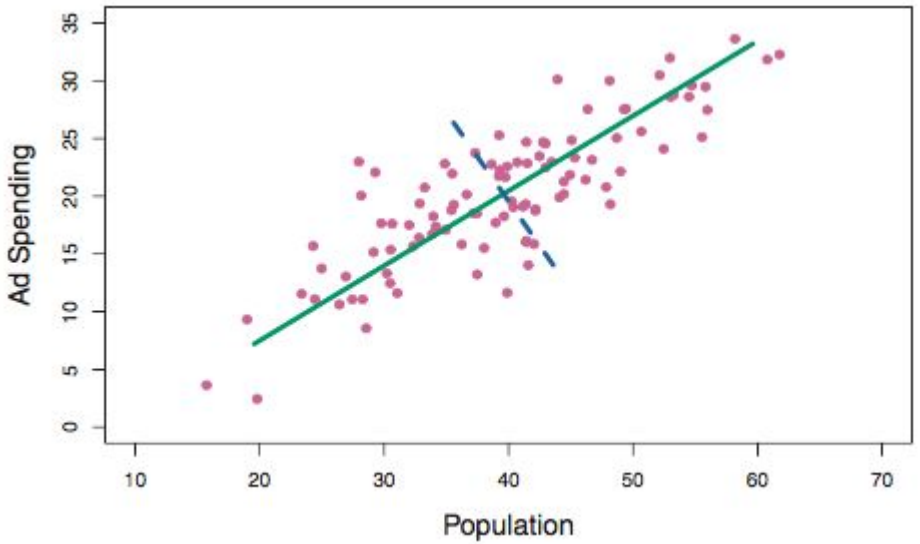
Unsupervised Deep Learning

Outline

- Review linear dimensionality reduction - PCA and SVD
- Autoencoders
- Word Embeddings
- tSNE

PCA & SVD

Principal Components Analysis



- We have observed **p** features for each data point, but we suspect that most of the variation in the data lives within a smaller subspace
- In this example, we have 2 features, but most variation lies along a 1-D subspace
- The directions where the data vary the most are the *principal components*
- The PC are linear combinations of the input feature space
- The reduced PC space represents a *new coordinate system* for our data

Principal Components Analysis

We will have learned a new coordinate system, where each dimension of that system is a weighted linear combination of our original input features.

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \ldots + \phi_{p1} X_p$$

Finding the first Principal Component is akin to finding the direction the direction that would maximize the variance in the new representation.

$$\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} \phi_{j1}^2 = 1.$$

Subsequent PCs are found by repeating this procedure with the residuals

Principal Components Analysis

More simply, PCs be computed the (sorted) eigenvalues of the covariance matrix of X

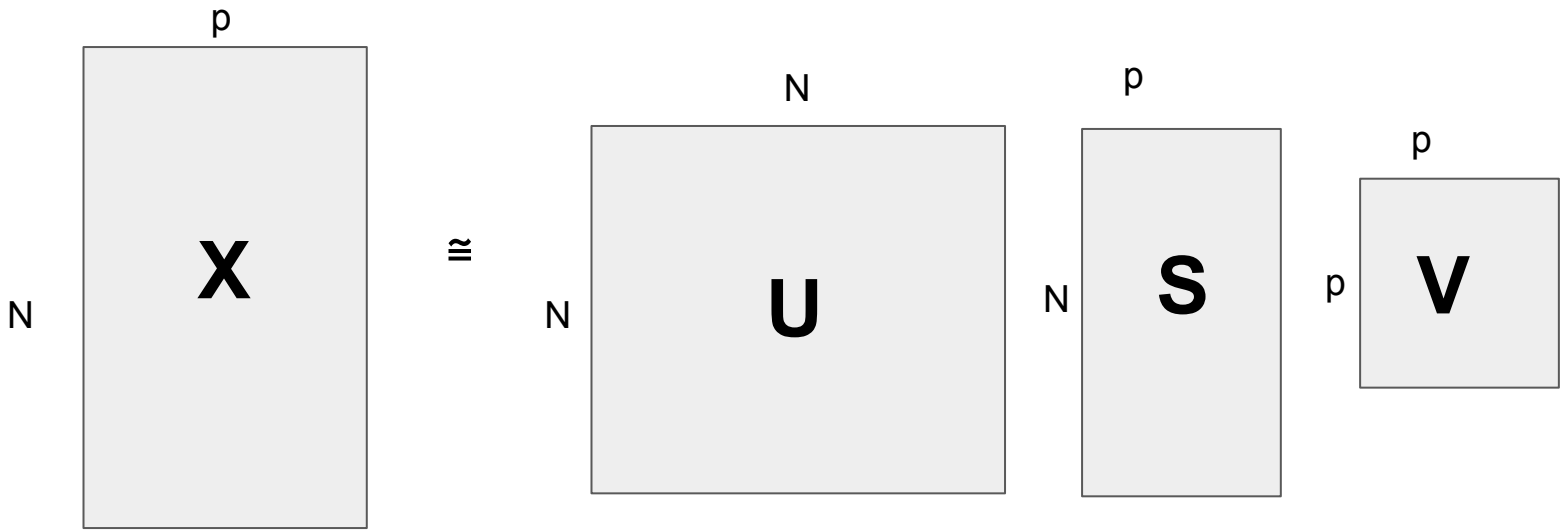
$$\mathbf{C} = \mathbf{X}^T\mathbf{X}$$

$$\mathbf{C} = Q\Lambda Q^{-1}$$

- If we have p input features, PCA will recover p principal components
- The components are typically listed in the order of how much variation in the data is explained
- Hopefully, we can select only the first few PCs which capture most of the variance in the data
- In this way, we get a dimensionality reduction by finding a reduced coordinate system that is aligned with important directions of variation in the data.

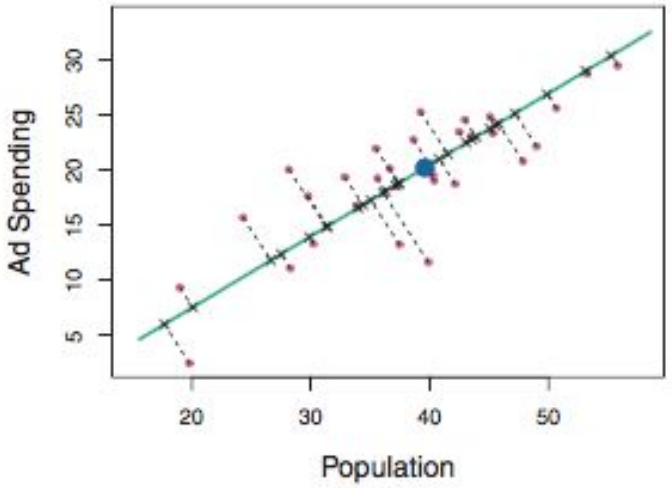
Principal Components Analysis

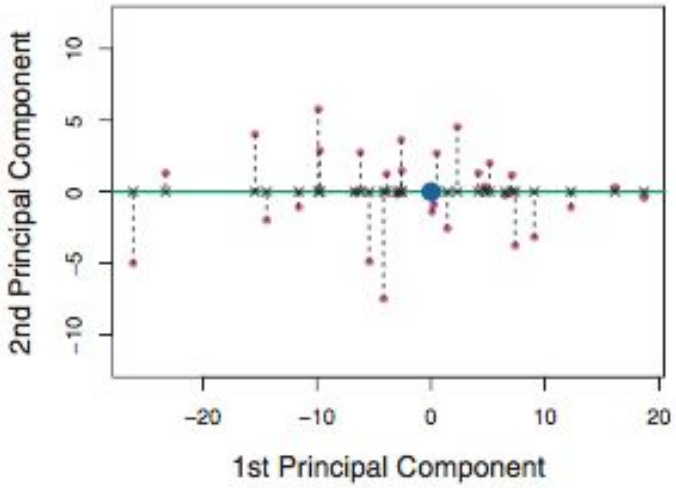
Equivalently, can be computed from the singular vector decomposition of the data matrix X. The right singular vectors of X (matrix V) are the eigenvectors of the X's covariance matrix.



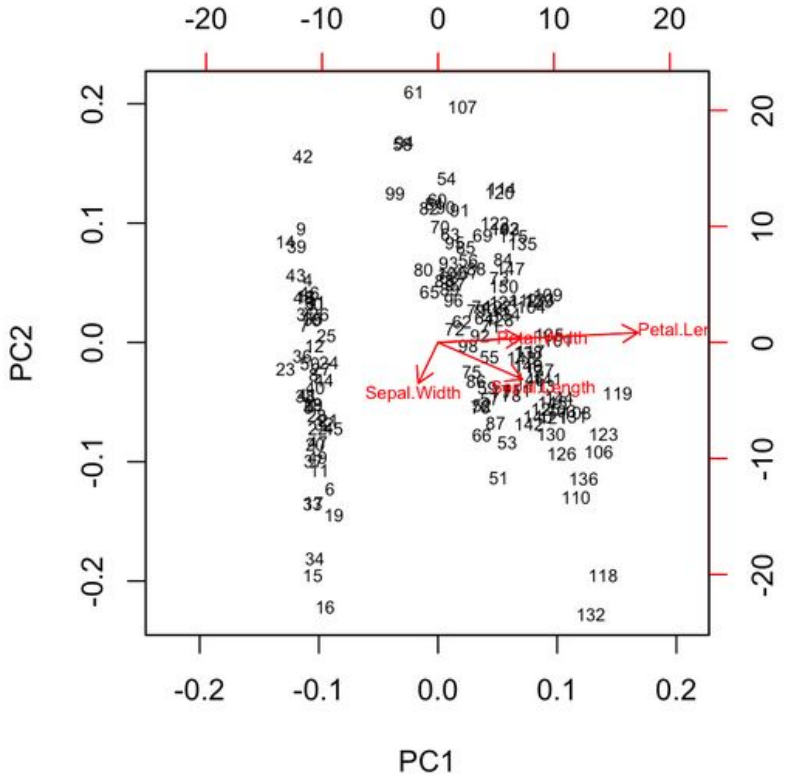
$$\mathbf{X} \approx \mathbf{U}\mathbf{S}\mathbf{V}^T$$

Principal Components Analysis





Principal Components Analysis



- Visualizing the observations in the new (reduced) coordinate system
- We can also see how the original features align with each principal component (loadings)

PCA Summary

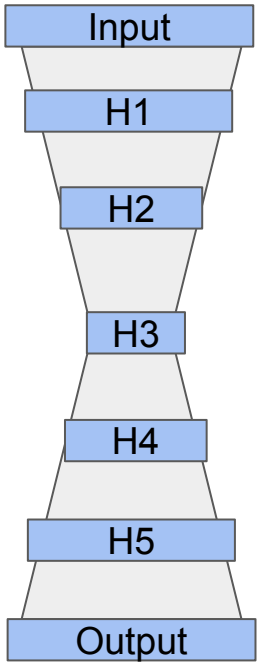
- Linear dimensionality reduction
- Finds new coordinate system that is aligned with directions of high variation in the data
- The new representation is a weighted linear combination of the input features
- Always a useful baseline

Autoencoders

Autoencoders

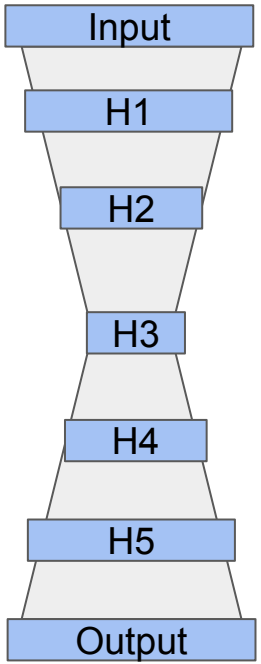
- A neural network whose output is supposed to be similar to its input
  - Input a datapoint, undergo some transformations, then **reconstruct** the original datapoint
- Loss function is some measure of the **reconstruction error** between the input observation and the output vector
- Completely unsupervised - just recreating the input

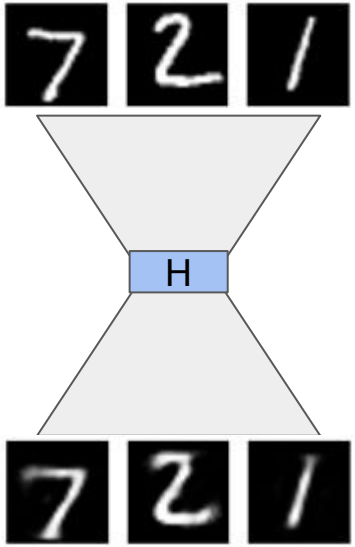
Autoencoders

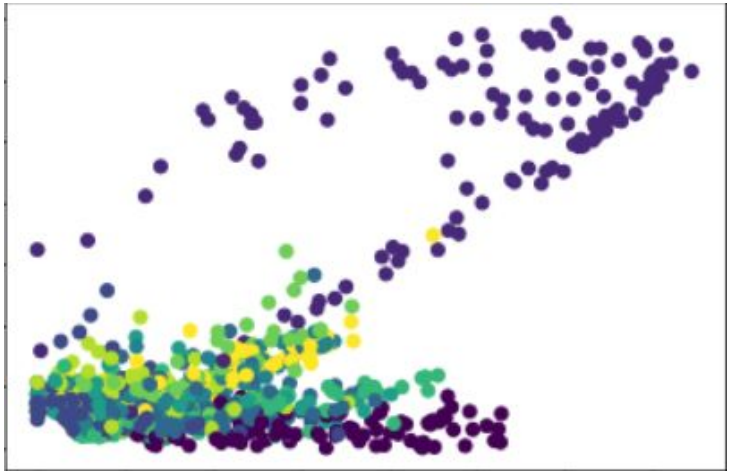


- A deep network (any number of layers)
- Input layer and output layer must have have the same size (are intended to represent the same objects)
- There is middle layer which the *smallest* size (the belt, or squeeze layer)
- If the output can accurately reconstruct the input, then the representation in the middle layer is a sufficient low-dimensional transformation of the data

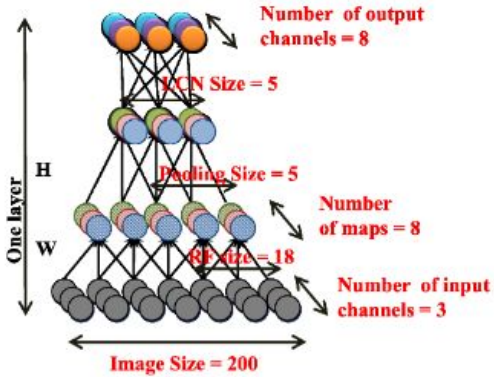
Autoencoders







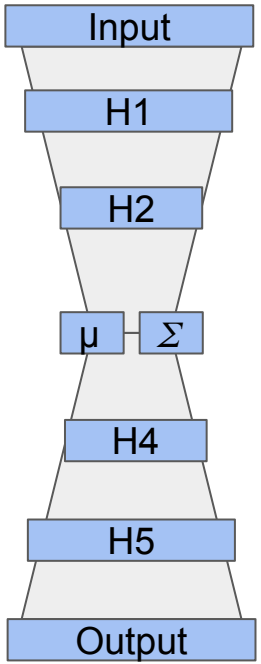
“Building High-Level Features Using Large Scale Unsupervised Learning”



- 2014 paper from Google
- Trained an autoencoder on every frame of every video in YouTube
- Can explore the learned feature representations (face detectors, cat detectors) that are extracted in a completely unsupervised fashion

https://static.googleusercontent.com/media/research.google.com/en//archive/unsupervised_icml2012.pdf

Variational Autoencoders



- Similar structure except we place constraints on our embedding (middle) layer
- We want the representations in the embeddings to follow some familiar probability distribution (such as multivariate Gaussian)
- In the middle, our network predicts the mean and variance of Gaussian distribution
- Why is this useful?

https://towardsdatascience.com/what-a-disentangled-net-we-weave-representation-learning-in-vaes-pt-1-9e5dbc205bd1

Exercise

- See the accompanying notebook and try your hand at training autoencoders

Word Embeddings

Intro to neural language modeling

Outline

1. Key ideas in natural language
2. Classic natural language ideas
3. Word embeddings
4. Neural word embeddings
    a. GloVe
    b. Word2Vec + Skipgram
5. Practicalities
6. Additional resources

What is one-hot encoding (OHE)?

OHE is the basis of classic ML in NLP

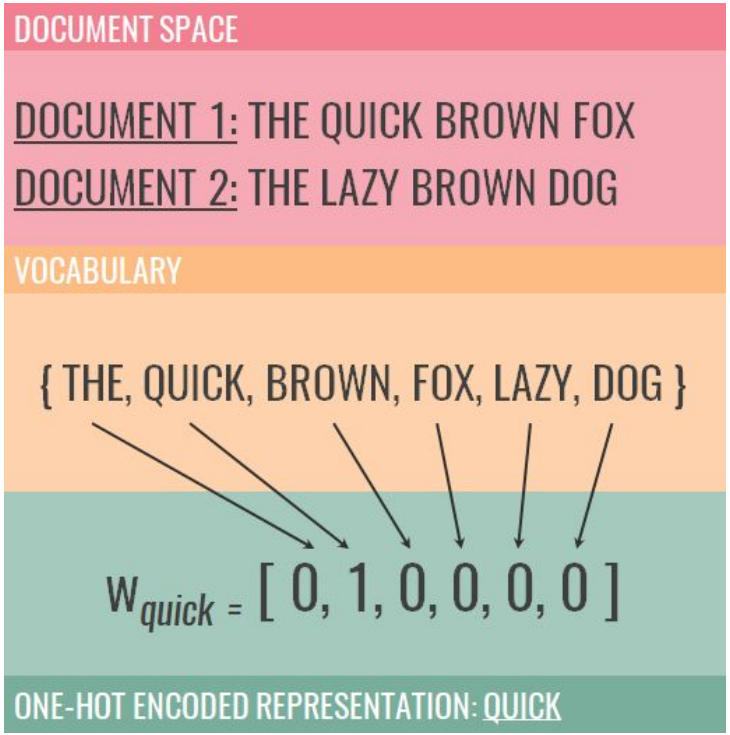
It is a method that converts categorical data into a numeric vector that computers can understand and is easy to interpret

The way it works is by constructing a *V* length vector which is the length of the vocabulary (and where each index represents a word)

For a document that has a word a 1 is assigned to that index in the vector

One hot vectors can be very large depending on the vocabulary

Frequently it is used with the “hashing trick” and “sparse encoding” to reduce the vector size

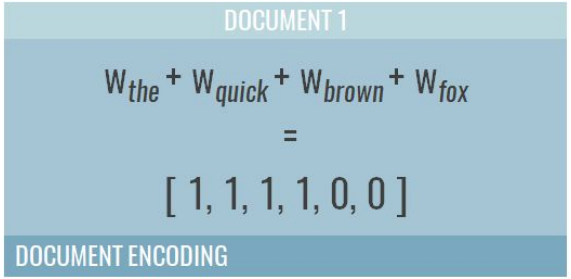


A simple language models uses matrices to represent relations among words/documents

- Language can be represented through:
  - Term-document (TDMs) or
  - Document-term matrices (DTMs - transpose of TDMs)
- A document is the sum of one-hot-encoding word-vectors
- We frequently call this the *bag-of-words representation*

- The relation among documents is based on some metric that assesses the difference/similarity between document vectors

- Models that use DTMs are called vectorial-semantic models (VSMs)



Summing OHE elements gives us a BOW vector

	DOCUMENT 1	DOCUMENT 2
THE	1	1
QUICK	1	0
BROWN	1	1
FOX	1	0
LAZY	0	1
DOG	0	1

TERM DOCUMENT MATRIX

Encoding two different documents - notice that sequence is obfuscated in BOW

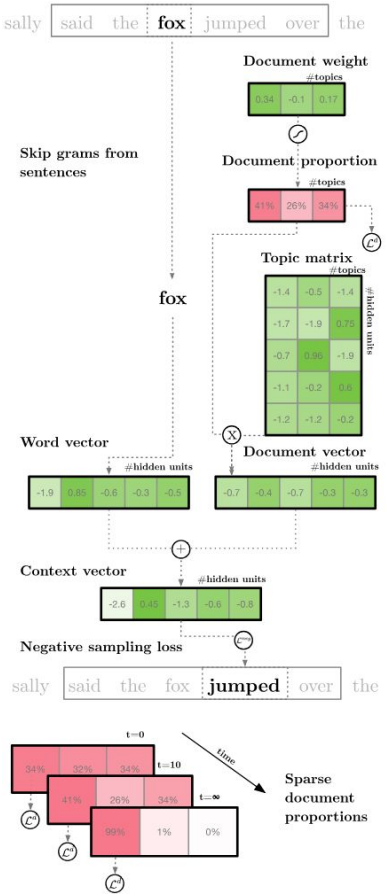
Classic ML NLP approaches remain important

Machine learning methods that use DTMs include:

- Search
- Topic/document clustering
- Word disambugation
- Sentiment analysis
- Automatic key phrase extraction

Note:

- Some recent developments use a mixture of classic models and neural embeddings such as LDA2Vec
- Many deep NLP methods build on foundational concepts in NLP
- Studying classic approaches and their problems gives you insight into how neural approaches resolves them



*** Source: Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec*
https://www.groundai.com/project/mixing-dirichlet-topic-models-and-word-embeddings-to-make-lda2vec/

What are some OHE of limitations?

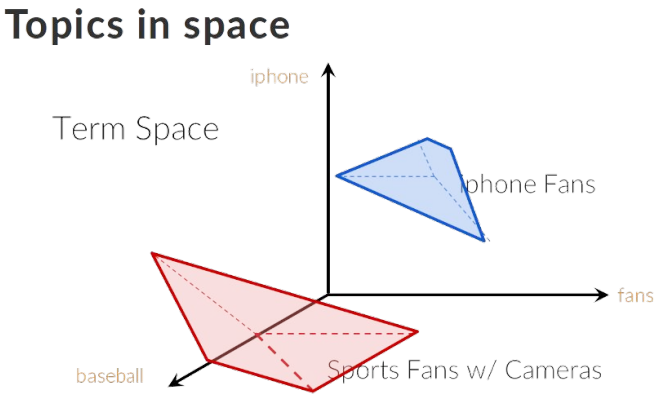
OHE does not consider relations among words

OHE is used to measure statistical properties across phrases and corpora in extremely high dimensions

By definition OHE assumes words are orthogonal to one another

- For example the words: ***pencil*** and ***pen*** are independent of one another although they are both are writing instruments
- Relations among words only surface as a result of the statistical similarities among global structure in documents
- This is a central limitation since local information about sequence and position inform the relation among words

Pencil and pen are treated as orthogonal despite being writing instruments when using OHE



Vector-semantic models: Documents are reconstructed in term space through the addition of OHE, notice that baseball, fans and iphone are orthogonal to one another in 3-space

Reducing Dimensionality of the System

Early attempts to reduce the dimensionality of systems and generate lower dimensional representations of the DTM included SVD

SVD is a generalization of PCA to non-square matrices

It is a decomposition technique that uses “singular” rather than “principal” values to approximate the DTM

Early applications included latent semantic analysis/indexing for recommendations systems (Netflix prize) and is still used in search (Google banners)

While the original reconstruction error was built to optimize user-movie relations, re-focusing on local term-term relations we are able to generate meaningful embeddings and is the basis of GloVe(more to come on this later)

SVD through a user-movie matrix

Singular-Value decomposition:

$$X_{m\times n}=U\Sigma V^T$$

where:

- $U_{m\times d}$ - latent user matrix
- $V^T_{d\times n}$ - latent movie matrix
- $\Sigma_{d\times d}$ - Singular value matrix.

Example: User- Ratings:

Ratings matrix: $X=U_{6\times5}\times\Sigma_{5\times5}\times V^T_{5\times5}$

Reconstruction error using MSE:

$$\text{argmin}||X-X_k||_F = \text{argmin}\ \Sigma_i\Sigma_j(X(i,j)- X_k(i,j))^2$$

$$= \underset{U,V,\Sigma}{\text{argmin}}\Sigma_i\Sigma_j(X(i,j)- U(i,.)\times\Sigma_k\times V(.,j))^2$$

		Alien	Predator	Matrix	Casablanca	Amelie
SCI-FI	USER 1	4	4	3	0	0
	USER 2	5	4	4	0	0
	USER 3	2	4	5	0	0
	USER 4	5	5	3	0	0
ROMANCE	USER 5	0	1	0	5	4
	USER 6	0	1	0	4	5

USER-MOVIE MATRIX (X)

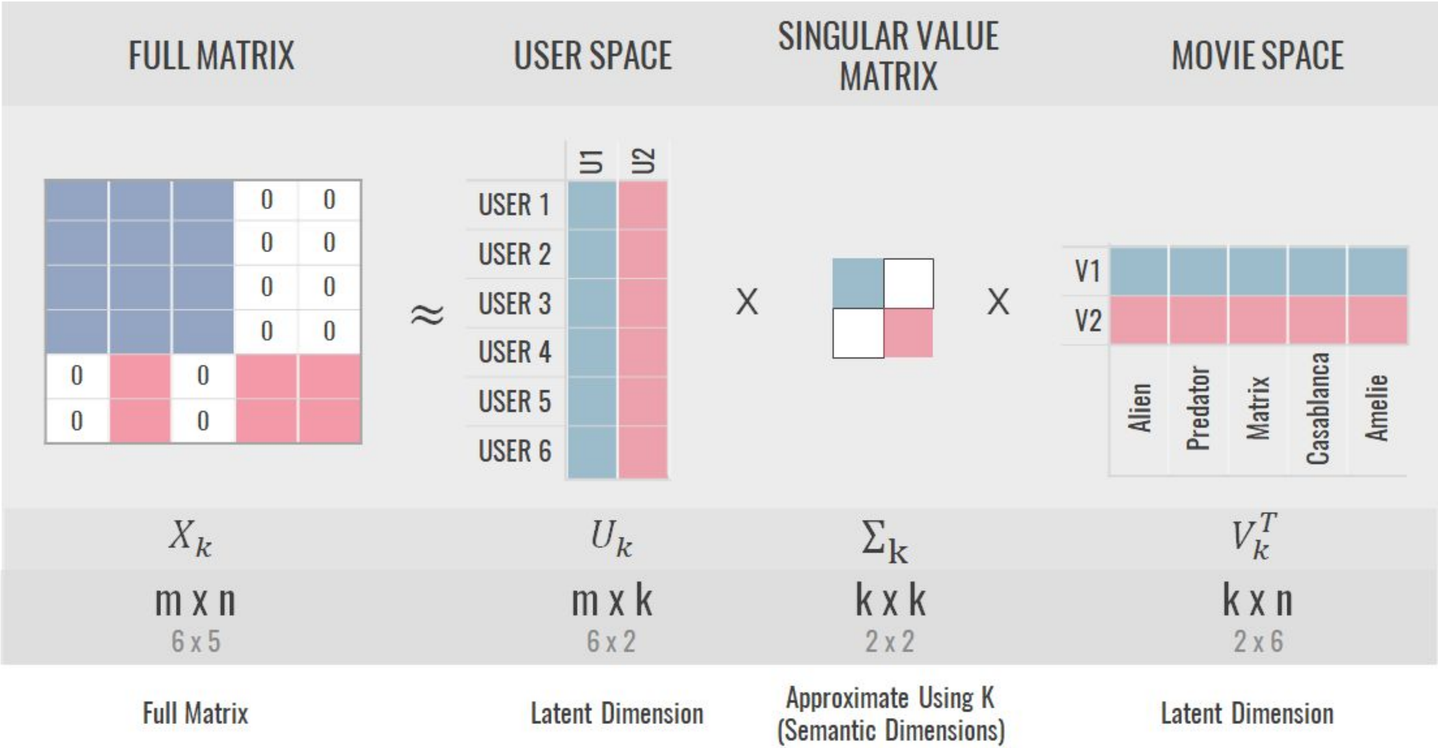
Example: User-Ratings

Clearly there are two genres of movies, along with differential ratings for users.

How could we decompose this matrix to provide recommendations and approximate its reconstruction like in PCA?

We can't use PCA because the matrix isn't square so we use SVD

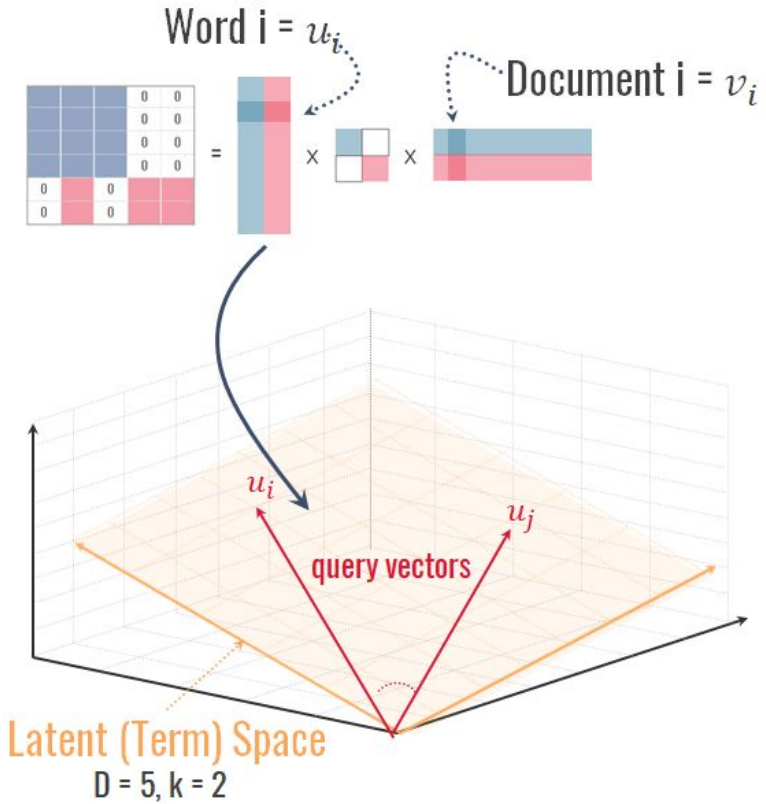
SVD in Pictures - Lower dimension user space



The rows in the user space U_2 and columns in V^T_2 exist in R

Figure 2: X_k approximates X where k=2.

Measuring similarity in semantic space



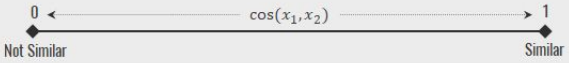
SIMILARITY CALCULATION

Term:

$$\text{sim}(w_i, w_j) = \cos(u_i \Sigma_k, u_j \Sigma_k)$$

Document:

$$\text{sim}(d_1, d_2) = \cos(\Sigma_k v_i, \Sigma_k v_j)$$



$$\cos(x_1, x_2) = \frac{x_1 \cdot x_2}{||x_1|| \times ||x_2||}$$

Recall from linear algebra that this can be viewed as an orthogonal projection of one vector onto another.

Note the cos similarity is a value between -1 and 1.

A value of cos(x1,x2) = 0 indicates the vectors are orthogonal and of -1 that they are in opposite directions

The normalized direction gives us an indication of how close the vectors are through it's angle

An alternate view

Since singular values are distributed in the similarity query between two users or movies, recommendation systems are frequently constructed using only U (user) and V (movie)

The reconstruction error is then written as:

$$\text{argmin}||X - X_k||_F = \underset{U,V}{\text{argmin}} \Sigma_i \Sigma_j (X(i,j) - U(i,.)V(.,j))^2$$

The interpretation views the user and movie ratings as a linear combination

This looks very similar to the minimization problem for NNs

We can use **SGD** to minimize for the values of U and V or **alternating least squares**

Linear model with 2 biases

Penalties on u and v

$$L = \sum_i \sum_j (X(i,j) - \boldsymbol{u}_i^T \cdot \boldsymbol{v}_j + b_i + c_j)^2 + \lambda_u \sum_i \| \boldsymbol{u}_i \|^2 + \lambda_v \sum \| \boldsymbol{v}_j \|^2$$

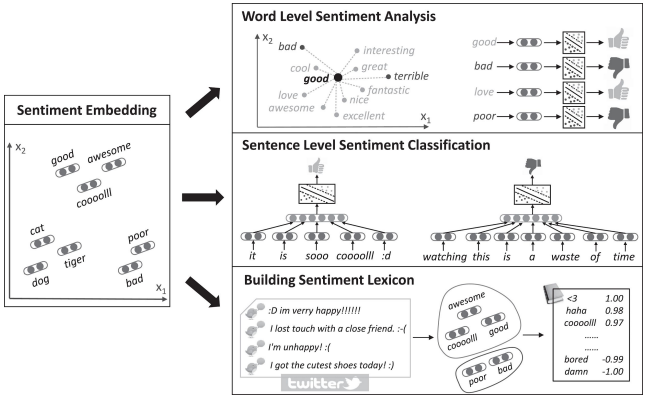
We can also add in bias terms and a regularizers and minimize with respect to U,V

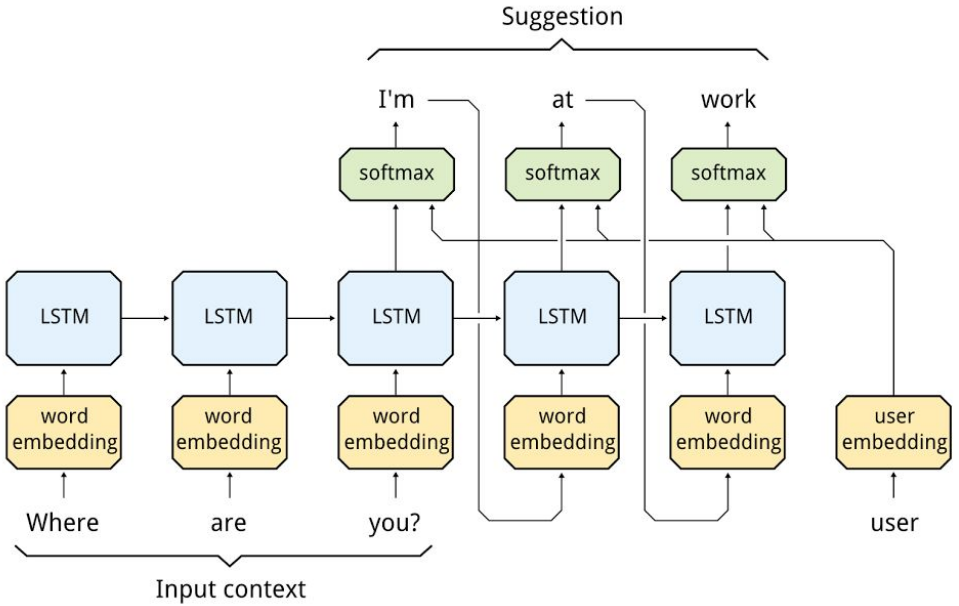
For details on factorization see: https://www.ethanrosenthal.com/2016/01/09/explicit-matrix-factorization-sgd-als/

Word embeddings in the real world

Word embeddings are used in several common applications, including:

1. Feature extraction
2. Sentiment analysis
3. Question answer
4. Translation
5. Anywhere you would use language in your pipeline





*** Source: Chat Smarter with Allo May 18, 2016 Posted by Pranav Khaitan, Google Research*

https://research.googleblog.com/2016/05/chat-smarter-with-allo.html

Why use word embeddings?

Pros of using Embeddings

Word embeddings bridge the “meaning gap”
- Words in relation to one another have meaning and are no longer orthogonal

Reduces the size of the object contained in memory
- Generates a low dimensional (between 50-300) embedding of words
- Using a DTM with 50K words and 10^6 documents would be roughly 190Gigabytes (stored as float32s)

More expressive representation of the data
- Analogical reasoning enables vector math with words
- Composition of word vectors is more interpretable

Caveats when using Embeddings

- Positions in space of word-vectors do not hold significance in that space/dimension
  - Only their relative distances/angles to one another do
- The embeddings are not unique
- There are different embedding approaches two common ones are
  - Probabilistic prediction approaches (predict based on context/center words) and
  - Reconstruction approaches (approximate the DTM)
- Generating good embeddings requires a lot of data and training them is challenging

Embeddings

Instead of approximating the TDM we embed words as a set of vectors in some high dimensional space (although still lower than TDM space)

There is more than enough information in these high dimensions to express the relation among words (usually D=50,100,300 - empirically after 300 dims there is no gain)

On the right, the example, illustrates several type relations, note the angle, direction and trajectory

A common example is:

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

However other like grammar, capital among countries and compositions (like the one above) similarly exist

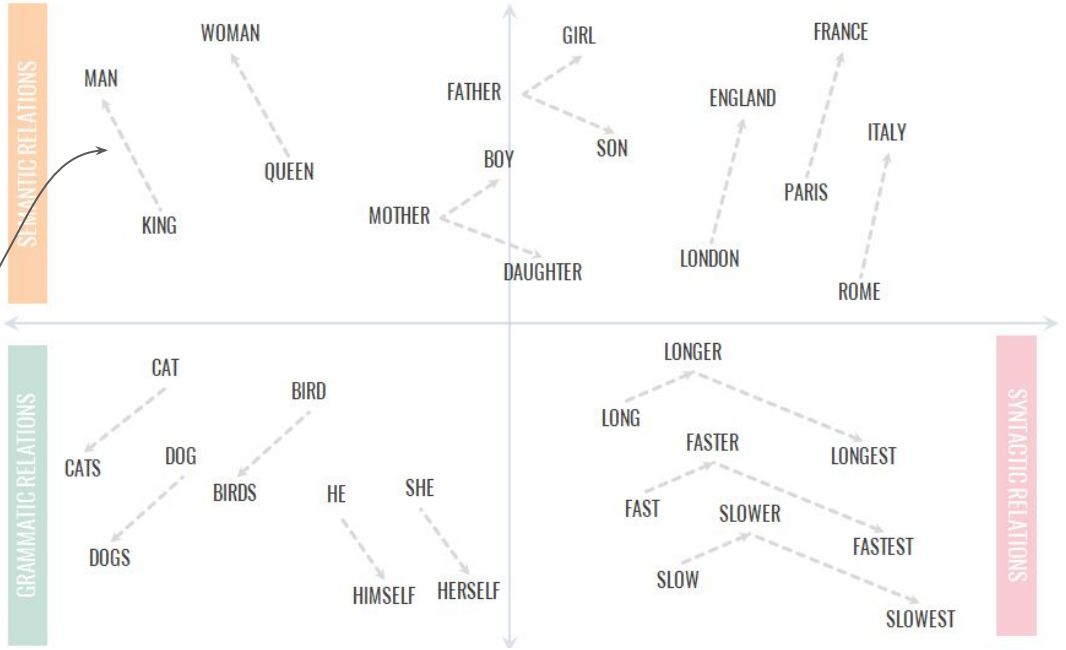


Figure: Example 2D Embedding Space Obtained Through Word2Vec

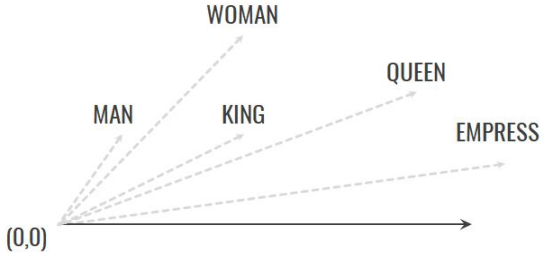
Embeddings enable vector math

The relations among embeddings can be summarized using vector math - summing and subtracting allows us to interpret relations, i.e.:

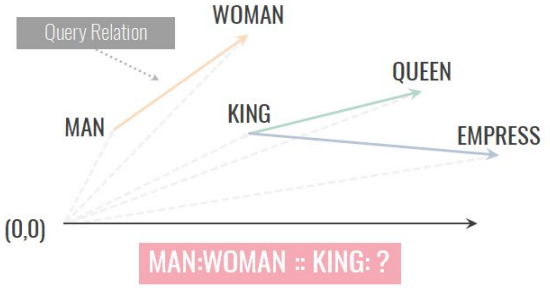
vec(king)−vec(man)+vec(woman)≈vec(queen)

The distances and angles between vectors therefore can be interpreted as “similarity of idea/structure”

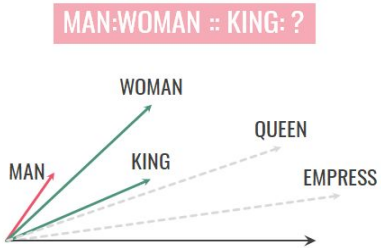
Using this approach we use analogic reasoning to approximate similar relations between pairs of words



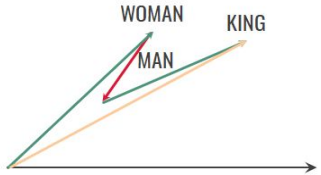
2d Projected Vector Embeddings



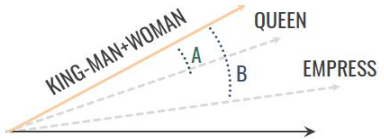
Semantic Relationships



2d Projected Vector Embeddings



$$x_{woman} - x_{man} + x_{king} = x_{king} - x_{man} + x_{woman}$$



$$d = \text{argmax}_i \frac{(x_{woman} - x_{man} + x_{king})^T x_i}{||x_{woman} - x_{man} + x_{king}||}$$

Common Embedding approaches

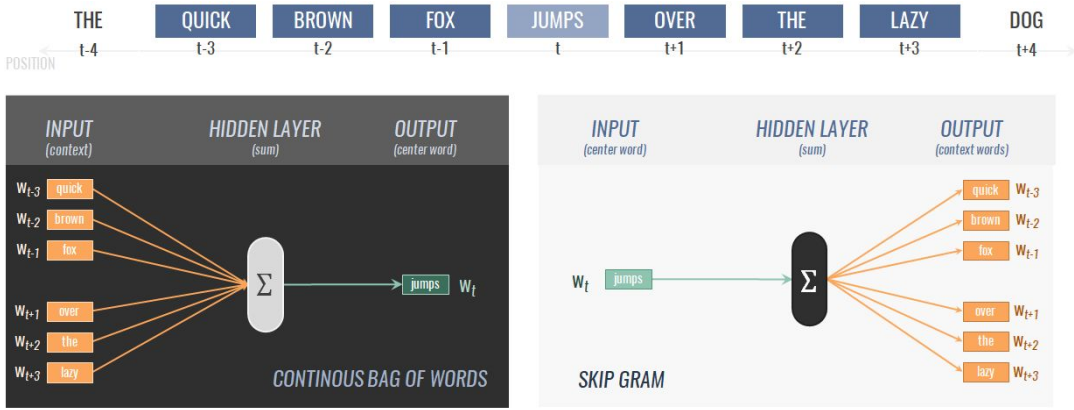
Two common approaches to embedding are:

1. GloVe - Global Vector (reconstructive) and
2. Word2Vec (predictive), which is made of two models
   a. Continuous bag of words and
   b. Skipgram

We'll focus first on GloVe and then discuss Word2Vec

GloVe and Skipgram type models tend to be most used in real world applications

- GloVe - similar idea to rec systems
- Skipgram - easy application



CBOW predicts a center word from context words

Skipgram predicts context words from a center word

GloVe

A reconstructive approach

Uses a weighted term-term matrix (call it X)

X is constructed such that i is the center word and j is the context word

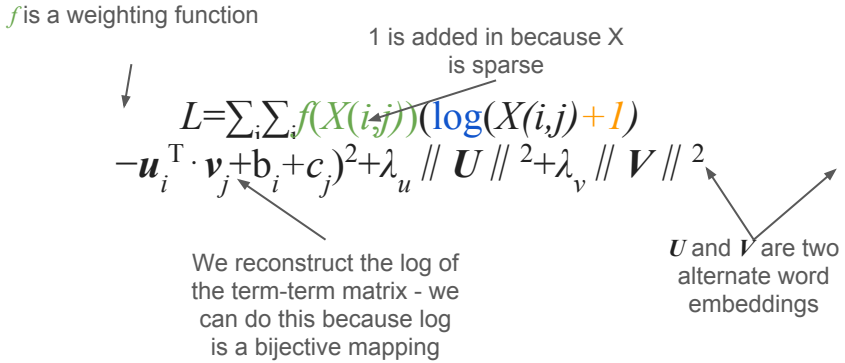
Context distance is calculated as distance from one another

Example: “The quick brown fox jumped”

- X(The, quick) = +1
- X(The, brown) = +½
- X(The, fox) = +⅓

Implications include that X is a sparse matrix and will have a similar to solution to rec system

We minimize the loss below with respect to the weights in vector U, V, b and c . The resulting approach generates a set of embeddings U, V



Setup: Skipgram

Skipgram is part of the Word2Vec probabilistic predictive embedding models

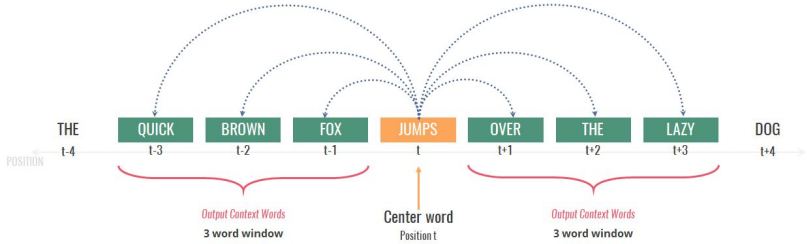
In Word2Vec, local relations provide greater contextual information than global structure

Accordingly two important ideas for Word2Vec models are “center” and “outer context” words

Context is preserved by calculating blocks of probability for surrounding words with the skip gram models

The probability distribution looks like

$$Pr(w_{t-m},...,w_{t-1},w_{t+1},...,w_{t+m}|w_t;\theta)=\prod_{-m\le j\le m;j\neq 0}Pr(w_{t+j}|w_t;\theta)$$



Sliding window (size = 3)	Target word	Context
[**the** quick brown fox]	the	quick, brown, fox
[the **quick** brown fox jumps]	quick	the, brown, fox, jumps
...	...	...
[jumps over the **lazy** dog]	lazy	jumps, over, the, dog
[over the lazy **dog**]	dog	over, the, lazy

The figure and table illustrate center-context word dependence for a window of m=3

Example:
“The quick brown fox jump over the lazy dog”

For a context window of m = 2 and target of jumps we have the following target-context probability for one phrase

$$\prod_{-m\le j\le m;j\neq 0}Pr(w_{t+j}|jumps;\theta)$$
$$=$$
$$Pr(u_{brown}|v_{jumps})Pr(u_{fox}|v_{jumps})Pr(u_{over}|v_{jumps})Pr(u_{the}|v_{jumps})$$

Architecture: skipgram

The skipgram architecture uses two matrices V and U to calculate the embeddings of the words

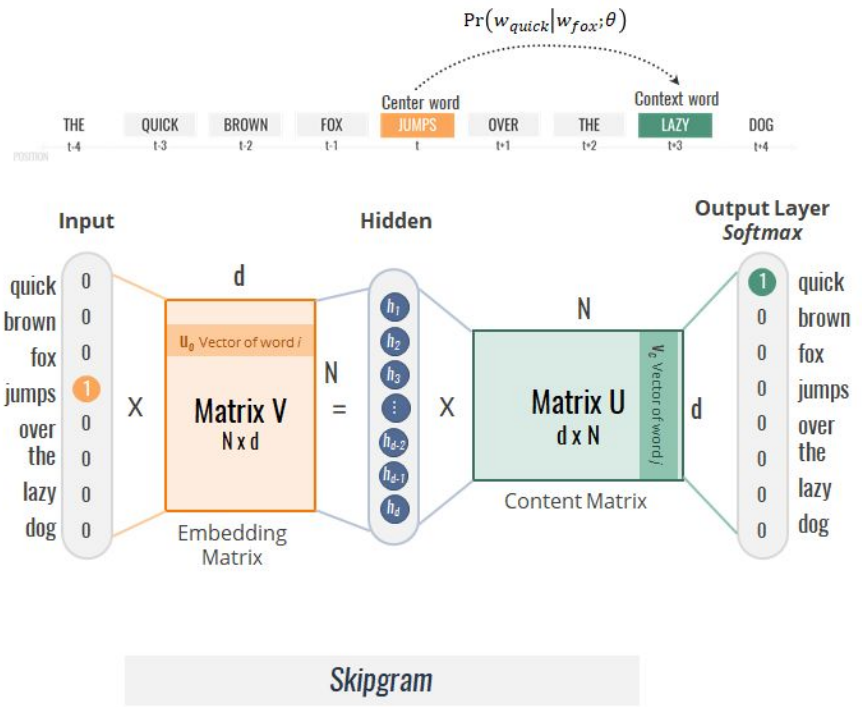
The matrices are optimized against a softmax function with cross entropy error

The input is a word and the output is the set of context words probability

The architecture can be trained using softmax however is very slow, several alternate methods have been proposed including “negative sampling”

<u>Summary:</u>

- <u>Input:</u> Target word
- <u>Output:</u> Context words
- <u>Output layer:</u> softmax
- <u>Loss:</u> Cross entropy



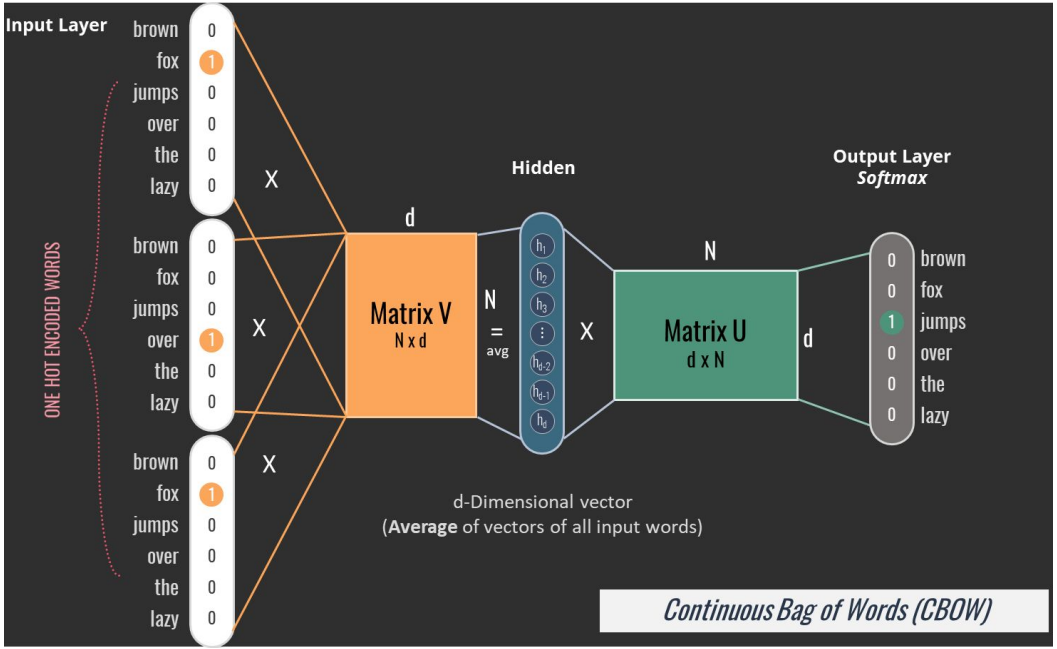
Architecture: CBOW

CBOW has a similar architecture to skipgram however the input and output are flipped

Instead, context words are taken as an input, averaged and then the softmax is calculated for the target (center) word

Summary

- Input: Target word
- Output: Context words
- Output layer: softmax
- Loss: Cross entropy



Several approaches to fitting Word2Vec models exist

	APPROACH	SPEED UP FACTOR	PERFORMANCE (SMALL VOCAB)	PERFORMANCE (LARGE VOCAB)
EXACT	Softmax	1x	Very good	Very poor
	Hierarchical Softmax	25x (50-100x)	Very poor	Very good
	Differentiated Softmax	2x	Very good	Very good
APPROXIMATE	CNN-Softmax	-	-	Bad-good
	Importance Sampling	(19x)	-	-
	Adaptive importance sampling	(100x)	-	-
	Target Sampling	2x	Good	Bad
	Noise Contrastive Estimation	8x (45x)	Very bad	Very bad
	Negative Sampling	(50-100x)	-	-
	Self-Normalisation	(15x)	-	-
	Infrequent Normalisation	6x (10x)	Very good	Good

Several variants exist to fit word embeddings.
Mikolov's seminal paper introduces negative sampling as an alternative to traditional softmax

Using embeddings

Why are neural embeddings preferred:

- Fast (compared to quadratic time scaling),
- Allow users to incorporate a new sentences/documents and
- Can easily add words to the vocabulary

They have two additional benefits:

1. Their performance for common linguistic tasks is superior to DSM methods and
2. They address non-linearities directly through their architecture rather than feature engineering

Practicalities:

- Embedding approaches do not only need to be with words
- Facebook: Startspace - Embed All the Things
- Evaluating embedding quality is challenging
- Embedding only makes sense when you have a lot of data and computational resources
- Most practitioners download pre-trained models and optimize for their network

Yoav Goldberg critique of word embeddings

- Mentions other classical methods that are tuned yield similar results
- Word embeddings are useful for general purpose situations however should not be viewed as the best approach for a particular language problem

Embeddings - what next?

A lot that wasn't covered

We didn't

1. Didn't deep dive into the math
2. Look at training the embedding
    - GloVe's SGD and ALS approaches are really interesting and build on earlier ideas
3. Discuss hyperparameter meanings or optimizations
4. Go into evaluation methods for embeddings
5. Briefly touched on different training methods but there's a lot more

References:

1. Evaluation Methods for Unsupervised Word Embedding - Tobias Schnabel, Igor Labutov, David Mimno, Thorsten Joachims (2015)
2. GloVe: Global Vectors for Word Representation - *Jeffrey Pennington, Richard Socher,Christopher D. Manning* (2014)
3. Efficient Estimation of Word Representations in Vector Space - *Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean* (09/07/2013)

Course notes with more detail can be found at:

Word Embeddings 1- Intro and Background
(*http://joshuahtouyz.com/blog/p1_word_embeddings_introduction*)

Word Embeddings 2- Singular Value Decomposition
(*http://joshuahtouyz.com/blog/p2_word_embeddings_svd*)

Word Embeddings 3- Word2Vec
(*http://joshuahtouyz.com/blog/p3_word_embeddings_word2vec_glove*)

Word Embeddings 4- Evaluating Embeddings
(*http://joshuahtouyz.com/blog/p4_embedding_evaluation_methods*)

Exercise

- See the accompanying notebooks to use pre-trained word embeddings to assist in NLP tasks

tSNE

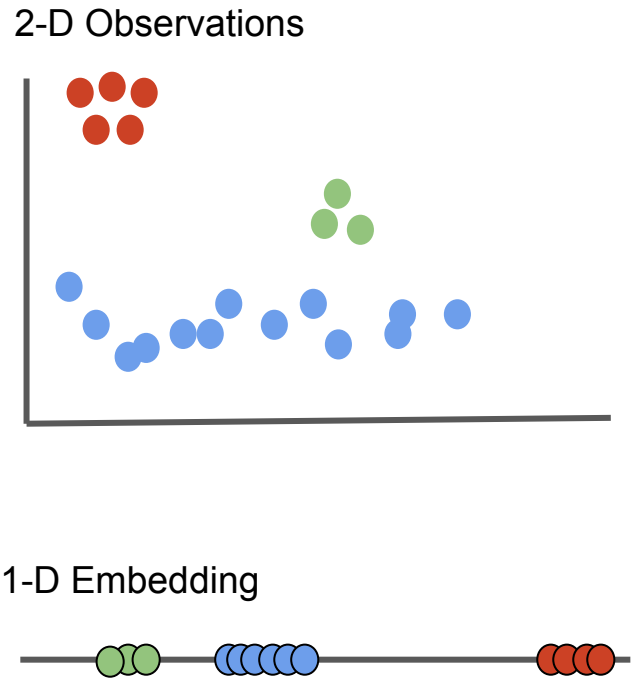
https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

https://distill.pub/2016/misread-tsne/

Low Dimensional Embeddings

Goal: For a set of high-dimensional observations, find a low-dimensional representation where geometric relationships between points in the high-D are preserved in the low-D points.

- Preserve local structure between near points, and global structure between distant groups
- Also known as Manifold Learning



t-Distributed Stochastic Neighbor Embedding

For high-D observations x_i and low-D representations y_i, the probability density $p(x_i)$ over all x_i should be similar to the density $q(y_i)$ over all y_i.

$$p(x_i) \approx q(y_i)$$

$$L = \sum_i \mathrm{KL}(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{i|j}}$$

- By minimizing this loss function, we find a representation that matches the two probability densities optimally.
- We need to pick a functional form of the probability density for our observations

t-Distributed Stochastic Neighbor Embedding

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

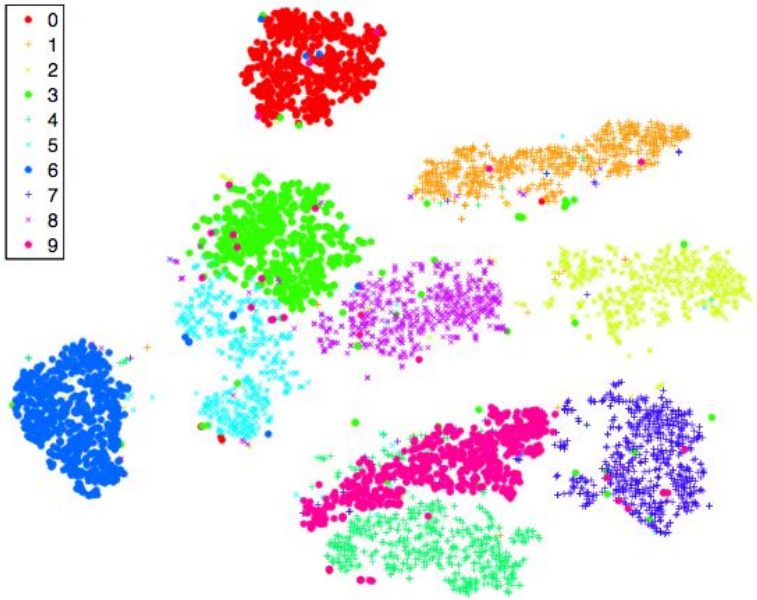
(Something like a Gaussian distribution)

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}.$$

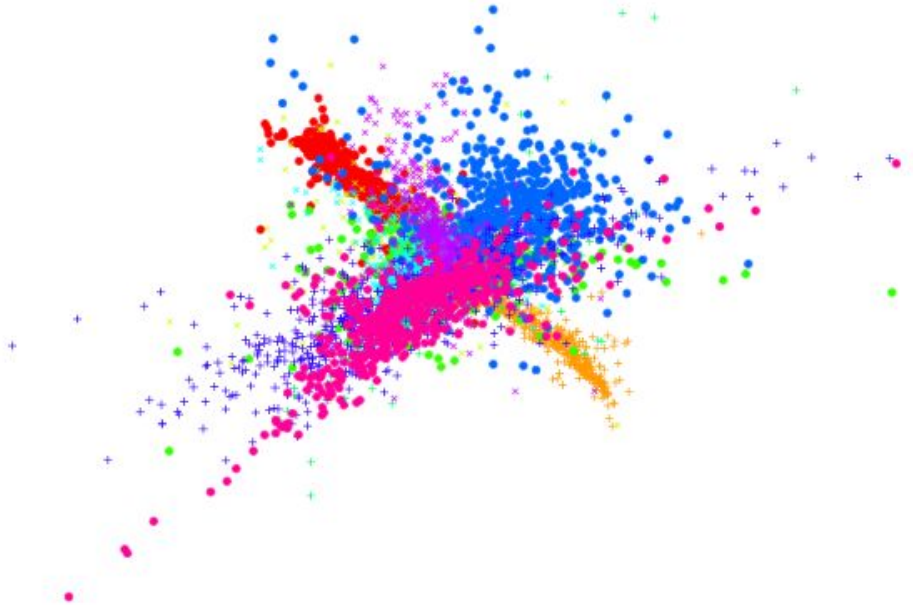
(Something like a Student's-t distribution)

- The "variance" term defines relevant length-scale for neighbors. An inherent hyperparameter of the algorithm (called perplexity).
- The Student's-t distribution has heavy tails, does a better job of handling the problem that distance and volume are hugely different in the two different dimensionalities

MNIST: 784-D to 2-D



(a) Visualization by t-SNE.



(b) Visualization by LLE.