\boldsymbol{\psi}_c))-\sum_{i=1}^{K}E_{c'\sim p_{Noise}(c'|w)}\log(1+\exp(\boldsymbol{\phi}_w^T\boldsymbol{\psi}_c))\right)$$

- Mikolov et al. suggest to use $p_{Noise}(c|w)=\frac{p_{Corpus}(c)^{3/4}}{\sum_{c=1}^{U}p_{Corpus}(c')^{3/4}}$ where $p_{Corpus}(c)$ is the unigram probability estimate (i.e. relative frequency of c in corpus).

Word2vec in Python

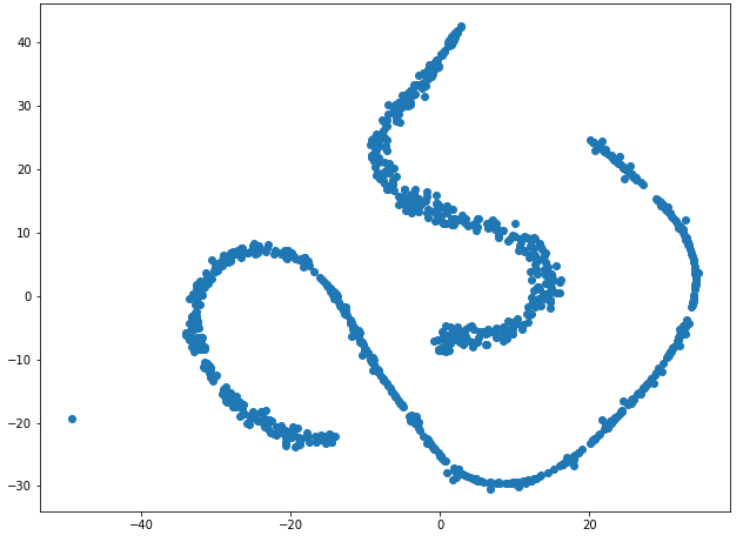
- The Gensim library includes a word2vec implementation

```
import gensim
# Create a dictionary from the documents
gensim_docs=mycrawled_prunedtexts
gensim_dictionary = gensim.corpora.Dictionary(gensim_docs)
# Training algorithm: set to 0 for CBOW, 1 for skip-gram
trainingalgorithm=0
# Train the word2vec model
word2vecmodel = gensim.models.word2vec.Word2Vec(sentences=gensim_docs, \
    size=100, window=5, min_count=1, workers=4, sg=trainingalgorithm)
word2vecmodel.wv['developer']
word2vecmodel.wv.similarity('developer','micro')
Out[163]: 0.99988717
word2vecmodel.wv.similarity('developer','politics')
Out[164]: 0.9990227
word2vecmodel.wv.similar_by_word("developer")
Out[165]:
[('micro', 0.9998870491981506),
 ('manipulate', 0.9998836517333984),
 ('comparable', 0.9998786449432373),
 ('limitation', 0.9998782873153687),
 ('complicate', 0.9998737573623657),
 ('macro', 0.9998735189437866),
 ('specification', 0.9998718500137329),
 ('gui', 0.9998717904090881),
 ('detection', 0.9998699426651001),
 ('dedicate', 0.9998670816421509)]
```

Word2vec in Python

- Visualization like t-SNE can be used to inspect the word2vec resulting vectors

```
# Collect vectors of all words in the vocabulary
word2vec_allvectors=numpy.zeros((len(remainingvocabulary),100))
for i in range(len(remainingvocabulary)):
    tempvector=word2vecmodel.wv[remainingvocabulary[i]]
    word2vec_allvectors[i,:]=tempvector
# Take a random subset of 1000 words
wordsubsetindices=numpy.random.permutation(len(remainingvocabulary))
# Create and plot a 2D t-SNE visualization
tsneplot3 = tsnemodel.fit_transform(\
    word2vec_allvectors[wordsubsetindices[0:1000],:])
myfigure, myaxes = matplotlib.pyplot.subplots();
myaxes.scatter(tsneplot3[:,0],tsneplot3[:,1]);
```



This looks suspicious:
indicates that word similarities form 1-2 continuous sequences.

(Maybe the problem is that we trained with entire models and not individual sentences?)

Word2vec in Python

- Gensim comes with several pretrained word embedding results (these take a lot of disk space, each can be over 1Gb!)

```
# Load an example pretrained word embedding result
import gensim.downloader
word2vec_pretrainedvectors = gensim.downloader.load(\
    'word2vec-google-news-300')
# Inspect the result
word2vec_pretrainedvectors.similar_by_word("developer")
Out[212]:
[('developers', 0.7661452312469482),
 ('Developer', 0.6808924884796143),
 ('Developers', 0.6242722728729248),
 ('Jack_Antaramian', 0.5900515913963318),
 ('vintner_Fess_Parker', 0.5850045084953308),
 ('redeveloper', 0.5840520858764648),
 ('Aldar_Abu_Dhabi', 0.5792878866195679),
 ('Craig_Ustler', 0.5788025856018066),
 ('Panda_Software_www.pandasoftware.com', 0.5760154724121094),
 ('Carl_Paladino_wealthy', 0.5742105245590210)]
```

Word2vec in Python

- A continuous-bag-of-words (CBOW) word2vec model can be used to predict the central word based on a given context

```
word2vecmodel.predict_output_word(['orthogonal','variance'], topn=10)
Out[218]:
[('reverse', 5.128806e-05),
 ('stl', 4.939698e-05),
 ('ot', 4.9375514e-05),
 ('neutral', 4.9210972e-05),
 ('carter', 4.9023565e-05),
 ('chi', 4.889668e-05),
 ('winner', 4.8874033e-05),
 ('bond', 4.8860038e-05),
 ('flow', 4.884721e-05),
 ('saturn', 4.8747846e-05)]
```

- Word2vec models can also be integrated in other (neural) language models in various ways
- There are extensions that embed larger entities: in particular, embedding paragraphs or documents (doc2vec):
  Quoc V. Le, Tomas Mikolov. Distributed Representations of Sentences and Documents. ArXiV, 2014. https://arxiv.org/abs/1405.4053v2
- To be discussed in next lectures