Machine Learning Course

Lecture 13: Word Embeddings and Autoencoders

Harbour.Space University
March 2020

Radoslav Neychev

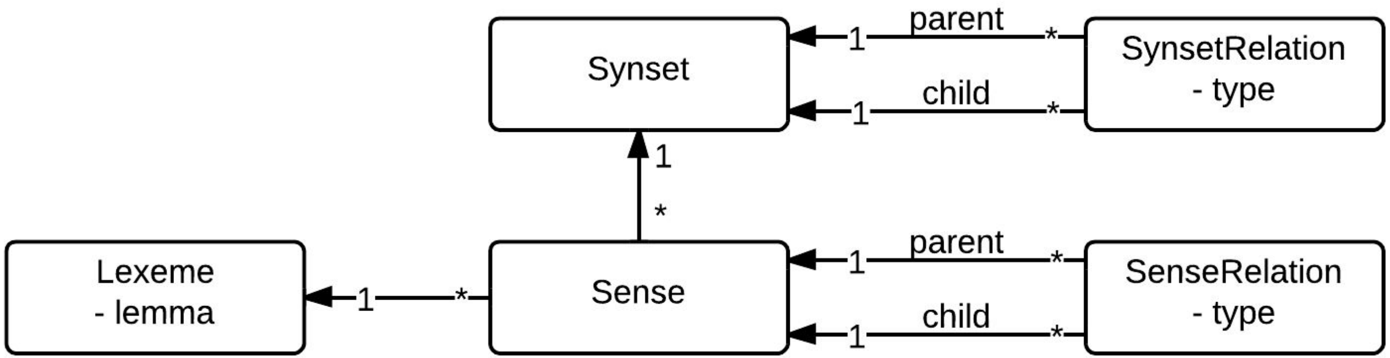
Outline

1. Discrete representations.
2. Matrix of co-occurrence.
3. Embeddings (GloVe, word2vec).
4. Examples.
5. Autoencoders
6. Denoising Autoencoders
7. Practice Session

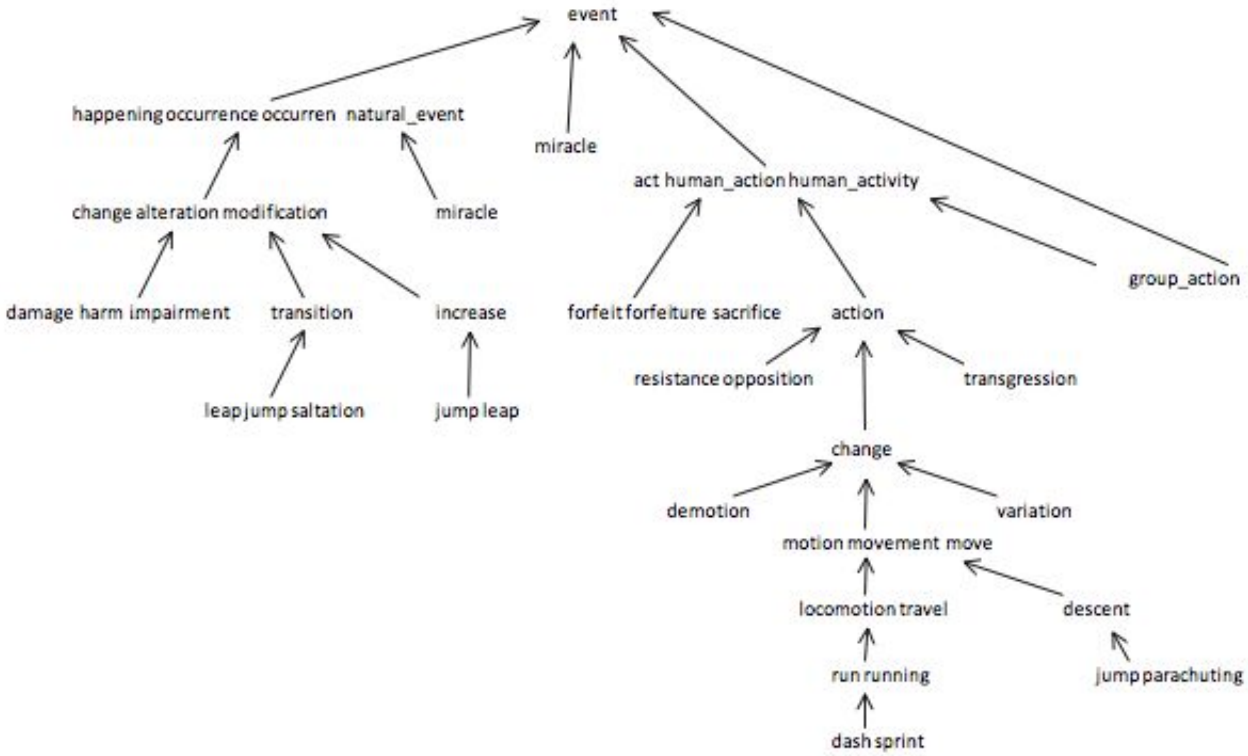
Source: [Stanford CS224d course materials]

How to represent text in a computer?

Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets



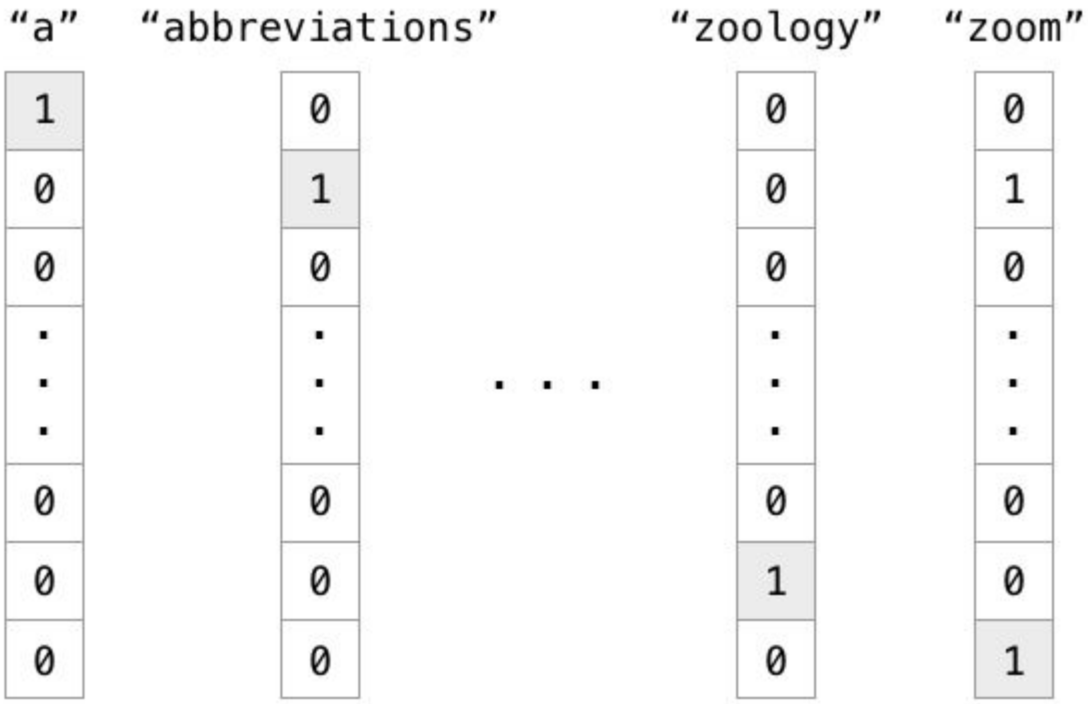
How to represent text in a computer: WordNet



Discrete representations: problems

- Missing new words
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity

Discrete representations: one-hot encoding



TF-IDF

If word is repeated in the query, it's probably important

Repetitions of query words in the document → good

Rare words more important

$$s(Q,D) = \sum_{w} tf_{w,Q} \cdot \underbrace{\frac{tf_{w,D}}{tf_{w,D} + \frac{k|D|}{avg|D|}}}_{} \cdot \log \frac{|C|}{df_w}$$

The more query words we match, the better. Σ over the vocabulary

Repetitions of same word less important than different words. Except in very long documents

TF - term frequency

IDF - Inverse Document Frequency

TF-IDF: make it simple

$$\mathrm{tf}(''\mathsf{this}'', d_1) = \frac{1}{5} = 0.2$$

$$\mathrm{tf}(''\mathsf{this}'', d_2) = \frac{1}{7} \approx 0.14$$

$$\mathrm{idf}(''\mathsf{this}'', D) = \log\left(\frac{2}{2}\right) = 0$$

$$\mathrm{tfidf}(''\mathsf{this}'', d_1, D) = 0.2 \times 0 = 0$$

$$\mathrm{tfidf}(''\mathsf{this}'', d_2, D) = 0.14 \times 0 = 0$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

Word ‘this’ is not very informative

Words cooccurrences

One of the most successful ideas of statistical NLP:

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

Words cooccurrences

Finding N-grams in a text

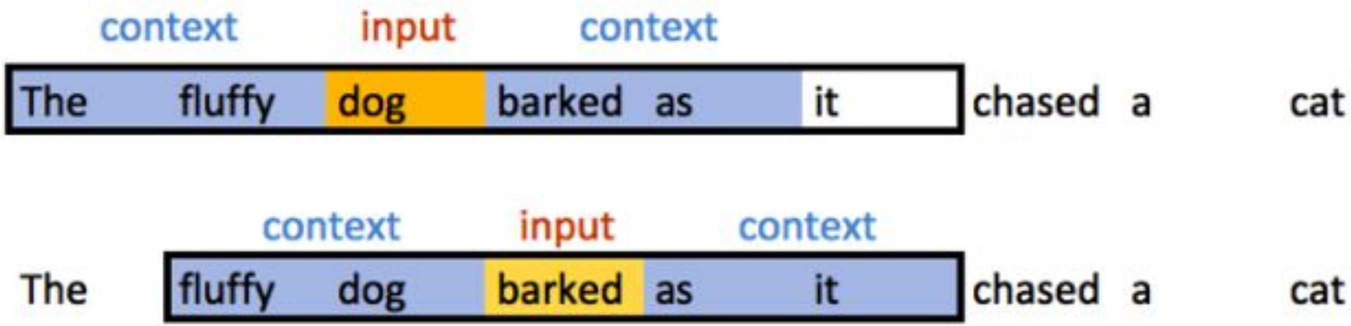
Word-document cooccurrence matrix

Window around each word

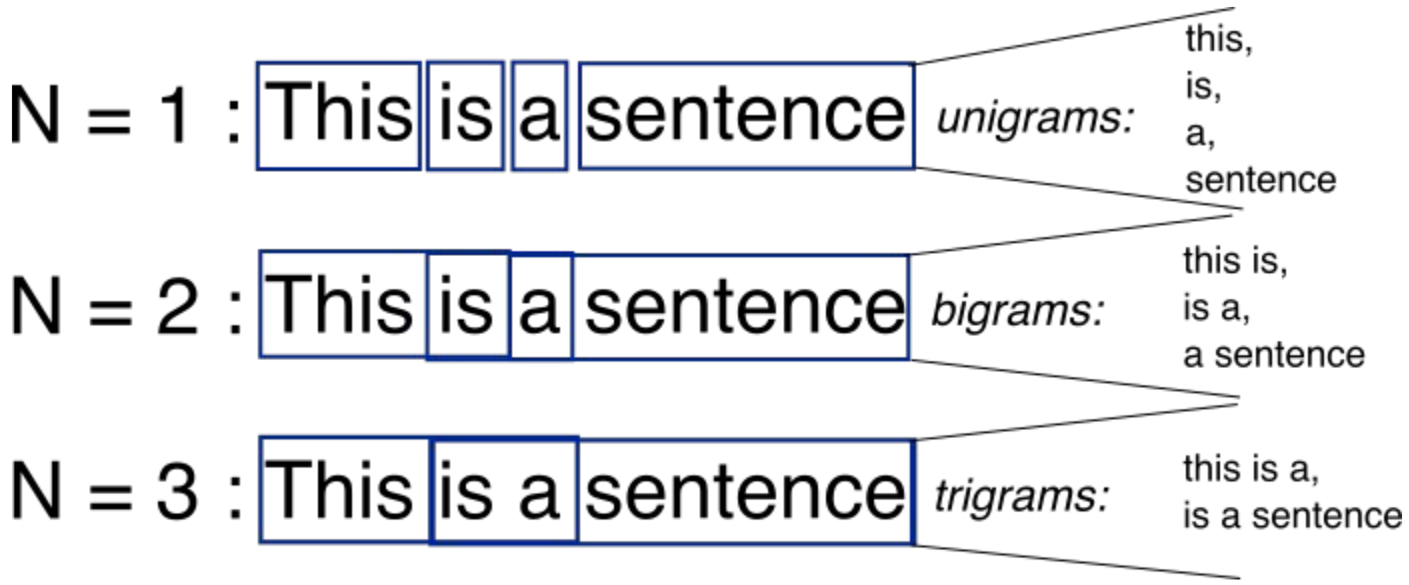
Word-document cooccurrence matrix

$$X = \begin{array}{c c} & \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{array}{c} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} & \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Words cooccurrences: sliding window



Words cooccurrences: n-grams

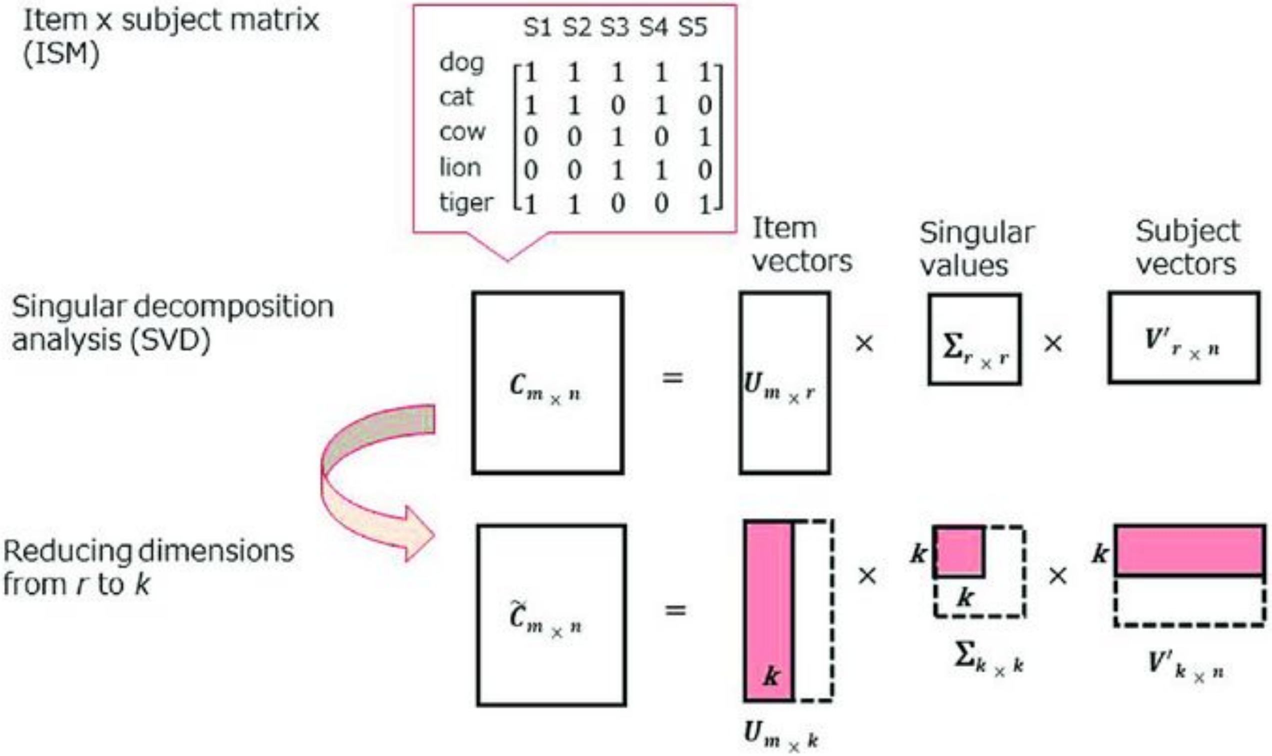


Cooccurrence vectors: problems

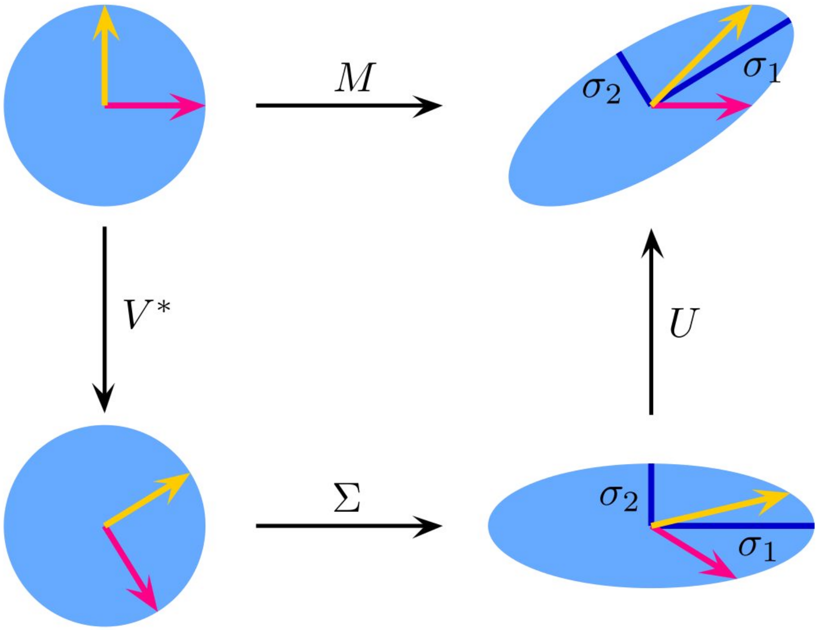
- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues

Models are less robust

Reducing dimensionality: SVD of cooccurrence matrix



SVD: intuition

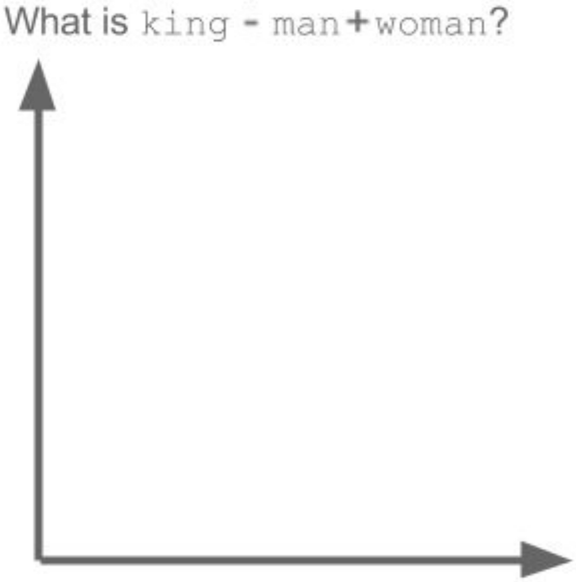


$$M = U \cdot \Sigma \cdot V^*$$

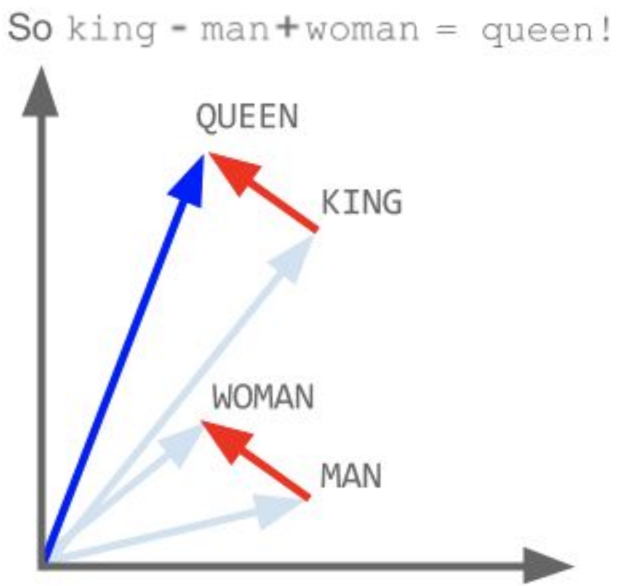
SVD: problems

- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when n<m)
- Hard to incorporate new words or documents
- Different learning regime than other DL models

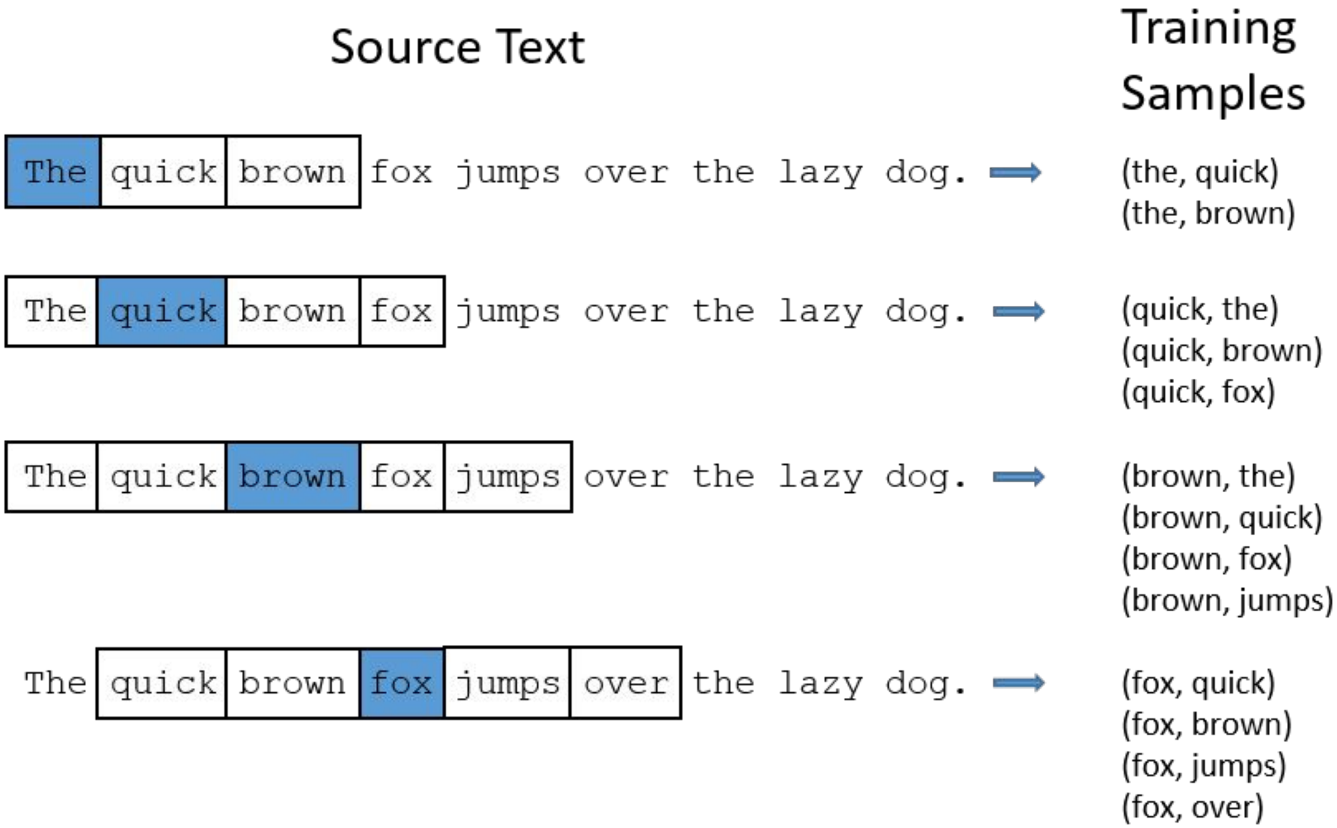
Embeddings: intuition



Embeddings: intuition

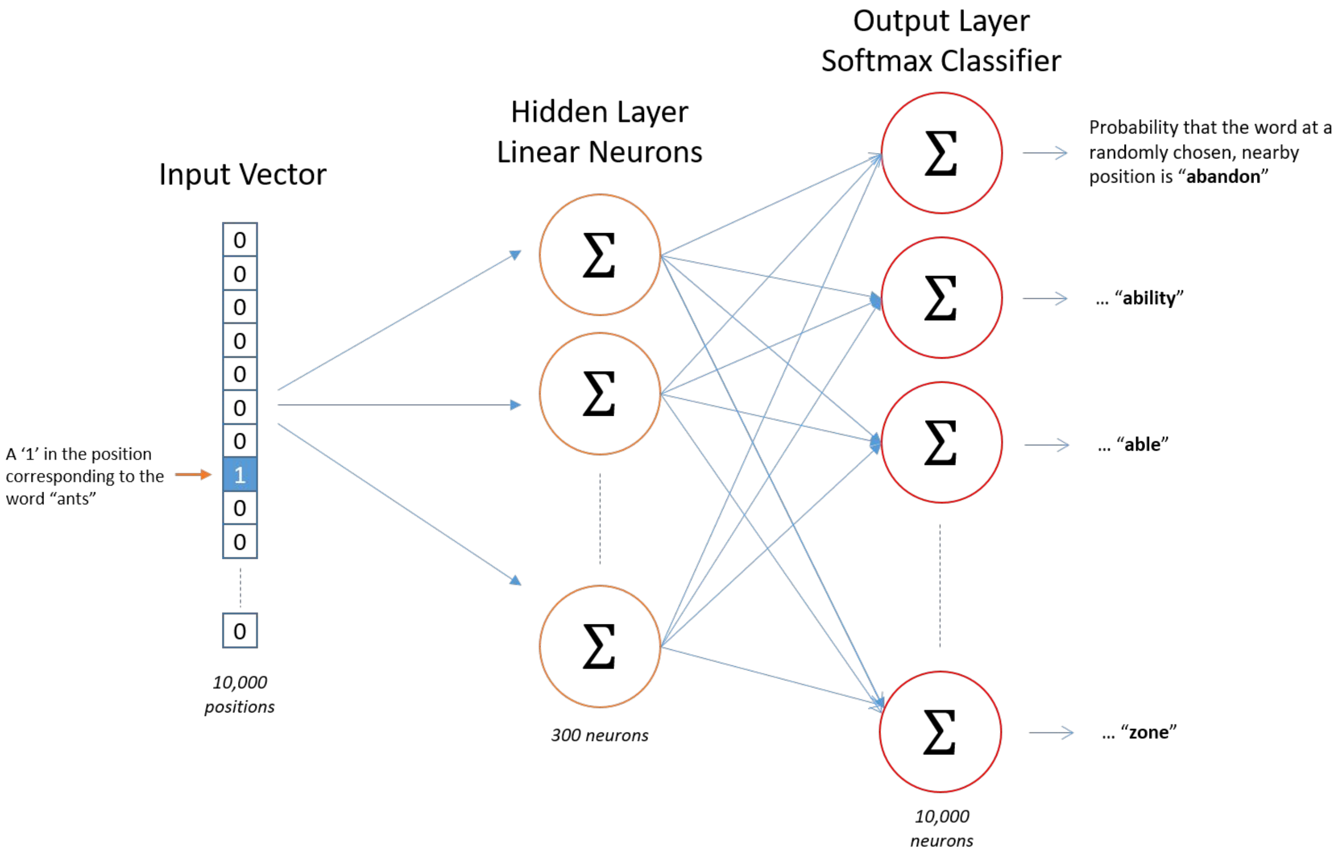


Embeddings: word2vec



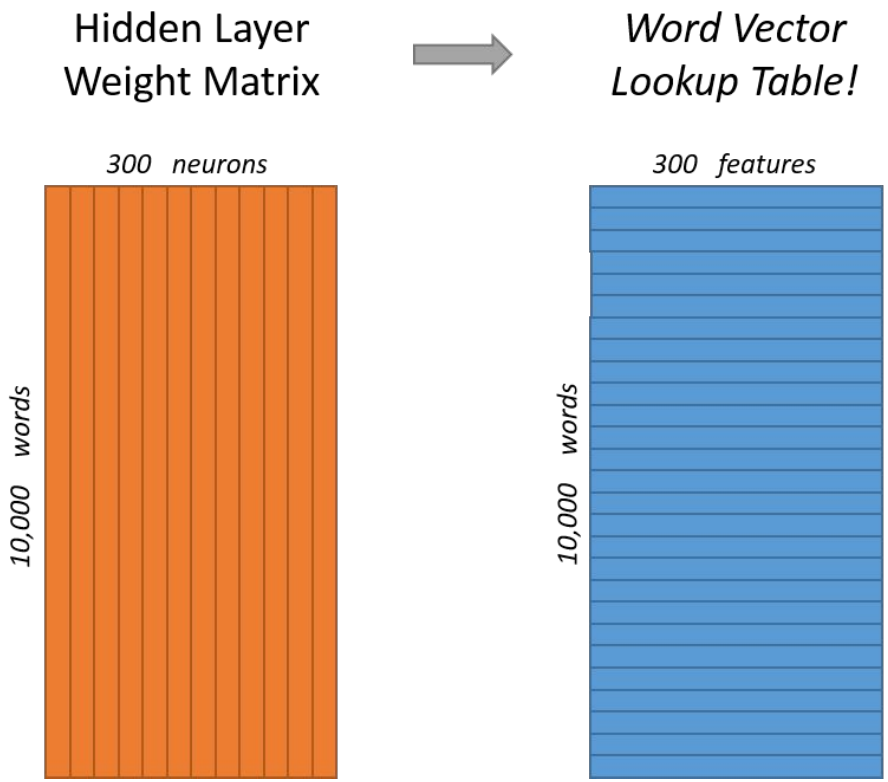
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec

- Word vectors with 300 components
- Vocabulary of 10,000 words.
- Weight matrix with 300 x 10,000 = 3 million weights each!

Training is too long and computationally expensive

How to fix this?

Embeddings: word2vec

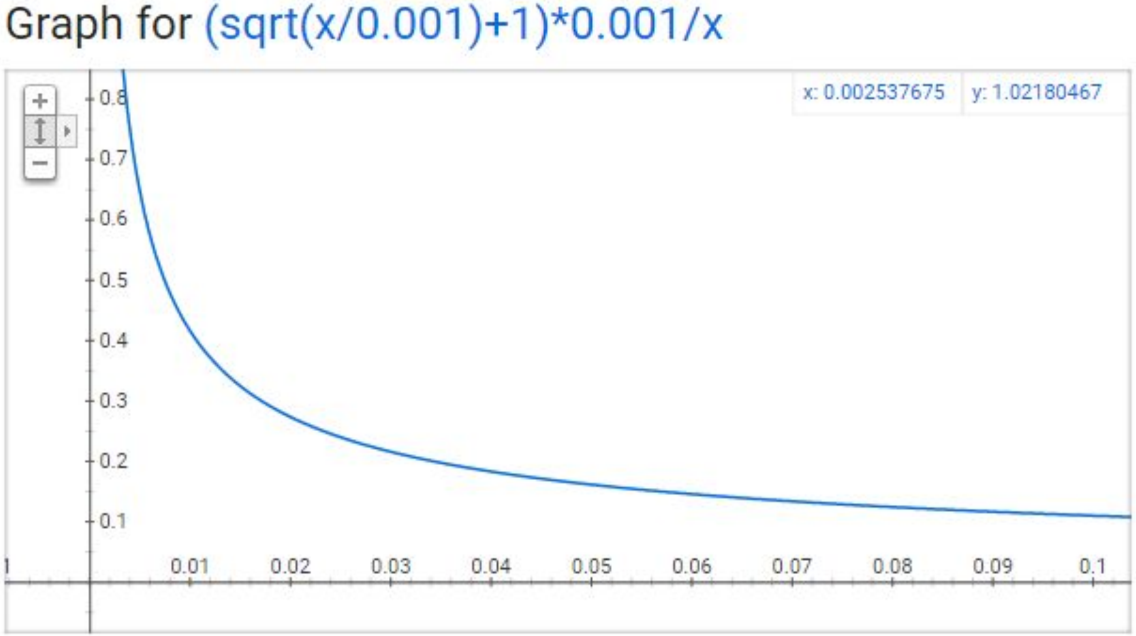
Basic approaches:

1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Embeddings: word2vec

Subsampling frequent words.

w_i is the word, $z(w_i)$ is the fraction of this word in the whole



$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

Source: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

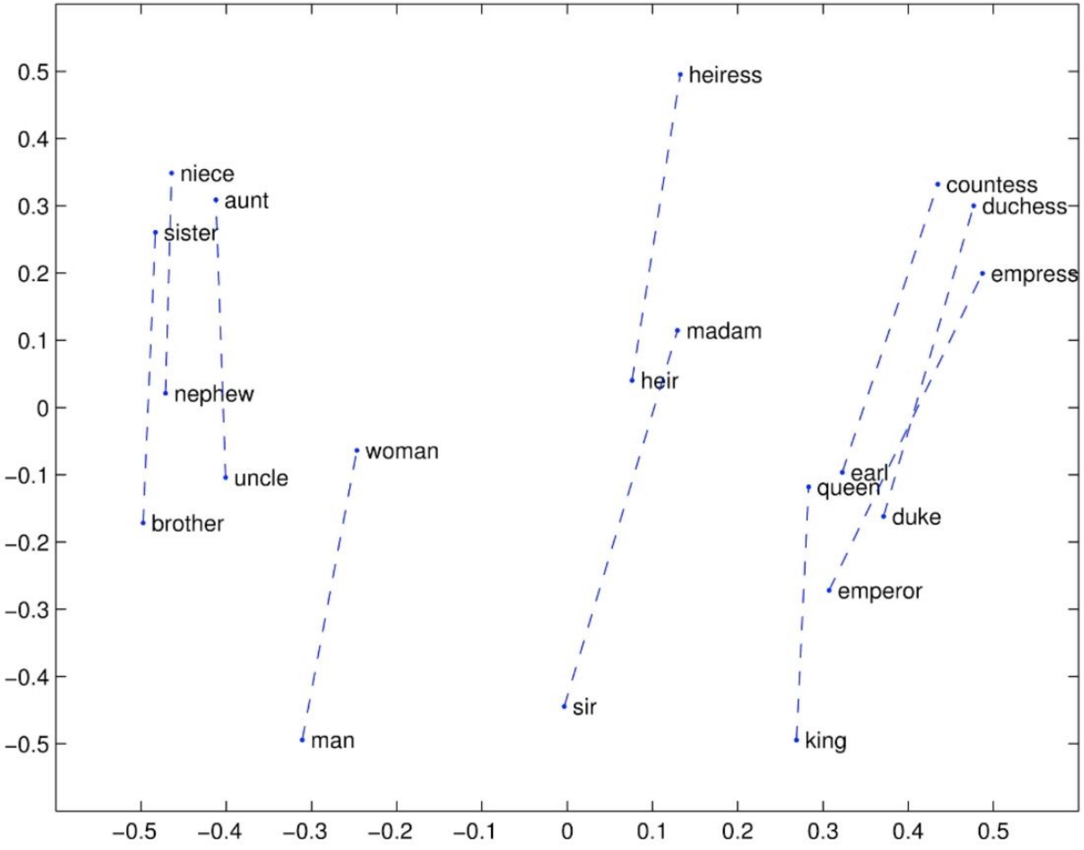
Embeddings: negative sampling

Negative Sampling idea: only few words error is computed. All other words has zero error, so no updates by the backprop mechanism.

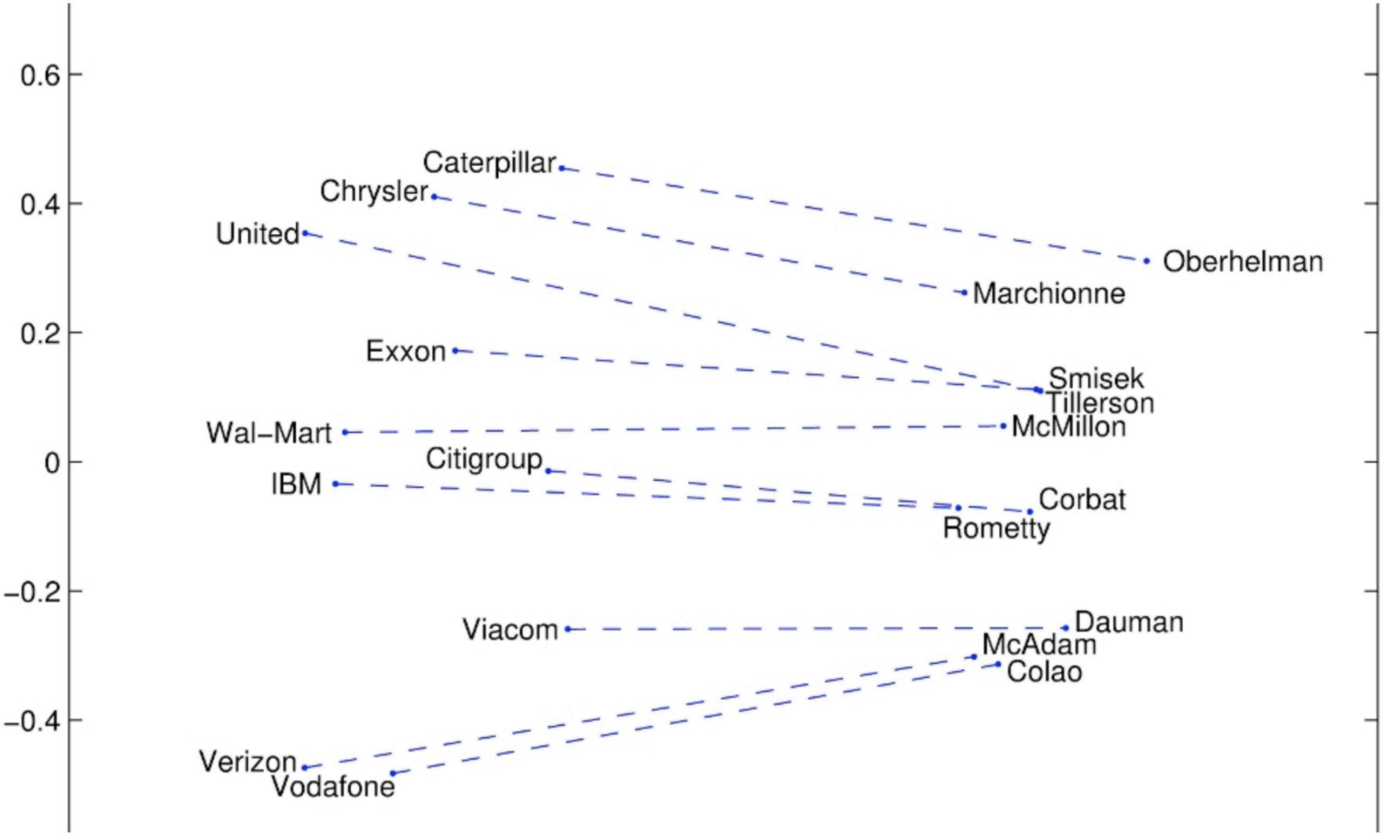
More frequent words are selected to be negative samples more often.The probability for a selecting a word is just it’s weight divided by the sum of weights for all words.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4}\right)}$$

GloVe Visualizations



GloVe Visualizations: Company - CEO



Conclusion

Word vectors are simply vectors of numbers that represent the meaning of a word

Approaches:

- One-hot encoding
- Bag-of-words models
- Counts of word / context co-occurrences
- TF-IDF
- Predictions of context given word (skip-gram neural network models, e.g. word2vec)

Autoencoders

Denote **z** as encoded with encoder E input **x**

$$\mathbf{z} = E(\mathbf{x}, \boldsymbol{\theta}_E)$$

Decoder D recovers **x** from latent representation

$$\hat{\mathbf{x}} = D(\mathbf{z}, \boldsymbol{\theta}_D)$$

Optimal parameters learned w.r.t. loss function L

$$[\boldsymbol{\theta}_E, \boldsymbol{\theta}_D] = \arg\min L(\hat{\mathbf{x}}, \mathbf{x})$$

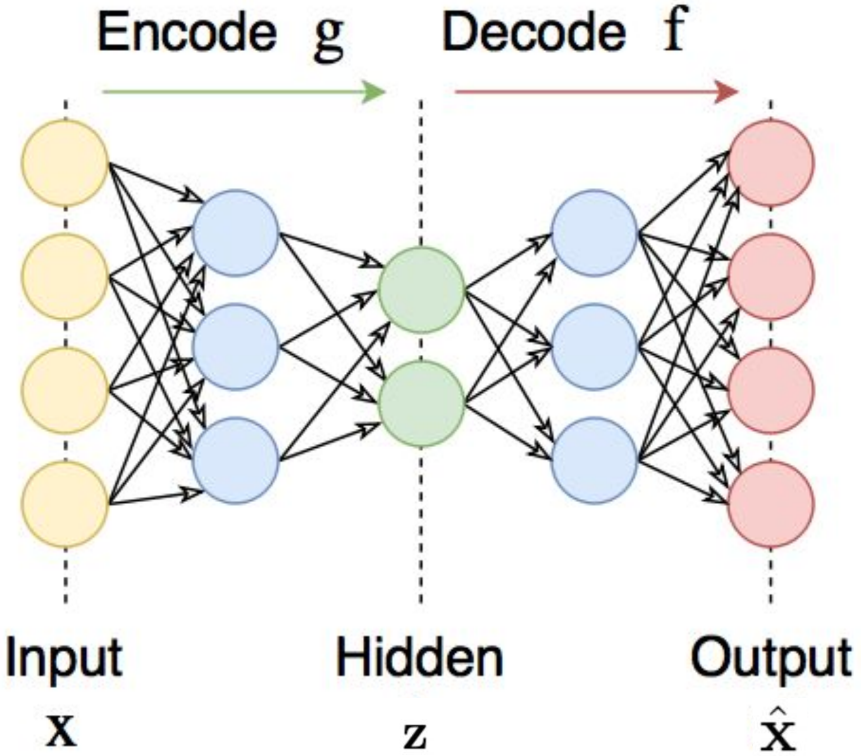


Image source: [Habr post on autoencoders and GANs]()

Autoencoders

Denote **z** as encoded with encoder E input **x**

$$\mathbf{z} = E(\mathbf{x}, \boldsymbol{\theta}_E)$$

Decoder D recovers **x** from latent representation

$$\hat{\mathbf{x}} = D(\mathbf{z}, \boldsymbol{\theta}_D)$$

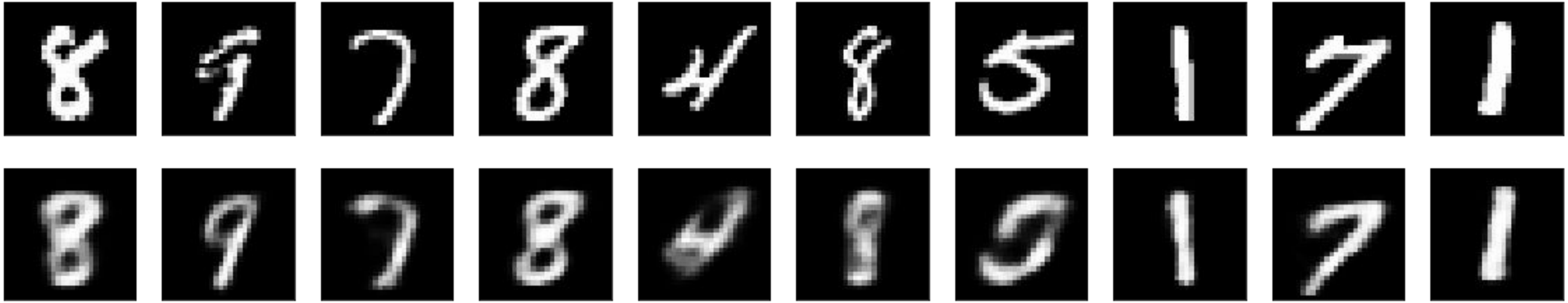
Simple example: PCA

Optimal parameters learned w.r.t. loss function L

$$[\boldsymbol{\theta}_E, \boldsymbol{\theta}_D] = \arg\min L(\hat{\mathbf{x}}, \mathbf{x})$$

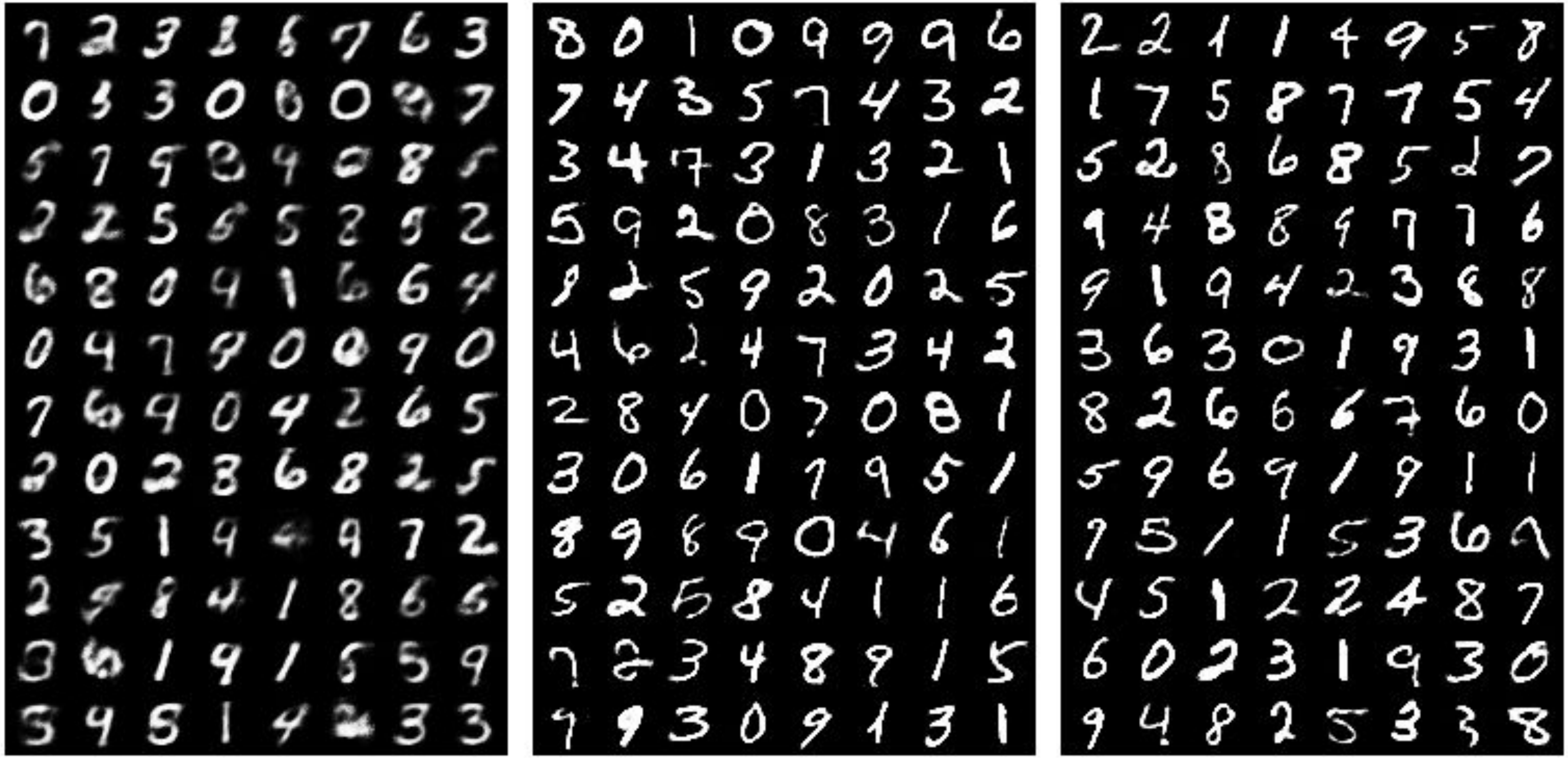
Image source: [Habr post on autoencoders and GANs]()

PCA performance on MNIST



16 components

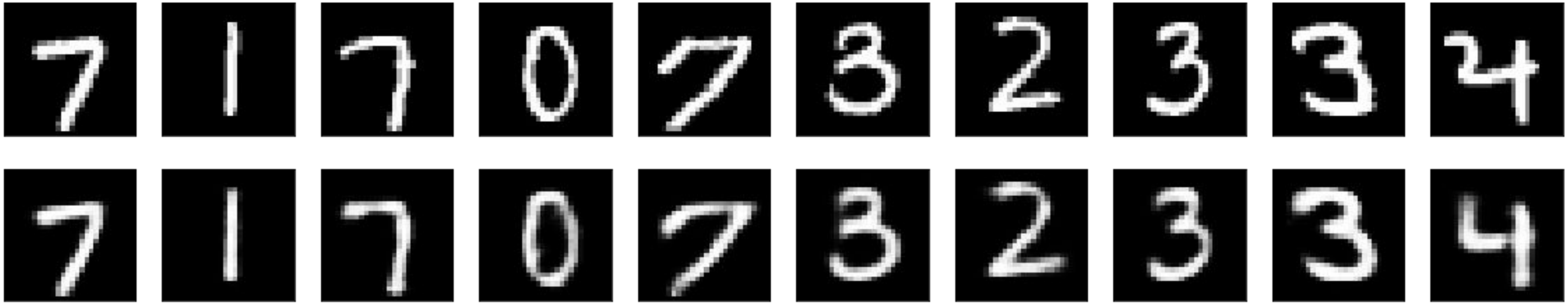
Simple Autoencoder



Epoch 1 Epoch 100 Epoch 200

Image source: https://reyhaneaskari.github.io/AE.htm

Convolutional AE performance on MNIST



7 x 7 latent space

Autoencoders

Denote **z** as encoded with encoder E input **x**

$$\mathbf{z} = E(\mathbf{x}, \boldsymbol{\theta}_E)$$

Decoder D recovers **x** from latent representation

$$\hat{\mathbf{x}} = D(\mathbf{z}, \boldsymbol{\theta}_D)$$

Optimal parameters learned w.r.t. loss function L

$$[\boldsymbol{\theta}_E, \boldsymbol{\theta}_D] = \arg\min L(\hat{\mathbf{x}}, \mathbf{x})$$

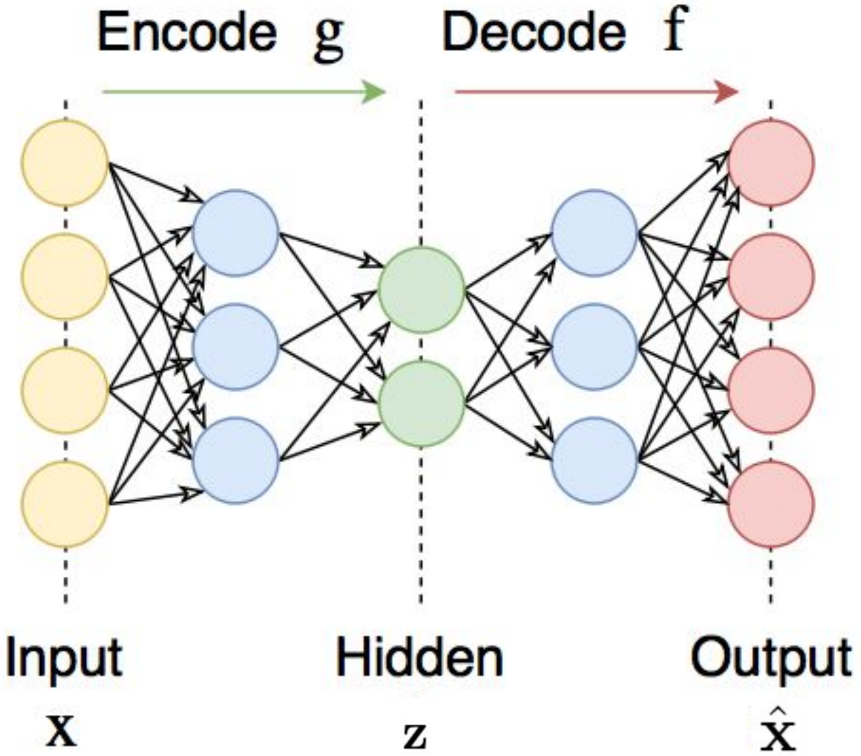
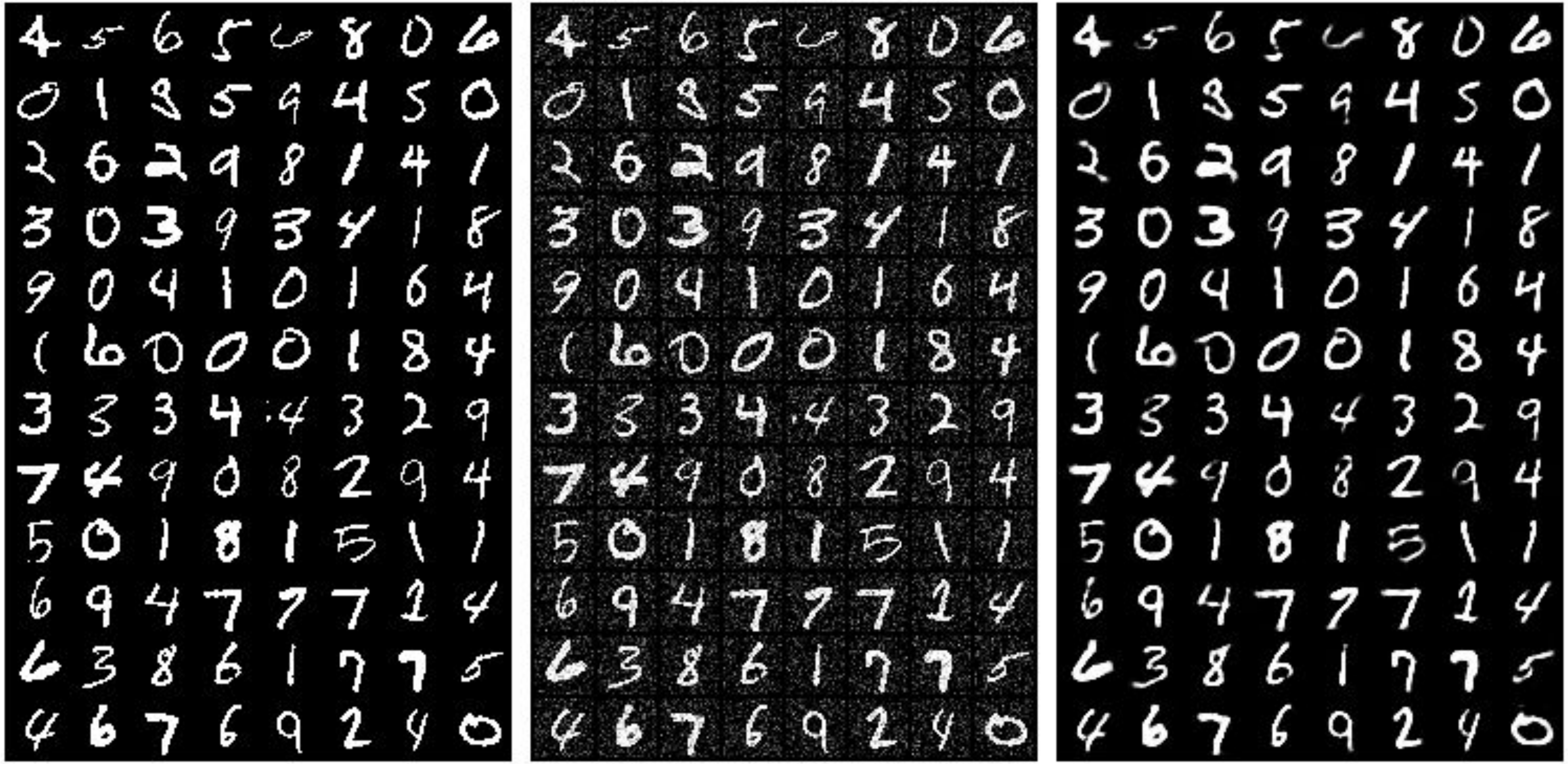


Image source: [Habr post on autoencoders and GANs]()

Denoising Autoencoder



input noisy input output

Image source: https://reyhaneaskari.github.io/AE.htm

Denoising Autoencoder

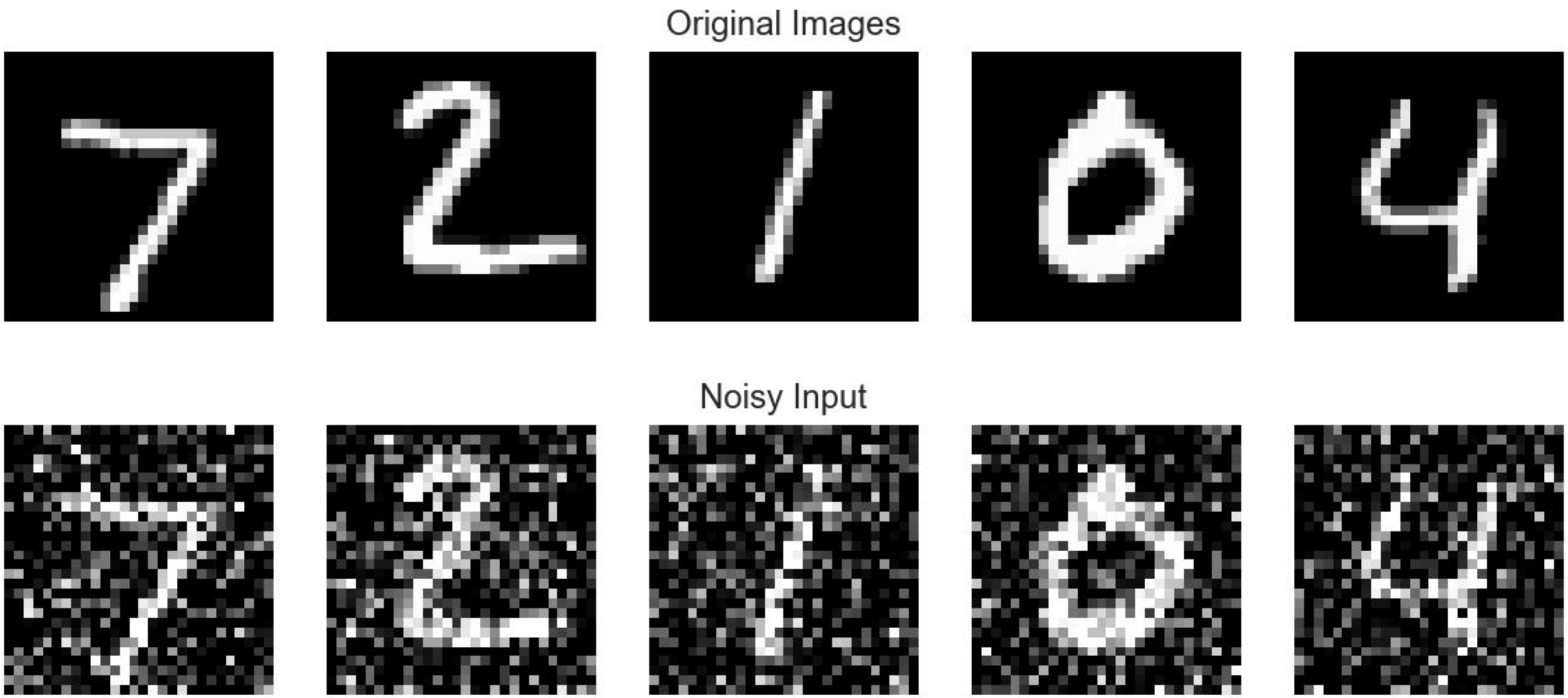


Image source: [towardsdatascience post]

Denoising Autoencoder

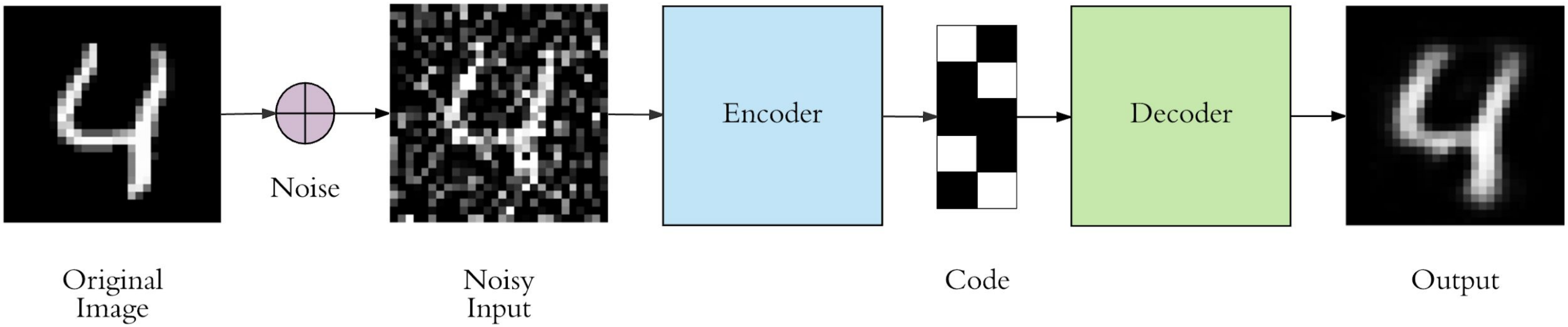


Image source: [towardsdatascience post]

Denoising Autoencoder

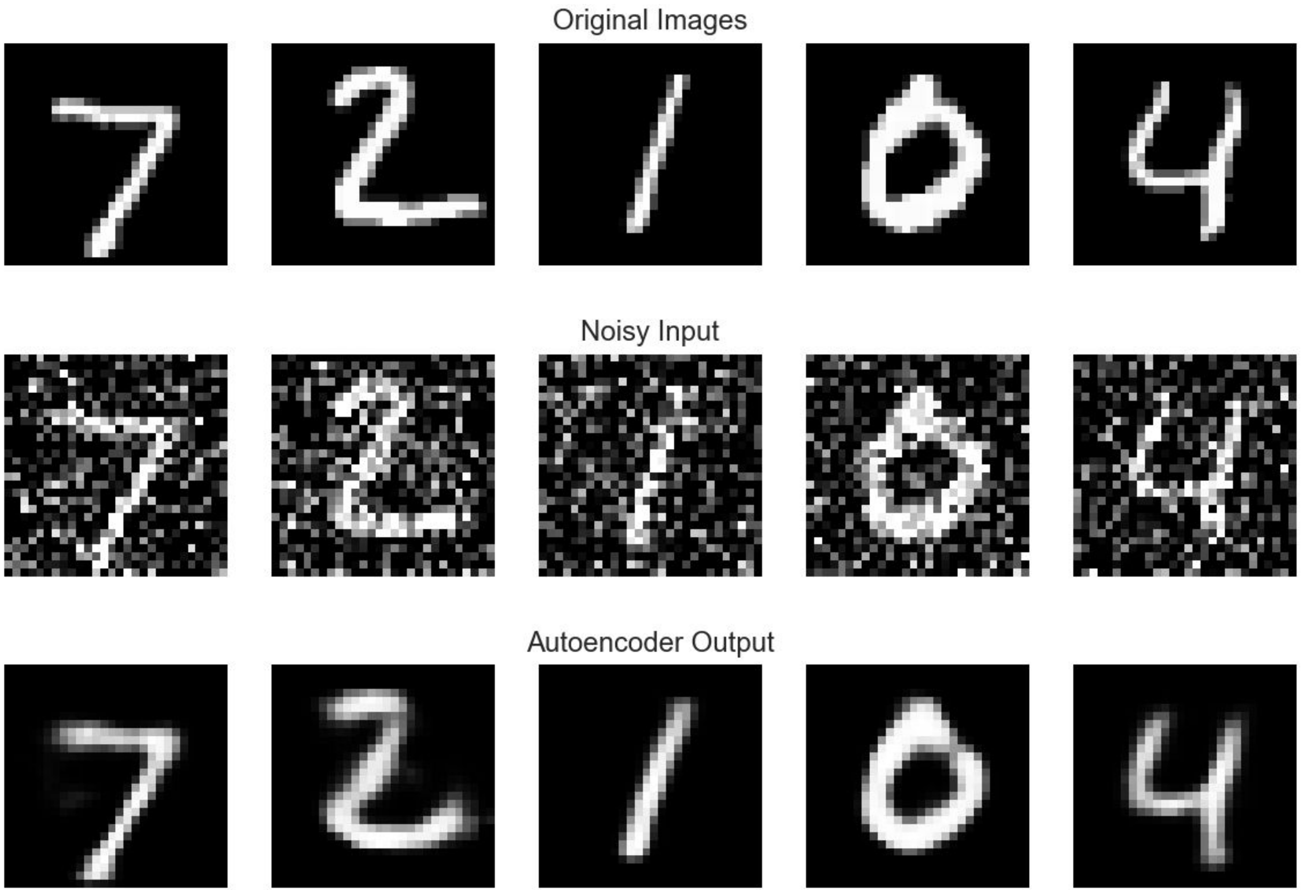


Image source: [towardsdatascience post]

Outro

Autoencoders are used for variety of problems:

- Data compression
- Feature extraction
- Denoising
- Anomaly detection

Remember: *a well-defined problem is halfway to being solved*