

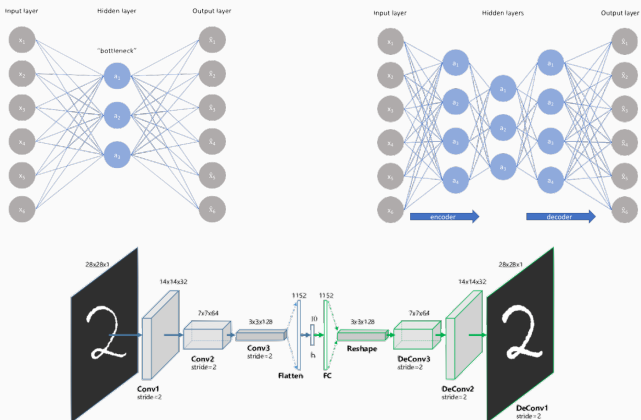
CS-GY 6923: Lecture 13

Semantic Embeddings, Image Generation

NYU Tandon School of Engineering, Prof. Christopher Musco

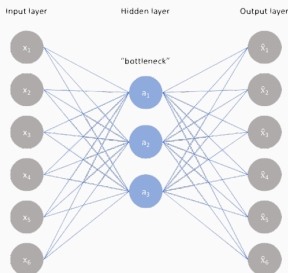
AUTOENCODER

Recap: Goal of autoencoder models is to map input data to a close approximation of the original that takes less space to represent.



PRINCIPAL COMPONENT ANALYSIS

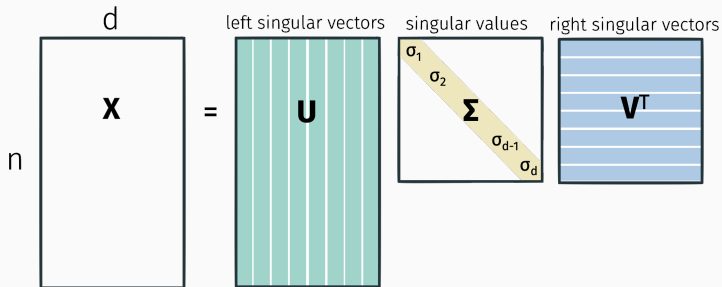
PCA is the “linear regression” of autoencoders:



- Simplest possible model. One layer, no non-linearities.
 - $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{W}_1\mathbf{W}_2$ where $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{W}_1 \in \mathbb{R}^{d \times k}$, $\mathbf{W}_2 \in \mathbb{R}^{k \times d}$.
 - Want to minimize $\min_{\mathbf{W}_1, \mathbf{W}_2} \|\mathbf{X} - \mathbf{X}\mathbf{W}_1\mathbf{W}_2\|_F^2$.
- Equivalent to low-rank approximation. Can be efficiently and provably optimized using the SVD.

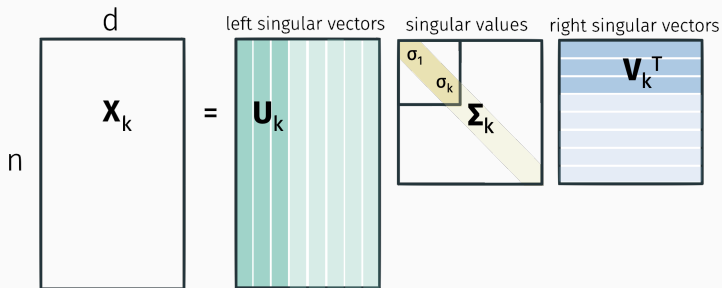
SINGULAR VALUE DECOMPOSITION

Any matrix X can be written:



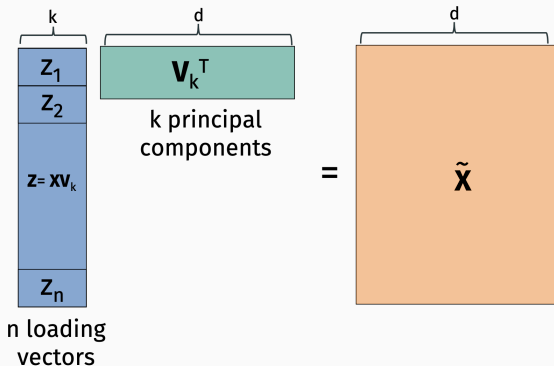
Where $U^T U = I$, $V^T V = I$, and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_d \geq 0$. I.e. U and V are orthogonal matrices. Can be computed in $O(nd^2)$ time (faster with approximation algos).

PARTIAL SINGULAR VALUE DECOMPOSITION



Can be computed in roughly $O(ndk)$ time.

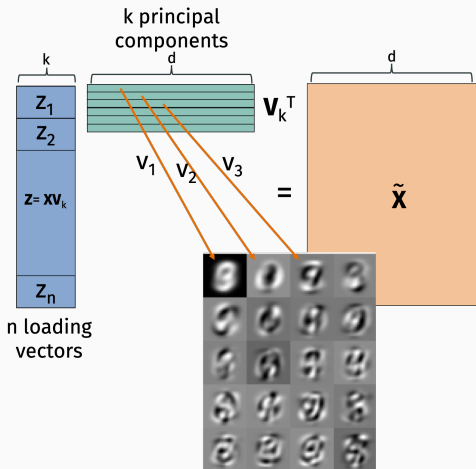
PRINCIPAL COMPONENT ANALYSIS



Eckart-Young-Mirsky Theorem: $\tilde{\mathbf{X}} = \mathbf{XV}_k\mathbf{V}_k^T$ is the optimal low-rank approximation to \mathbf{X} . So $\mathbf{W}_1 = \mathbf{V}_k$ and $\mathbf{W}_2 = \mathbf{V}_k^T$ are optimal autoencoder parameters.

PRINCIPAL COMPONENTS

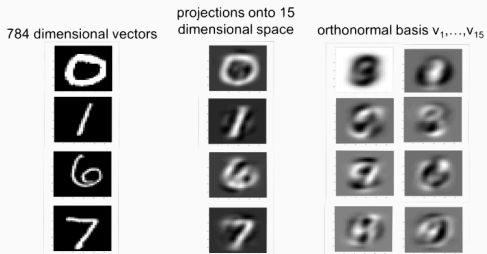
MNIST principal components:



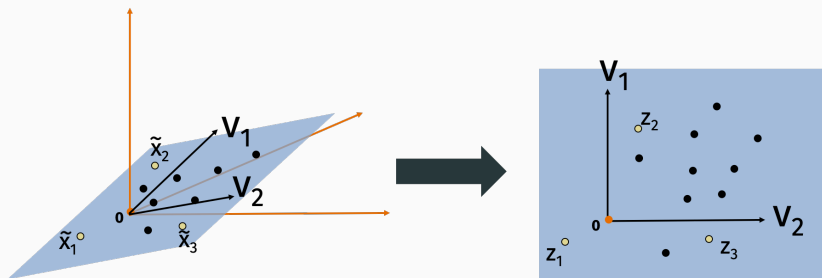
Principal components are a small set of vectors that can be recombined to approximate rows in \tilde{X} .

PRINCIPAL COMPONENTS

MNIST principal components:



PCA PRESERVES GEOMETRY OF INPUT DATA



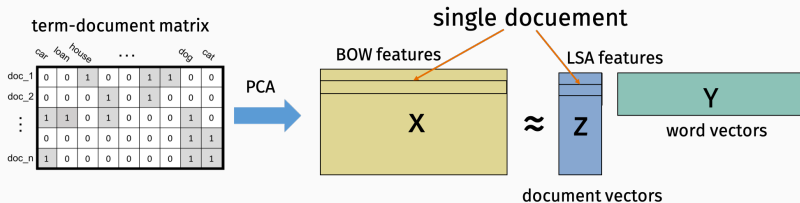
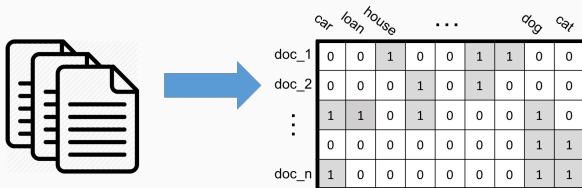
$$\|\mathbf{x}_i\|_2^2 \approx \|\mathbf{z}_i\|_2^2$$

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx \langle \mathbf{z}_i, \mathbf{z}_j \rangle$$

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \approx \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$$

LATENT SEMANTIC ANALYSIS

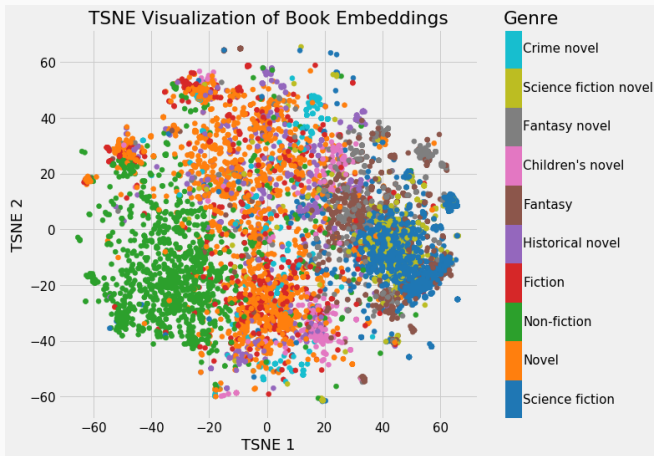
Word-document matrix:



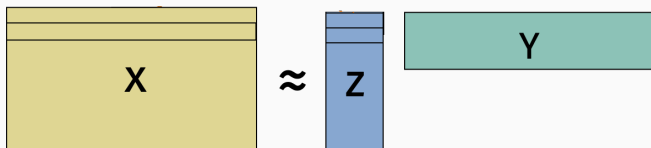
For documents with a lot of shared words, $\langle x_i, x_j \rangle$ is a large positive number.

DOCUMENT EMBEDDINGS

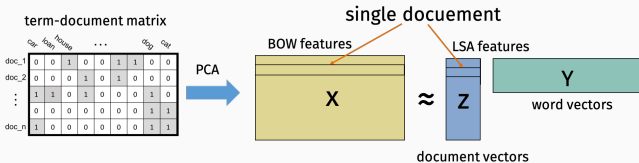
For similar documents, $\langle \mathbf{z}_i, \mathbf{z}_j \rangle$ should be large. I.e. \mathbf{z}_i and \mathbf{z}_j point in the same direction.



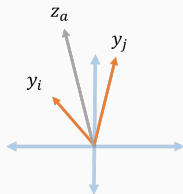
Simple but useful observation: The i, j entry of \tilde{X} equals $\langle z_i, y_j \rangle$.



WORD EMBEDDINGS



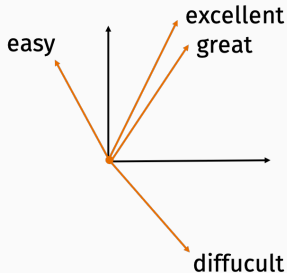
- $\langle \mathbf{y}_i, \mathbf{z}_a \rangle \approx 1$ when doc_a contains $word_i$.
- If $word_i$ and $word_j$ both appear in doc_a , then $\langle \mathbf{y}_i, \mathbf{z}_a \rangle \approx \langle \mathbf{y}_j, \mathbf{z}_a \rangle \approx 1$, so we expect $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ to be large.



If two words appear in the same document their word vectors tend to point more in the same direction.

WORD EMBEDDINGS

Result: Map words to numerical vectors in a semantically meaningful way. Similar words map to similar vectors. Dissimilar words to dissimilar vectors.

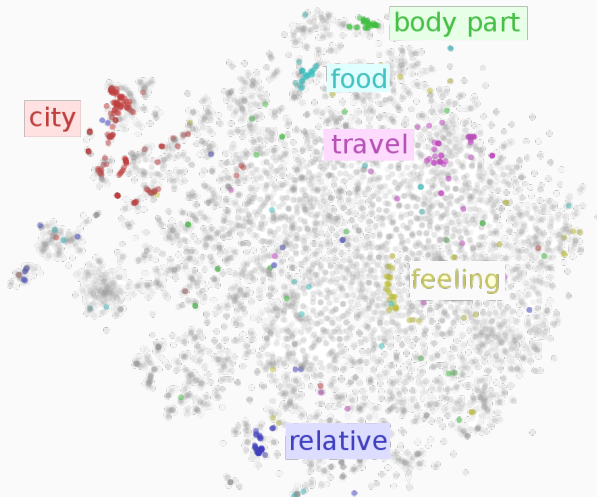


Extremely useful “side-effect” of LSA.

Capture e.g. the fact that “great” and “excellent” are near synonyms. Or that “difficult” and “easy” are antonyms.

WORD EMBEDDINGS

For similar words, $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ should be large. I.e. \mathbf{y}_i and \mathbf{y}_j point in the same direction.



Review 1: *Very small and handy for traveling or camping. Excellent quality, operation, and appearance.*

Review 2: *So far this thing is great. Well designed, compact, and easy to use. I'll never use another can opener.*

Review 3: *Not entirely sure this was worth \$20. Mom couldn't figure out how to use it and it's fairly difficult to turn for someone with arthritis.*

Goal is to classify reviews as “positive” or “negative”.

BAG-OF-WORDS FEATURES

Vocabulary: Small, handy, excellent, great, quality, compact, easy, difficult.

Review 1: *Very small and handy for traveling or camping. Excellent quality, operation, and appearance.*

[, , , , , , ,]

Review 2: *So far this thing is great. Well designed, compact, and easy to use. I'll never use another can opener.*

[, , , , , , ,]

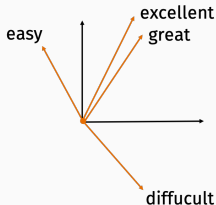
Review 3: *Not entirely sure this was worth \$20. Mom couldn't figure out how to use it and it's fairly difficult to turn for someone with arthritis.*

[, , , , , , ,]

SEMANTIC EMBEDDINGS

Bag-of-words approach typically only works for large data sets.

The features do not capture the fact that “great” and “excellent” are near synonyms. Or that “difficult” and “easy” are antonyms.






This can be addressed by first mapping words to semantically meaningful vectors. That mapping can be trained using a much large corpus of text than the data set you are working with (e.g. Wikipedia, Twitter, news data sets).

USING WORD EMBEDDINGS

How to go from word embeddings to features for a whole sentence or chunk of text?

Very small and handy for traveling or camping. **remove "stop words"** → [small, handy, traveling, camping]

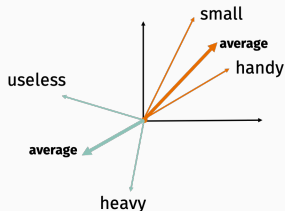
[small, handy, traveling, camping] **word embedding** → 
 $y_1 y_2 \dots y_q$


 $y_1 y_2 \dots y_q$ **???** → 
feature vector

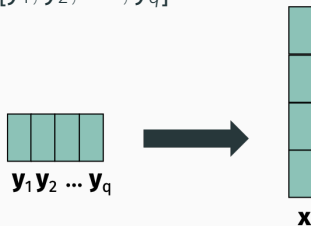
USING WORD EMBEDDINGS

A few simple options:

Feature vector $\mathbf{x} = \frac{1}{q} \sum_{i=1}^q \mathbf{y}_i$.

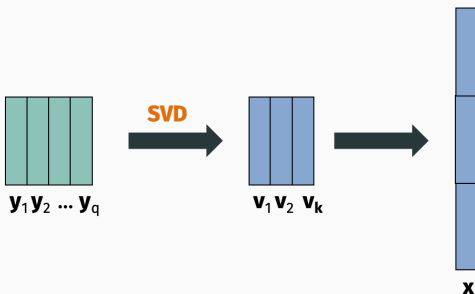


Feature vector $\mathbf{x} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_q]$.



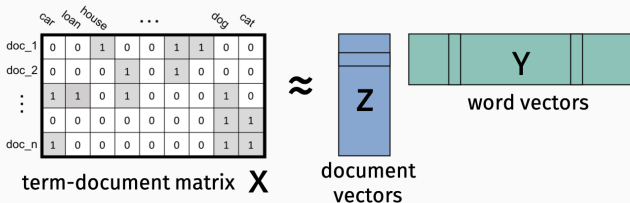
USING WORD EMBEDDINGS

To avoid issues with inconsistent sentence length, word ordering, etc., can concatenate a fixed number of top principal components of the matrix of word vectors:



There are much more complicated approaches that account for word position in a sentence. Lots of pretrained libraries available (e.g. Facebook's **InferSent**).

Another view on word embeddings from LSA:

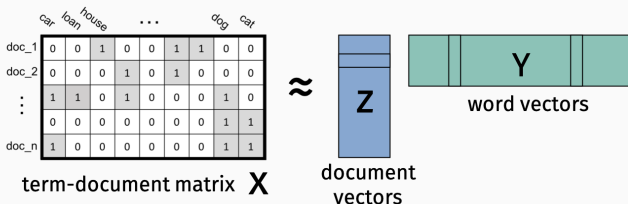


We chose Z to equal $XV_k = U_k \Sigma_k$ and $Y = V_k^T$.

Could have just as easily set $Z = U_k$ and $Y = \Sigma_k V_k^T$, so Z has orthonormal columns.

WORD EMBEDDINGS

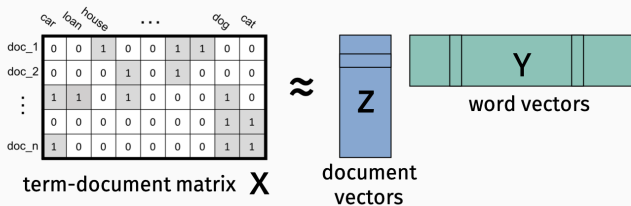
Another view on word embeddings from LSA:



- $X \approx ZY$
- $X^T X \approx Y^T Z^T Z Y = Y^T Y$
- So for $word_i$ and $word_j$, $\langle y_i, y_j \rangle \approx [X^T X]_{i,j}$.

What does the i, j entry of $X^T X$ represent?

WORD EMBEDDINGS



What does the i, j entry of $X^T X$ represent?

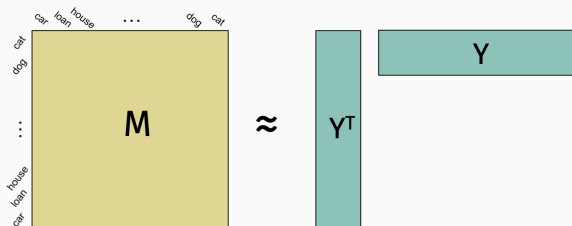
$\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ is larger if $word_i$ and $word_j$ appear in more documents together (high value in **word-word co-occurrence matrix**, $\mathbf{X}^T\mathbf{X}$).
Similarity of word embeddings mirrors similarity of word context.

General word embedding recipe:

1. Choose similarity metric $k(word_i, word_j)$ which can be computed for any pair of words.
2. Construct similarity matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ with $\mathbf{M}_{i,j} = k(word_i, word_j)$.
3. Find low rank approximation $\mathbf{M} \approx \mathbf{Y}^T\mathbf{Y}$ where $\mathbf{Y} \in \mathbb{R}^{k \times n}$.
4. Columns of \mathbf{Y} are word embedding vectors.

We expect that $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ will be larger for more similar words.

WORD EMBEDDINGS



How do current state-of-the-art methods differ from LSA?

- Similarity based on co-occurrence in smaller chunks of words. E.g. in sentences or in any consecutive sequences of 3, 4, or 10 words.
- Usually transformed in non-linear way. E.g.
 $k(\text{word}_i, \text{word}_j) = \frac{p(i,j)}{p(i)p(j)}$ where $p(i,j)$ is the frequency both i, j appeared together, and $p(i), p(j)$ is the frequency either one appeared.

MODERN WORD EMBEDDINGS

Computing word similarities for “window size” 4:

The girl walks to her **dog to the park.**
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog **park is always crowded** on Saturdays.

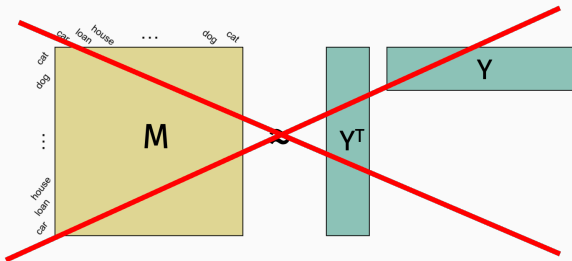
The girl walks to **her dog to the park.**
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

	dog	park	crowded	the
dog	0	2	0	3
park	2	0	1	2
crowded	0	1	0	0
the	3	2	0	0

Current state of the art models: GloVe, word2vec.

- **word2vec** was originally presented as a shallow neural network model, but it is equivalent to matrix factorization method (Levy, Goldberg 2014).
- For **word2vec**, similarity metric is the “point-wise mutual information”: $\log \frac{p(i,j)}{p(i)p(j)}$.

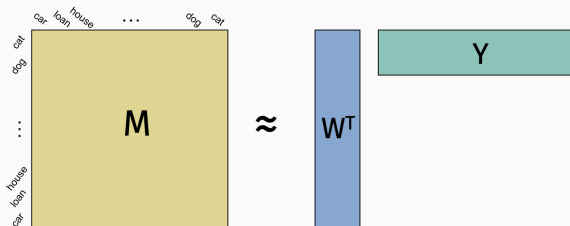
CAVEAT ABOUT FACTORIZATION



SVD will not return a symmetric factorization in general. In fact, if M is not positive semidefinite¹ then the optimal low-rank approximation does not have this form.

¹i.e., $k(\text{word}_i, \text{word}_j)$ is not a positive semidefinite kernel.

CAVEAT ABOUT FACTORIZATION



- For each word i we get a left and right embedding vector \mathbf{w}_i and \mathbf{y}_i . It's reasonable to just use one or the other.
- If $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ is large and positive, we expect that \mathbf{y}_i and \mathbf{y}_j have similar similarity scores with other words, so they typically are still related words.
- Another option is to use as your features for a word the concatenation $[\mathbf{w}_i, \mathbf{y}_i]$

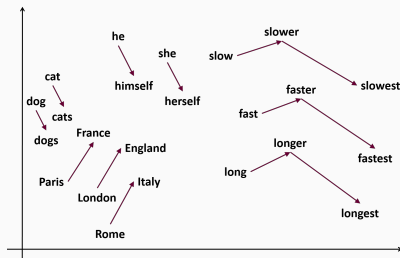
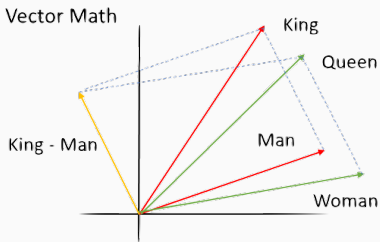
Lots of pre-trained word vectors are available online:

- Original gloVe website:
`https://nlp.stanford.edu/projects/glove/`
- Compilation of many sources:
`https://github.com/3Top/word2vec-api`

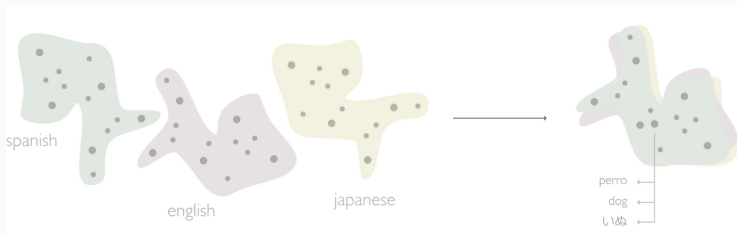
WORD EMBEDDINGS MATH

Lots of cool demos for what can be done with these embeddings. E.g. “vector math” to solve analogies.

Vector Math



FORWARD LOOKING APPLICATION: UNSUPERVISED TRANSLATION



- Train word-embeddings for languages separately. Obtain lowish dimensional point clouds of words.
- Perform rotation/alignment to match up these point clouds.
- Equivalent words should land on top of each other.

No needs for labeled training data like translated pairs of sentences!

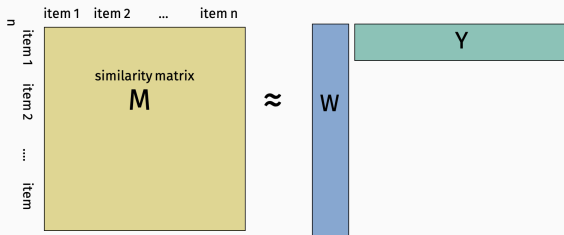
Why not monkey or whale language?



Earth Species Project (www.earthspecies.org), CETI Project
(www.projectceti.org)

SEMANTIC EMBEDDINGS

The same approach used for word embeddings can be used to obtain meaningful numerical features for any other data where there is a natural notion of similarity.

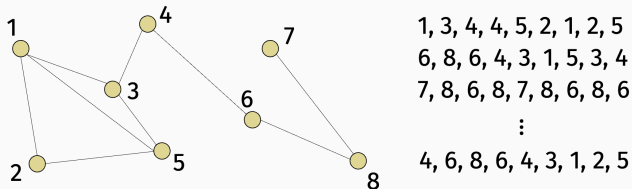


For example, the items could be nodes in a social network graph. Maybe we want to predict an individual's age, level of interest in a particular topic, political leaning, etc.

NODE EMBEDDINGS



Generate random walks (e.g. “sentences” of nodes) and measure similarity by node co-occurrence frequency.



NODE EMBEDDINGS

Again typically normalized and apply a non-linearity (e.g. log) as in word embeddings.

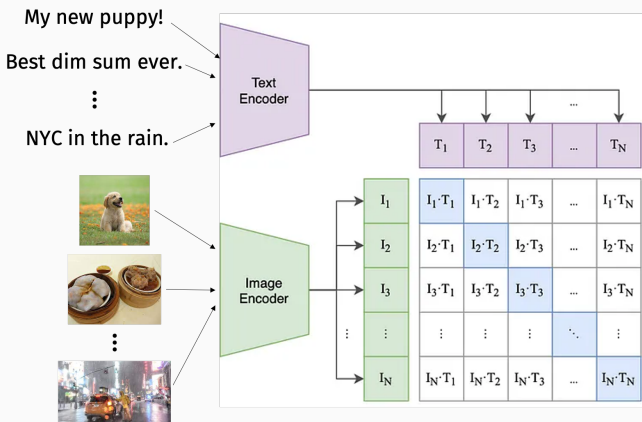
1, 3, 4, 4, 5, 2, 1, 2, 5
6, 8, 6, 4, 3, 1, 5, 3, 4
7, 8, 6, 8, 7, 8, 6, 8, 6
⋮
4, 6, 8, 6, 4, 3, 1, 2, 5

	node 1	node 2	...	node 8
node 1	0	2		1
node 2	2	0		0
⋮				
node 8	1	0		0

Popular implementations: **DeepWalk**, **Node2Vec**. Again initially derived as simple neural network models, but are equivalent to matrix-factorization (Qiu et al. 2018).

BIMODAL EMBEDDINGS

We can also create embeddings that represent different types of data. OpenAI's clip architecture:



Goal: Train embedding architectures so that $\langle T_i, I_j \rangle$ are similar if image and sentence are similar.

CLIP TRAINING

What do we use as ground truth similarities during training?
Sample a batch of sentence/image pairs and just use identity matrix.



My new puppy!	1	0	0
Best dim sum ever.	0	1	0
NYC in the rain.	0	0	1

This is called contrastive learning. Train unmatched text/image pairs to have nearly orthogonal embedding vectors.

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford^{*1} Jong Wook Kim^{*1} Chris Hallacy¹ Aditya Ramesh¹ Gabriel Goh¹ Sandhini Agarwal¹
Girish Sastry¹ Amanda Askell¹ Pamela Mishkin¹ Jack Clark¹ Gretchen Krueger¹ Ilya Sutskever¹

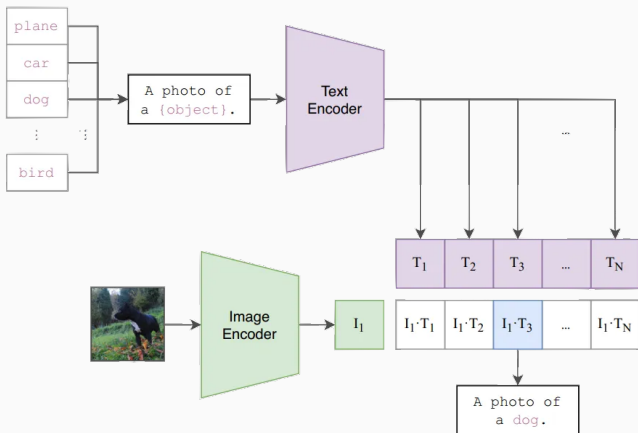
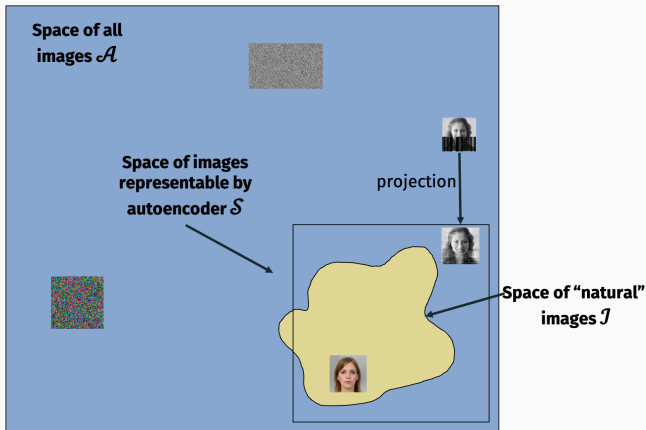


IMAGE SYNTHESIS

AUTOENCODERS LEARN COMPRESSED REPRESENTATIONS



$f(\mathbf{x}) = d(e(\mathbf{x}))$ projects an image \mathbf{x} closer to the space of natural images.

Suppose we want to generate a random natural image. How might we do that?

- **Option 1:** Draw each pixel value in x uniformly at random. Draws a random image from \mathcal{A} .



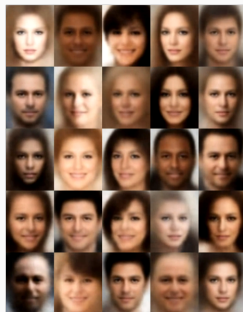
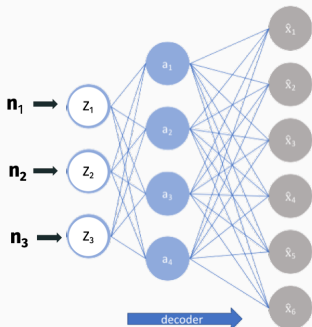
- **Option 2:** Draw x randomly from \mathcal{S} , the space of images representable by the autoencoder.



How do we randomly select an image from \mathcal{S} ?

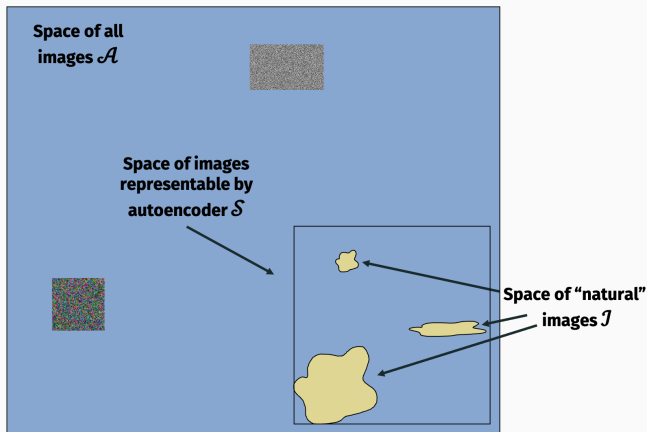
AUTOENCODERS FOR DATA GENERATION

Autoencoder approach to generative ML: Feed random inputs into decode to produce random realistic outputs.



Main issue: most random inputs words will “miss” and produce garbage results.

AUTOENCODERS FOR DATA GENERATION

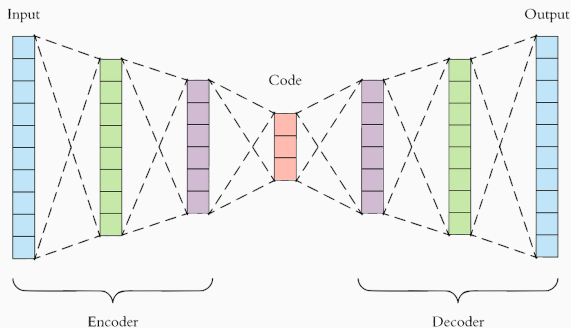


Variational auto-encoders attempt to resolve this issue.

VARIATIONAL AUTOENCODERS

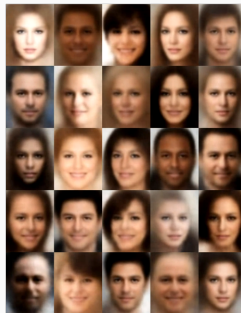
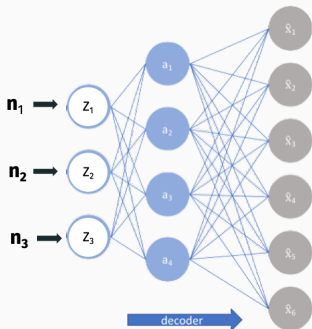
Variational auto-encoders attempt to resolve this issue. Basic ideas:

- Add noise during training.
- Add penalty term so that distribution of code vectors generated looks like mean 0, variance 1 Gaussian.



GENERATIVE ADVERSARIAL NETWORKS

Variation AE's give very good results, but tends to produce images with immediately recognizable flaws (e.g. soft edges, high-frequency artifacts).



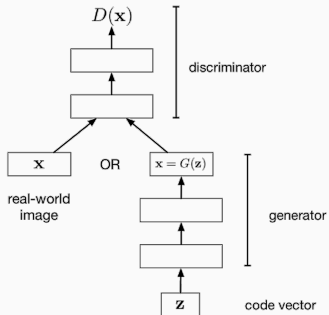
GENERATIVE ADVERSARIAL NETWORKS (GANs)

Lots of efforts to hand-design regularizers that penalize images that don't look realistic to the human eye.

Main idea behind GANs: Use machine learning to automatically encourage realistic looking images.

$$\min_{\theta} L(\theta) + P(\theta)$$

GENERATIVE ADVERSARIAL NETWORKS (GANS)

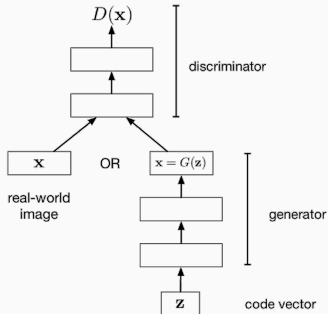


Let x_1, \dots, x_n be real images and let z_1, \dots, z_m be random code vectors. The goal of the discriminator is to output a number between $[0, 1]$ which is close to 0 if the image is fake, close to 1 if it's real.

Train weights of discriminator D_θ to minimize:

$$\min_{\theta} \sum_{i=1}^n -\log(D_\theta(x_i)) + \sum_{i=1}^m -\log(1 - D_\theta(G_{\theta'}(z_i)))$$

GENERATIVE ADVERSARIAL NETWORKS (GANS)



Goal of the generator $G_{\theta'}$ is the opposite. We want to maximize:

$$\max_{\theta'} \sum_{i=1} -\log(1 - D_{\theta}(G_{\theta'}(z_i)))$$

This is called an “adversarial loss function”. D is playing the role of the adversary.

GENERATIVE ADVERSARIAL NETWORKS (GANS)

$$\theta^*, \theta'^* \text{ solve } \min_{\theta} \max_{\theta'} \sum_{i=1}^n -\log(D_{\theta}(x_i)) + \sum_{i=1}^m -\log(1 - D_{\theta}(G_{\theta'}(z_i)))$$

This is called a minimax optimization problem. Really tricky to solve in practice.

- **Repeatedly play:** Fix one of θ^* or θ'^* , train the other to convergence, repeat.
- **Simultaneous gradient descent:** Run a single gradient descent step for each of θ^* , θ'^* and update D and G accordingly. Difficult to balance learning rates.
- Lots of tricks (e.g. slight different loss functions) can help.

GENERATIVE ADVERSARIAL NETWORKS (GANS)

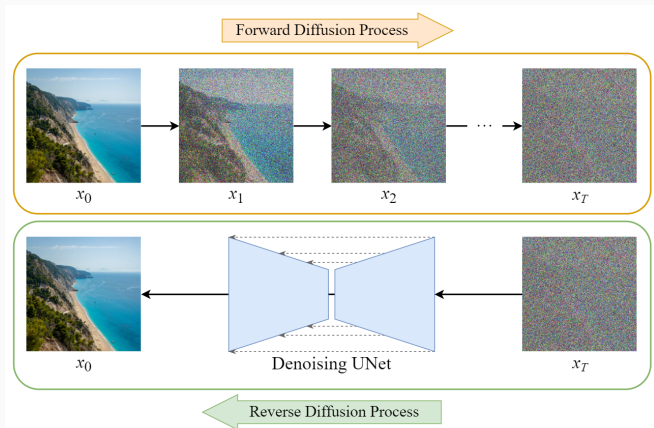
State of the art until a few years ago.



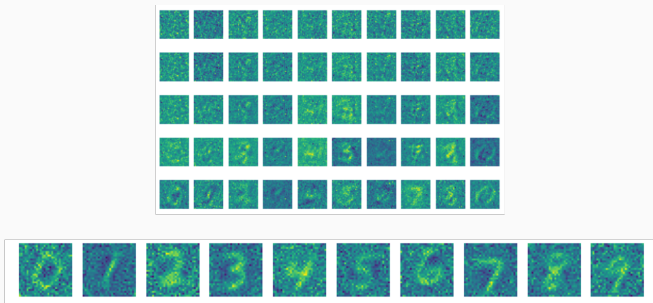
DIFFUSION

Auto-encoder/GAN approach: Input noise, map directly to image.

Diffusion: Slowly move from noise to image.



Teal created a demo for generating digits by training on MNIST.



SEMANTIC EMBEDDINGS + DIFFUSION

Text to image synthesis: Dall-E, Imagen, Stable Diffusion



"A chair that looks like an avocado"