Representation Learning

Machine Learning, A.Y. 2022/23, Padova

Fabio Aiolli

November 23rd, 2022

Representation Learning

Representation learning aims at learning representations of input data, typically by transforming it, such to make the input "easier" for a classification or a prediction task.

- **Why is it important?**
  The performance of any machine learning model is critically dependent on the representation it uses.
- **What makes a representation good?** [Bengio et. al., "Representation Learning: A Review and New Perspectives"]
  - Smoothness (e.g., Trans-E)
  - Multiple explanatory factors (e.g., PCA, word2vec)
  - A hierarchical organization of explanatory factors (e.g., CNN)
  - Sparsity (e.g., Sparse AE)
  - ...

Principal Component Analysis (PCA)

PCA tries to answer the question: is there another basis, which is a linear combination of the original basis, that best re-express our data?

PCA seeks to re-express the dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$ (examples on columns) by a simple linear transformation $\mathbf{P} \in \mathbb{R}^{m \times m}$:

$$\mathbf{PX} = \hat{\mathbf{X}}.$$

The equation above is actually a change of basis that can be interpreted in different ways:

- $\mathbf{P}$ is a matrix that transforms $\mathbf{X}$ into $\hat{\mathbf{X}}$;
- Geometrically, $\mathbf{P}$ transforms $\mathbf{X}$ into $\hat{\mathbf{X}}$ via a rotation and a stretching;
- The rows of $\mathbf{P}$ are a new set of basis vectors.

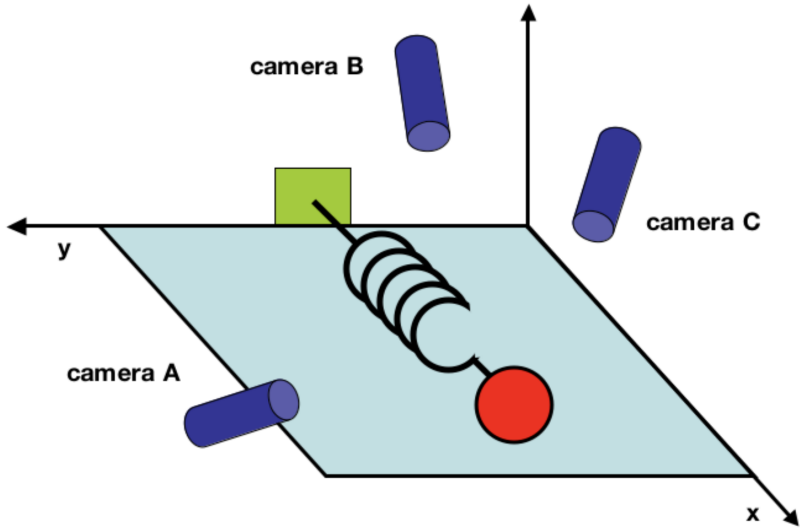
PCA - Searching for a new basis

What is a good choice of basis $\mathbf{P}$?

Two phenomena can potentially "contaminate" our data: noise and redundancy.

$\Rightarrow$ The linear transformation we aim to find is the one that minimizes both the *noise* and the *redundancy* of the signal at hand.

PCA - Noise and redundancy



Noise Any individual camera should record motion in a straight line. Therefore, any spread deviating from straight-line motion must be noise;

Redundancy we are recording the same dynamic from three different points of view. However, this dynamic can be easily described by a single variable.

PCA - Covariance matrix

Both noise and redundancy can be estimated using measures related to the variance of the data.

- The noise of a signal measured by the signal-to-noise ratio (SNR) $SNR = \frac{\sigma_{signal}}{\sigma_{noise}}$;
- Redundancy between features can be measured through their covariance.

Given our dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$ (centered, i.e., with mean 0!) the covariance matrix $\mathbf{S_X} \in \mathbb{R}^{m \times m}$ is computed by

$$\mathbf{S_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^\intercal.$$

PCA - Reducing features covariance

What if we could manipulate $\mathbf{S}_{\hat{\mathbf{X}}}$ *(via* $\mathbf{P}$*) in order to minimize the covariance of the variables?*

In algebraic terms, we are aiming at diagonalizing $\mathbf{S}_{\hat{\mathbf{X}}}$!

Goal of PCA

Find some orthonormal matrix $\mathbf{P}$ *where* $\hat{\mathbf{X}} = \mathbf{P}\mathbf{X}$ *such that*

$$\mathbf{S}_{\hat{\mathbf{X}}} = \frac{1}{n}\hat{\mathbf{X}}\hat{\mathbf{X}}^\intercal$$

is a diagonal matrix.

PCA - Diagonalizing $\mathbf{S}_{\hat{\mathbf{X}}}$ (1/2)

Let us rewrite $\mathbf{S}_{\hat{\mathbf{X}}}$ in terms of $\mathbf{P}$:

$$\begin{aligned}\mathbf{S}_{\hat{\mathbf{X}}} &= \frac{1}{n}\hat{\mathbf{X}}\hat{\mathbf{X}}^\intercal = \frac{1}{n}(\mathbf{PX})(\mathbf{PX})^\intercal \\ &= \frac{1}{n}\mathbf{P}(\mathbf{X}\mathbf{X}^\intercal)\mathbf{P}^\intercal = \frac{1}{n}\mathbf{PAP}^\intercal\end{aligned}$$

where $\mathbf{A} = \mathbf{X}\mathbf{X}^\intercal$ is a symmetric matrix.

From linear algebra we known that for a symmetric matrix $\mathbf{A}$ it holds that

$$\mathbf{A} = \mathbf{EDE}^\intercal$$

where $\mathbf{D} \in \mathbb{R}^{m \times m}$ is a diagonal matrix (eigenvalues) and $\mathbf{E} \in \mathbb{R}^{m \times m}$ is a matrix of eigenvectors of $\mathbf{A}$ arranged as columns.

PCA - Diagonalizing $\mathbf{S}_{\hat{\mathbf{X}}}$ (2/2)

Since we want $\mathbf{P}$ being orthonormal $\Rightarrow \mathbf{P} \equiv \mathbf{E}^\intercal$!

Finally by using the fact that being orthonormal holds the property $\mathbf{P}^\intercal = \mathbf{P}^{-1}$ we can compute

$$\begin{aligned}\mathbf{S}_{\hat{\mathbf{X}}} &= \frac{1}{n}\mathbf{P}\mathbf{A}\mathbf{P}^\intercal \\ &= \frac{1}{n}\mathbf{P}(\mathbf{P}^\intercal\mathbf{D}\mathbf{P})\mathbf{P}^\intercal = \frac{1}{n}(\mathbf{P}\mathbf{P}^\intercal)\mathbf{D}(\mathbf{P}\mathbf{P}^\intercal) \\ &= \frac{1}{n}(\mathbf{P}\mathbf{P}^{-1})\mathbf{D}(\mathbf{P}\mathbf{P}^{-1}) = \frac{1}{n}\mathbf{D}\end{aligned}$$

which clearly diagonalizes $\mathbf{S}_{\hat{\mathbf{X}}}$.

The rows of $\mathbf{P}$, i.e., the eigenvectors of $\mathbf{X}\mathbf{X}^\intercal$, are called **principal components**.

PCA - Dimensionality reduction

- Often the most interesting dynamics occur only in the first k dimensions, e.g., in the example of the spring, $k = 1$;
- Throwing out the less important axes can help reveal hidden, simplified dynamics in high dimensional data. This process is aptly named dimensional reduction.

Formally, given $k \ll m$ the dimensionality of $\mathbf{X}$ is reduced by

$$\mathbf{P}_k\mathbf{X} = \mathbf{X}_k \in \mathbb{R}^{k \times n}$$

where $\mathbf{P}_k$ are a sub-matrix of $\mathbf{P}$ containing the k rows corresponding to the k indexes in the diagonal of $\mathbf{D}$ associated with the highest eigenvalues.

Interestingly, $\mathbf{P}_k$ is also the minimizer of the squared reconstruction error:

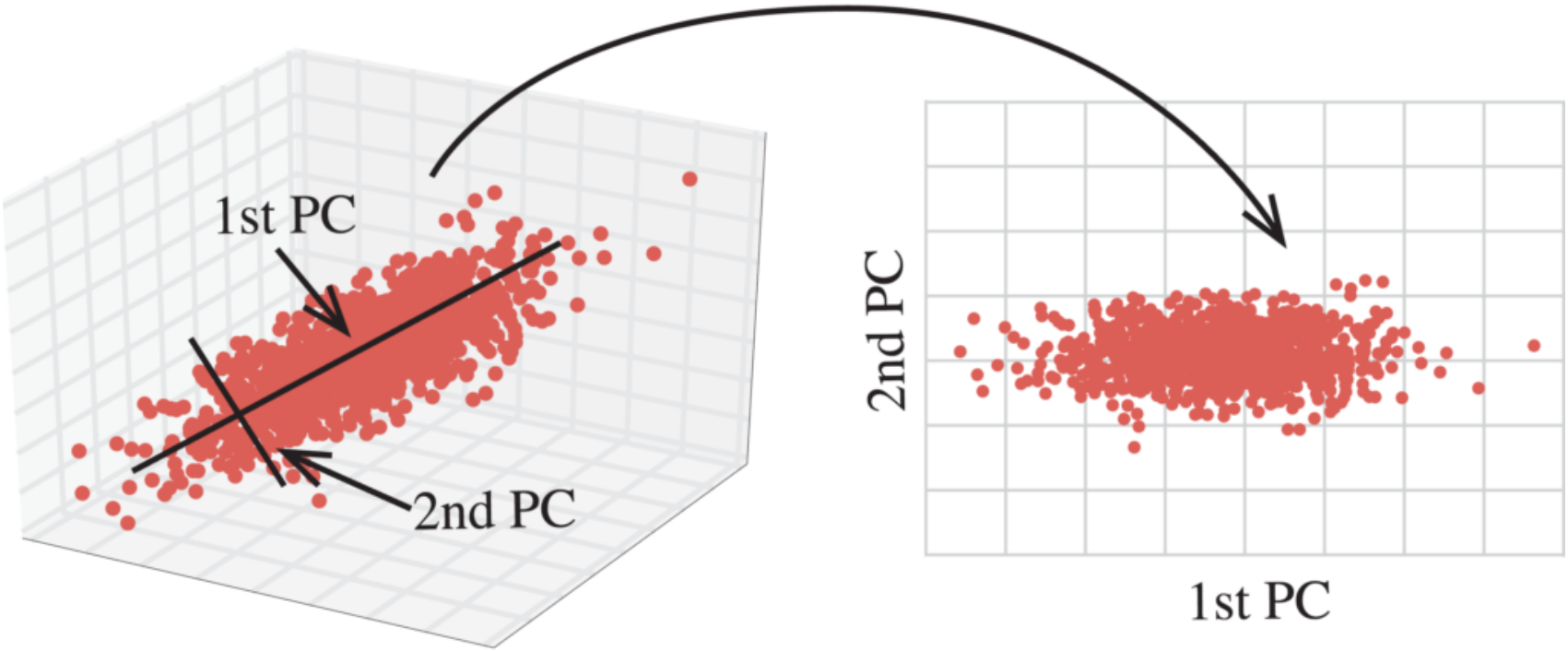
$$\min_{\mathbf{P} \in \mathbb{R}^{k \times m}} \|\mathbf{X} - \mathbf{P}^\intercal\mathbf{P}\mathbf{X}\|_F^2 \quad s.t.\ \mathbf{P}\mathbf{P}^\intercal = \mathbf{I}_{k \times k}.$$

PCA - Algorithm overview

Given a dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$ (instances on the columns):

1. center the data $\mathbf{X}$ by subtracting off the mean of each measurement type;
2. compute the eigenvectors of $\mathbf{X}\mathbf{X}^\intercal$, i.e., $\mathbf{E}\mathbf{D}\mathbf{E}^\intercal$ and posing $\mathbf{P} = \mathbf{E}^\intercal$;
3. select the $k \ll m$ most important components from $\mathbf{P}$ according to $\mathbf{D}$.

PCA - Example



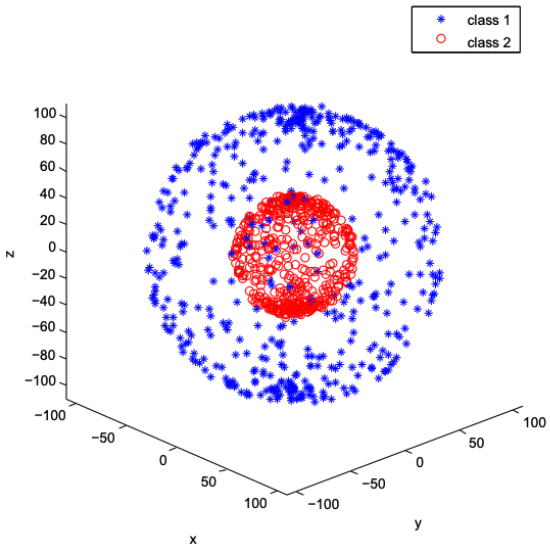
Kernel PCA

Extends conventional PCA to deal with non-linear correlations using the **kernel trick** (like SVM does).

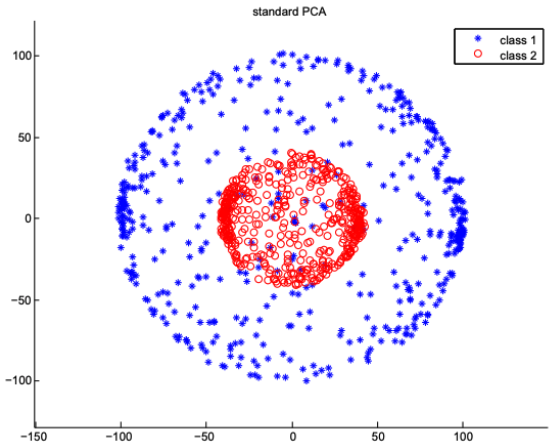
High-level algorithm:

1. Instead of using data points $\mathbf{X}$ directly, we first map them ($\phi(\mathbf{X})$) *implicitly* to some nonlinear feature space;
2. Center the data in the feature space: it can be done implicitly in terms of kernels only!
3. Implicitly extract principal component in that space, i.e., apply PCA in the feature space; Projections on these principal components can be computed in terms of kernels only
4. The result will be non-linear transformations in the original data space!

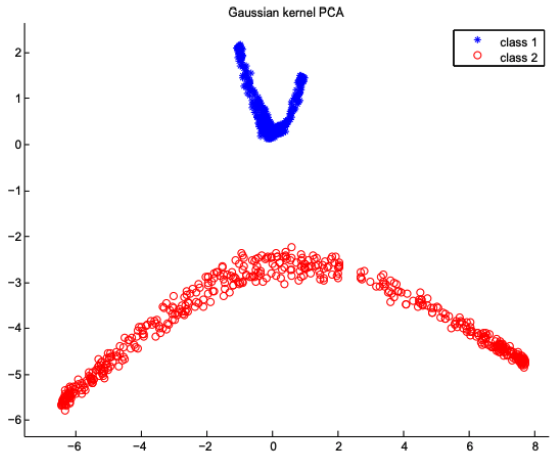
Kernel PCA vs Linear PCA



Input space 3D



Standard PCA



Kernel PCA

PCA - Applications

Some of the possible scenarios in which PCA can be useful:

- Apply *lossy compression*;
- *Visualize* the multi-dimensional data in 2D or 3D;
- Reduce the number of dimensions to *discard noisy features*;
- Perform a change of representation in order to make analysis of the data at hand.

Autoencoders - Introduction

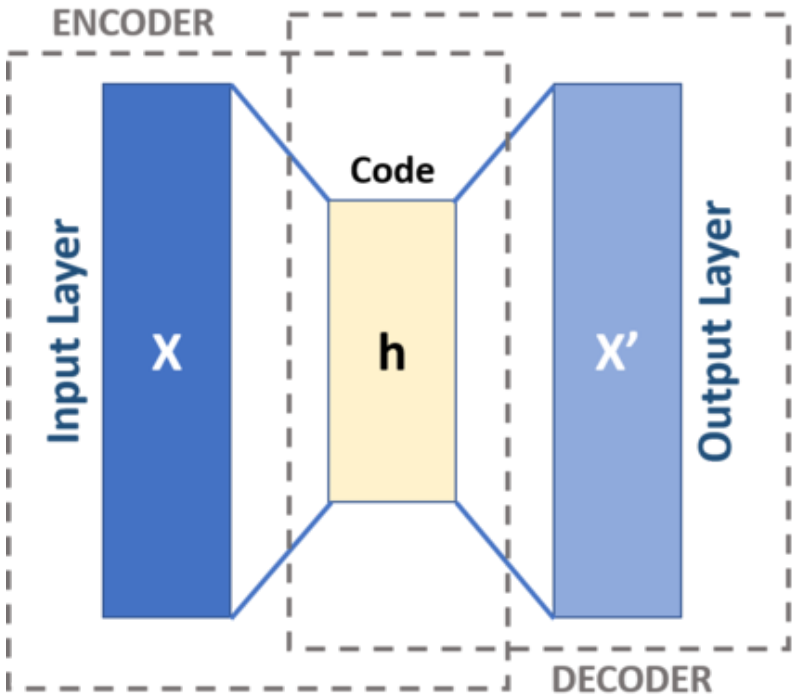
- Autoencoders (AEs) are an unsupervised learning technique based on feed forward neural networks;
- Historically, the sole aim of an autoencoder was to learn a representation (often called, *encoding* or *code*) for a set of data, typically for the purpose of dimensionality reduction;
- Recently, the autoencoder concept has become more widely used for learning **generative models** of data (Variational Autoencoders);

Goal of autoencoders

The goal of an autoencoder network is to "copy" the input on its output.

Autoencoders - Overview

- Encoder: creates a new representation (the **code**) of the input;
- Decoder: reconstructs the input starting from the code;
- Bottleneck layer: fundamental to compress the data and make the task harder!



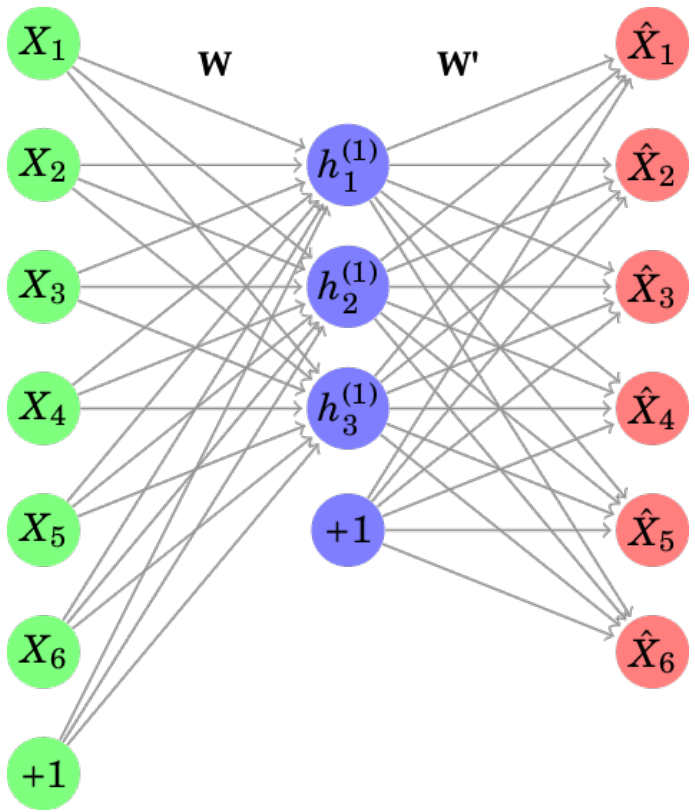
Autoencoders - Easiest architecture

In its easiest form an autoencoder has:

- a single hidden layer;
- a set of parameters $\mathbf{W}$, $\mathbf{W}'$ defined by two weight matrices and two bias vectors:

$$\mathbf{h} = f_{\mathbf{W}}(\mathbf{x}) = s_1(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\hat{\mathbf{x}} = g_{\mathbf{W}'}(\mathbf{h}) = s_2(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

  where s_1 and s_2 denote the activation functions, which usually are nonlinear;



Autoencoders - Loss and training objective

- The standard loss function is a mean squared error loss (called in general reconstruction loss)

$$\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2,$$

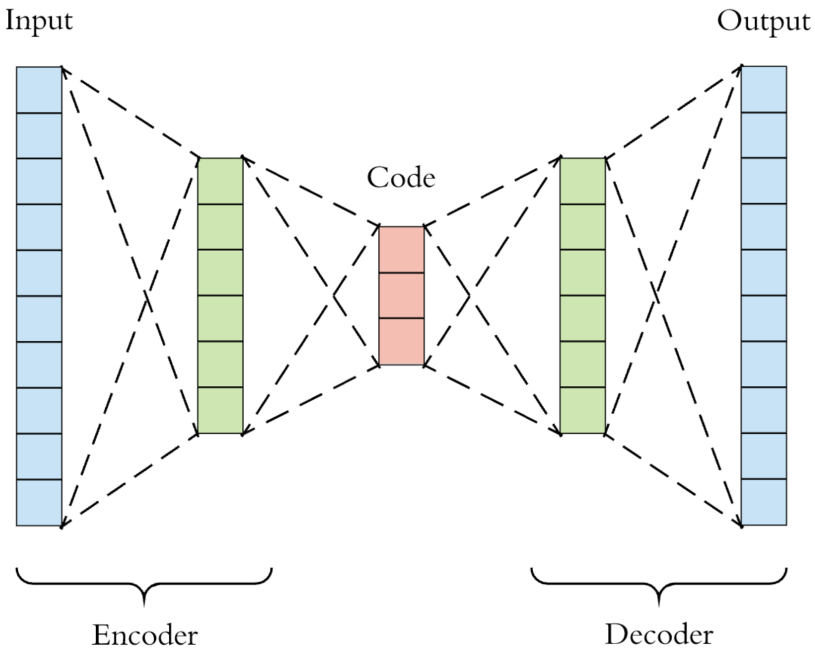
  where $\tilde{\mathbf{x}} = g_{\mathbf{W}'}(f_{\mathbf{W}}(\mathbf{x}))$;
- The final objective function that AEs aim to optimize is:

$$\mathcal{J}(\mathbf{W}, \mathbf{W}'; \mathbf{X}) = \sum_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(\mathbf{x}, g_{\mathbf{W}'}(f_{\mathbf{W}}(\mathbf{x}))).$$

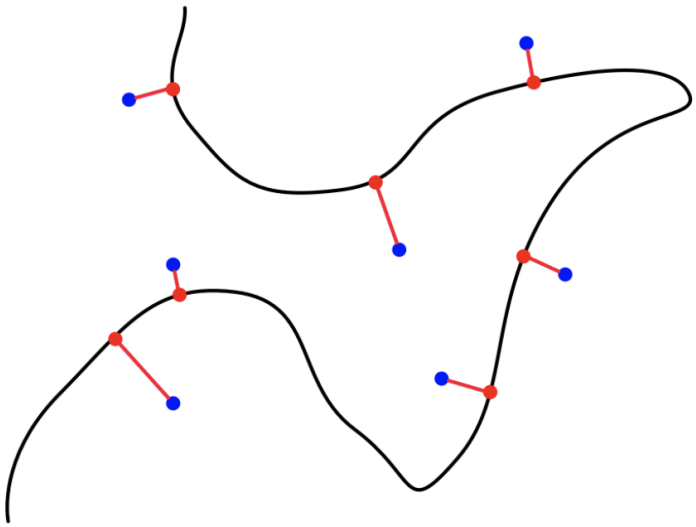
- The learning procedure is performed using standard tools for training neural networks, such as, for example backpropagation and **Stochastic Gradient Descent (SGD)**.

Deep (non-linear) autoencoders

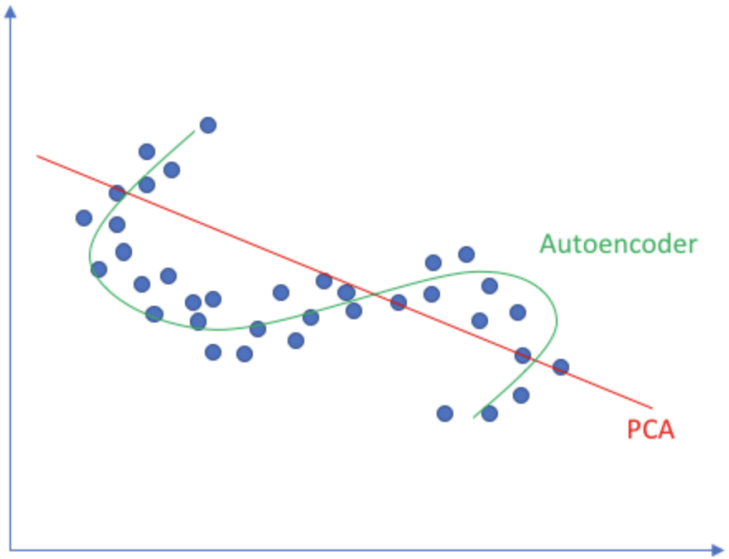
- **Deep** $\Leftarrow$ add more intermediate layers between the input and the code (as well as between the code and the output);
- **Non-linear** $\Leftarrow$ set non-linear activation functions to the neurons;
- Deep nonlinear AEs learn to project the data onto a nonlinear manifold (i.e., the image of the decoder);
- This can be seen as a kind of non-linear dimensionality reduction.



Autoencoders and PCA



(a) Non linear manifold



(b) AE vs PCA

Regularized autoencoders

Encodings produced by basic AEs do not generally present **any special properties**. Additionally, like many other machine learning techniques, AEs are susceptible to **overfitting**.

$\Rightarrow$ **Regularized autoencoders** use a loss function that encourages the model to have other properties besides the ability to copy its input to its output:

- Sparsity of the representation;
- Robustness to noise or to missing inputs;
- Smallness of the derivative of the representation.

The general form of a regularized AE objective function is:

$$\mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\theta}'; \mathbf{X}) = \sum_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(\mathbf{x}, g_{\boldsymbol{\theta}'}(f_{\boldsymbol{\theta}}(\mathbf{x}))) + \Omega(\boldsymbol{\theta}, \boldsymbol{\theta}'; \mathbf{X}).$$

Sparse autoencoders

- Sparsity in a representation means most values for a given sample are zero or close to zero;
- A way to achieve sparsity in the activation of the hidden unit, is by applying L1 or L2 regularization terms on the activation.
- For instance, in the case of L1 the loss function would become:

$$\Omega(\boldsymbol{\theta}, \boldsymbol{\theta}'; \mathbf{X}) = \lambda \sum_i |h_i|,$$

where h_i is the i-th entry of the code vector $\mathbf{h}$, and $\lambda \geq 0$ is a **regularization hyper-parameter**.

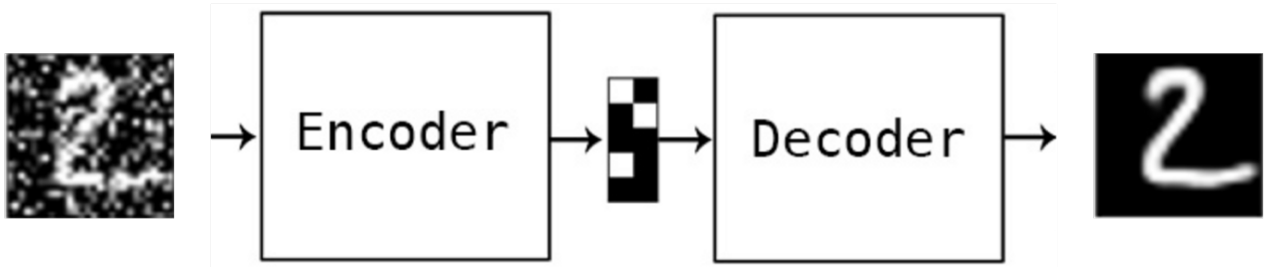
Denoising autoencoders

The objective of Denoising autoencoders (DAE) is that of cleaning the corrupted input, or denoising.

- Higher level representations are relatively stable and robust to the corruption of the input.

The training process of a DAE works as follows:

- The input $\mathbf{x}$ is corrupted into $\mathbf{x}'$ through stochastic mapping, $\mathbf{x}' \sim q(\mathbf{x}'|\mathbf{x})$;
- $\mathbf{x}'$ is then mapped to a hidden representation $\mathbf{h} = f_{\boldsymbol{\theta}}(\mathbf{x}')$;
- Finally the model reconstructs the input $\mathbf{x} \approx \tilde{\mathbf{x}} = g_{\boldsymbol{\theta}'}(\mathbf{h})$.



Convolutional Neural Networks (CNN)

- Convolutional networks (LeCun, 1989) are a specialized kind of NN for processing data that has a known grid-like topology;
- The name CNN indicates that the network employs a mathematical operation called convolution;
- Inspired by the receptive fields of photoreceptors in the human eye, CNNs consider groups of neighboring features, e.g., pixels (local connectivity);
- CNN learns different levels of abstraction of the input, e.g., images:
  - in the first few hidden layers, the CNN usually detects **general patterns**, like edges;
  - the deeper we go into the CNN, these learned abstractions become **more specific**, like textures, patterns and (parts of) objects

Convolution for single-channel images

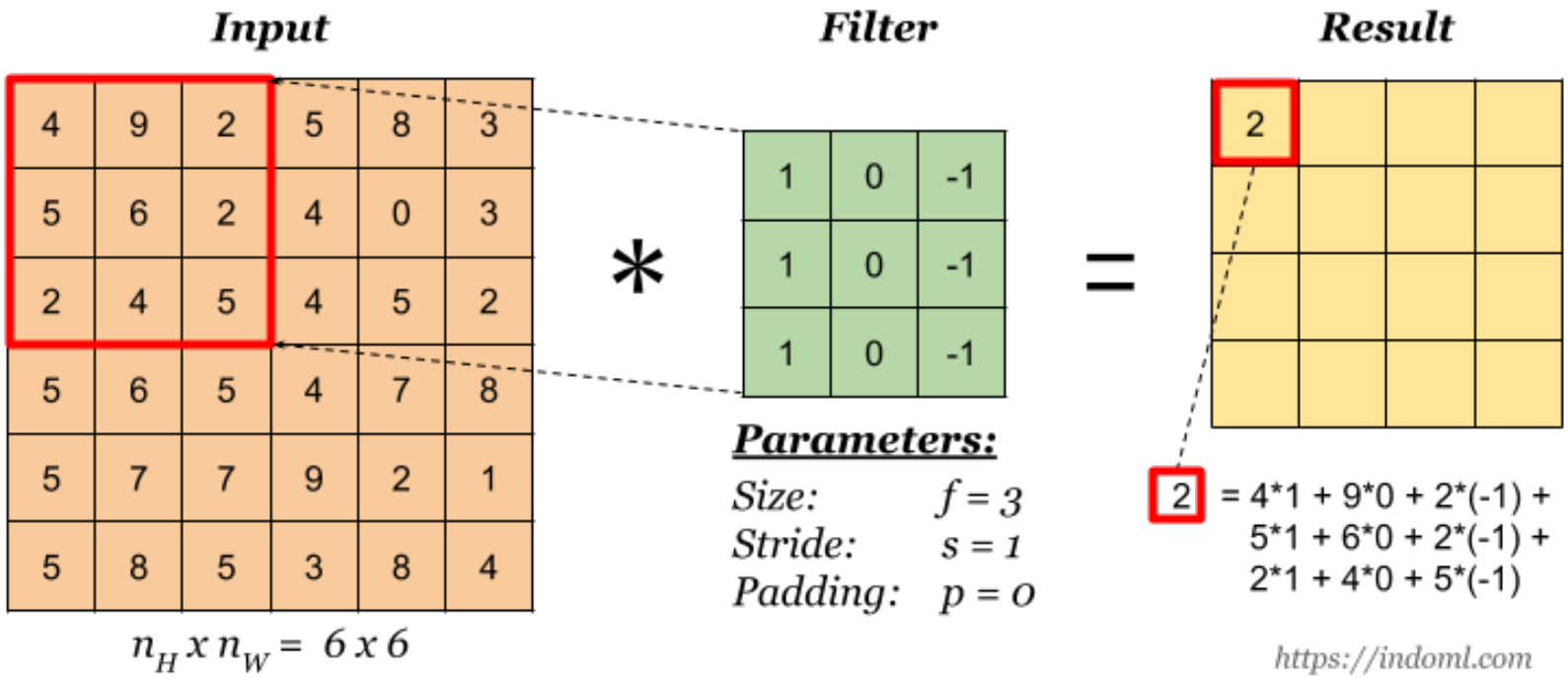


Figure: Example of convolution of an "image" with a *left sobel* convolution kernel.

Convolution for multi-channel images

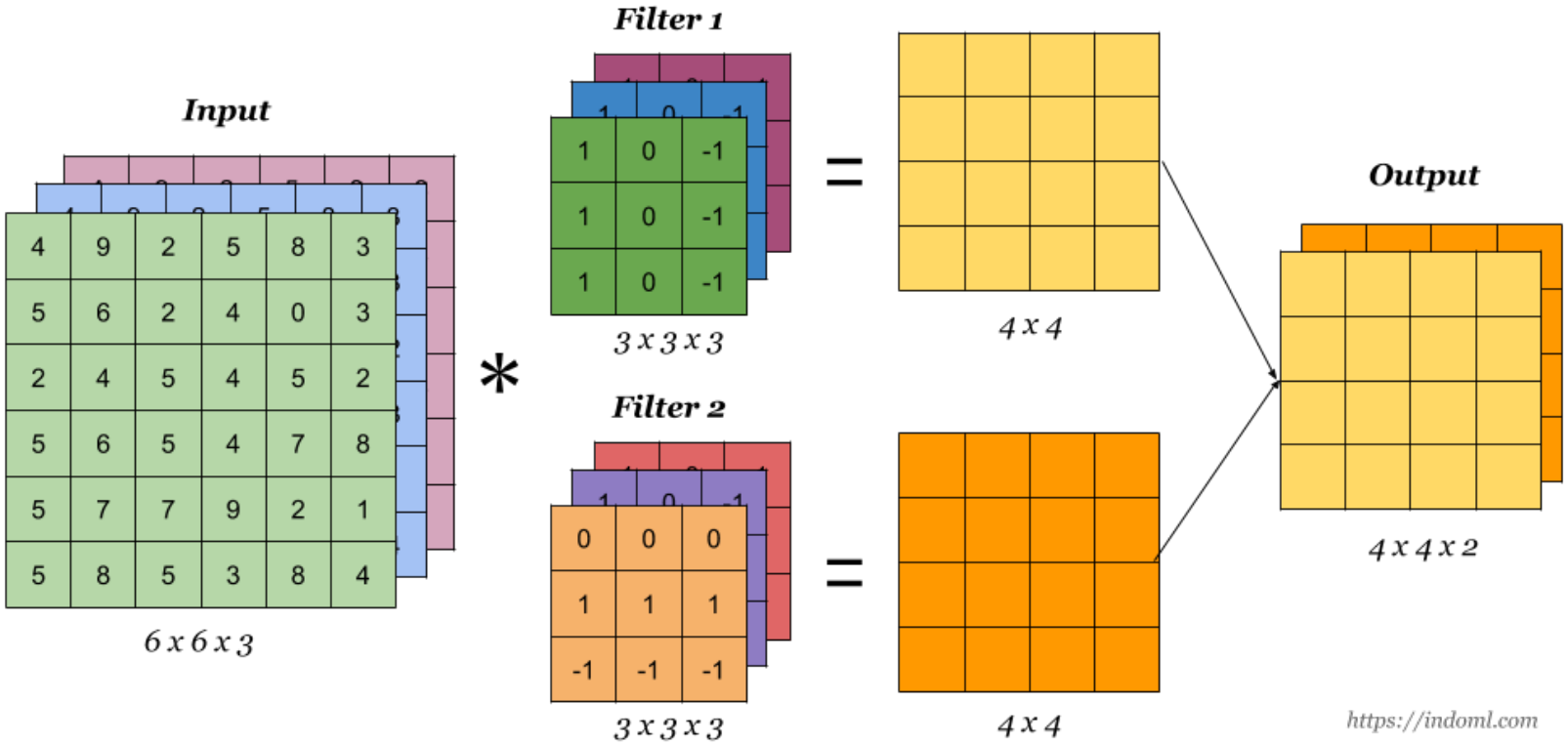


Figure: Example of application of 2 convolution filters to an "image" with 3 channels.

Convolution layer

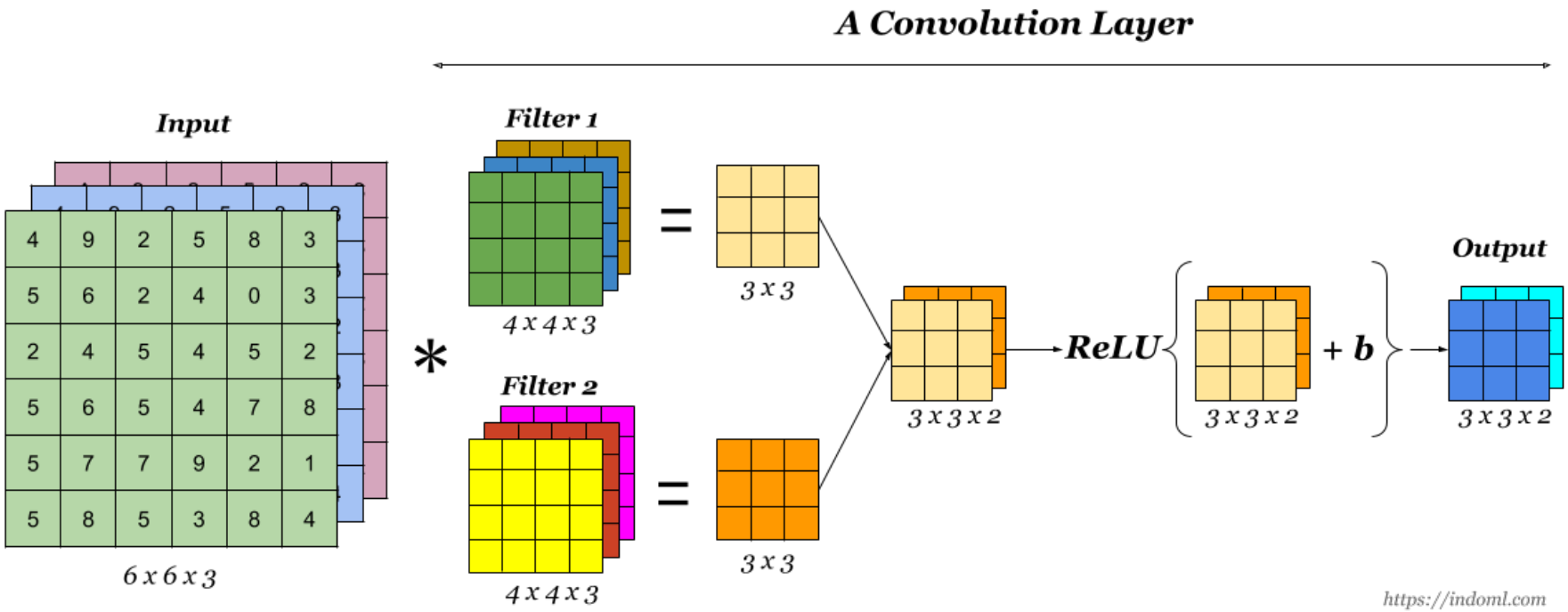
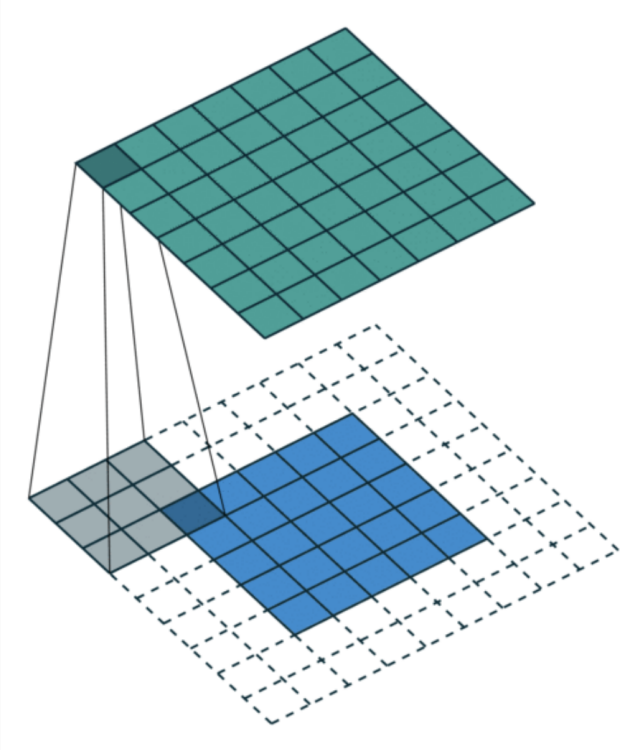
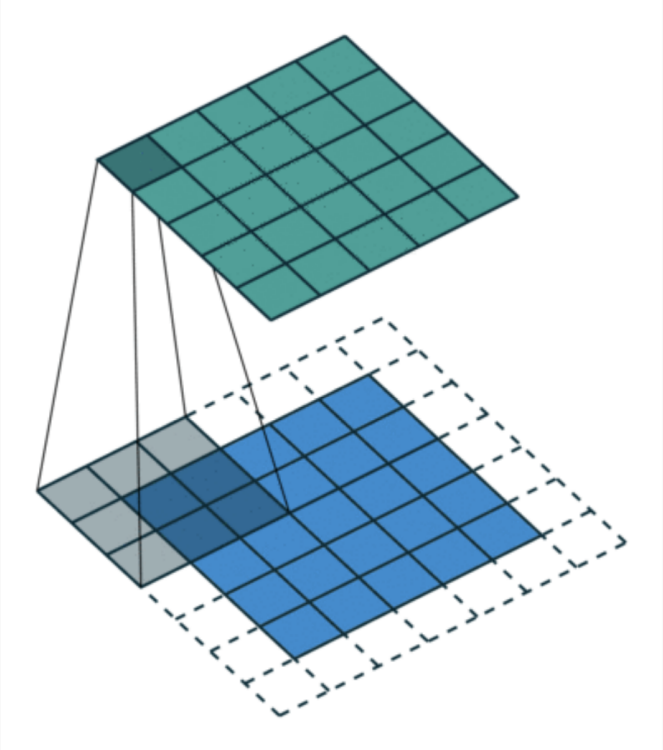


Figure: Example of a layer of a convolutional neural network.

Padding



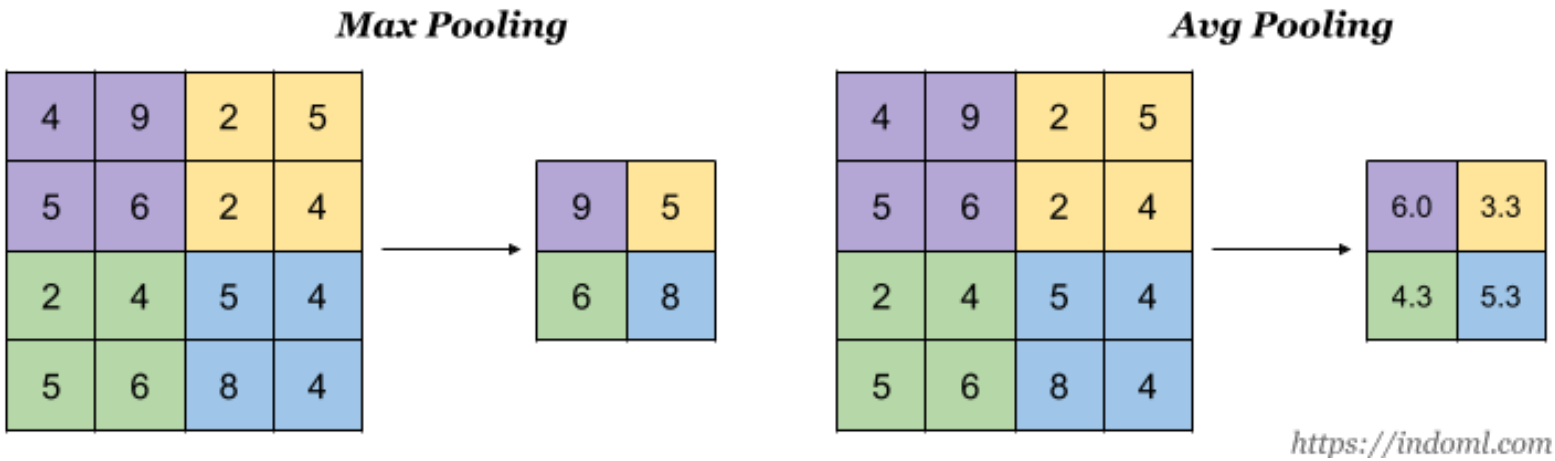
Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



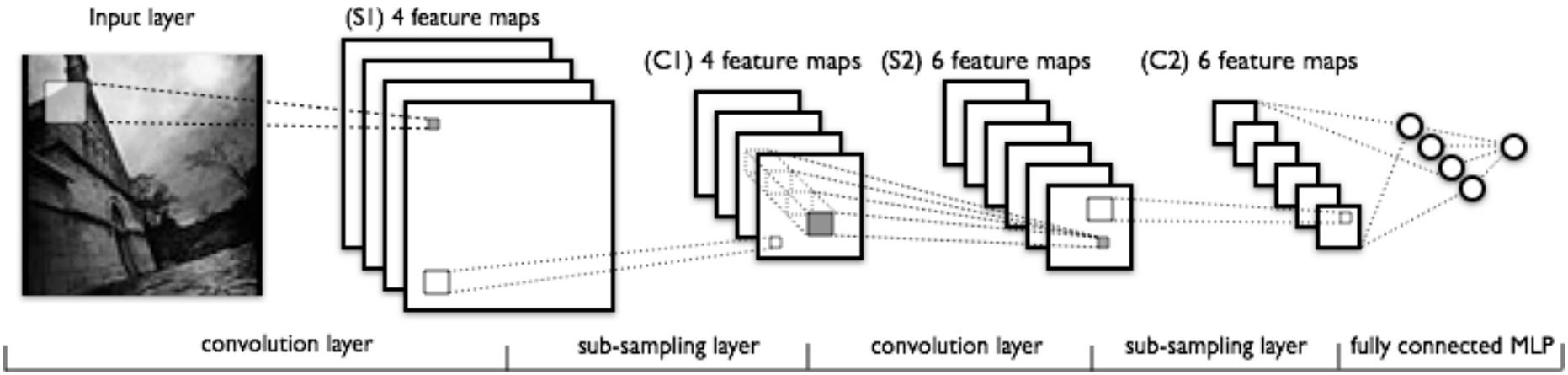
Same padding. Ensures that the output has the same size as the input.

Pooling

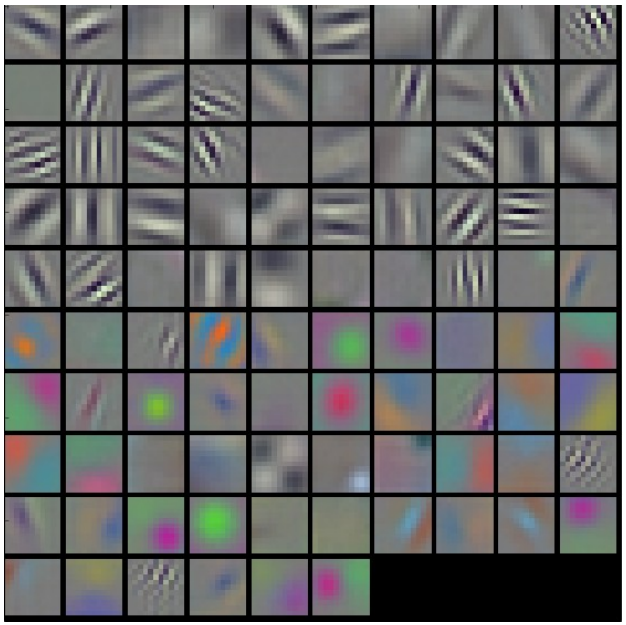
Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.

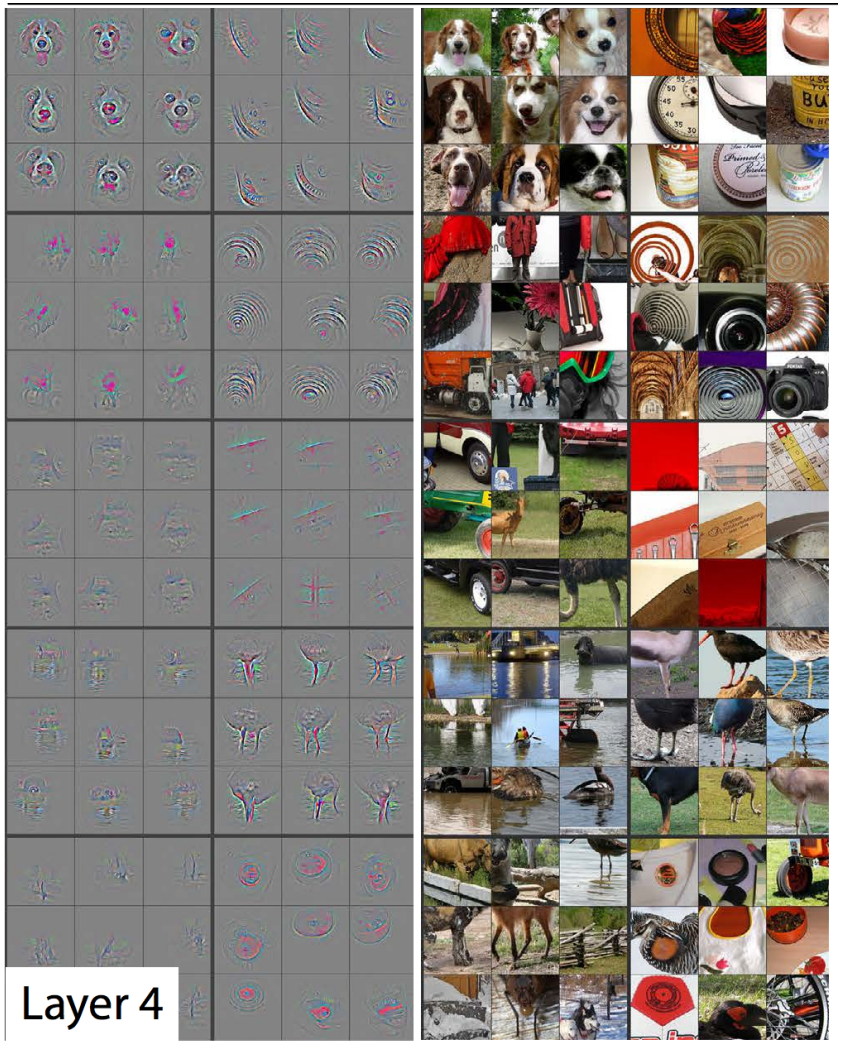


Typical CNN architecture



Visualizing CNN





(Neural) Word Embeddings - Word2Vec

- (Neural) Word embedding belongs to the text preprocessing phase $\Rightarrow$ transforms a text (set of words) into a vector of numbers;
- The word2vec technique is based on a feed-forward, fully connected architecture;
- Word2vec is similar to an autoencoder, encoding each word in a vector, but rather than training against the input words through reconstruction, word2vec trains words against other words that **neighbor** them in the input corpus (called context).

Goal of word2vec

word2vec aims at computing vector representations of words able to capture semantic and syntactic word similarity.

Word2Vec - Word context

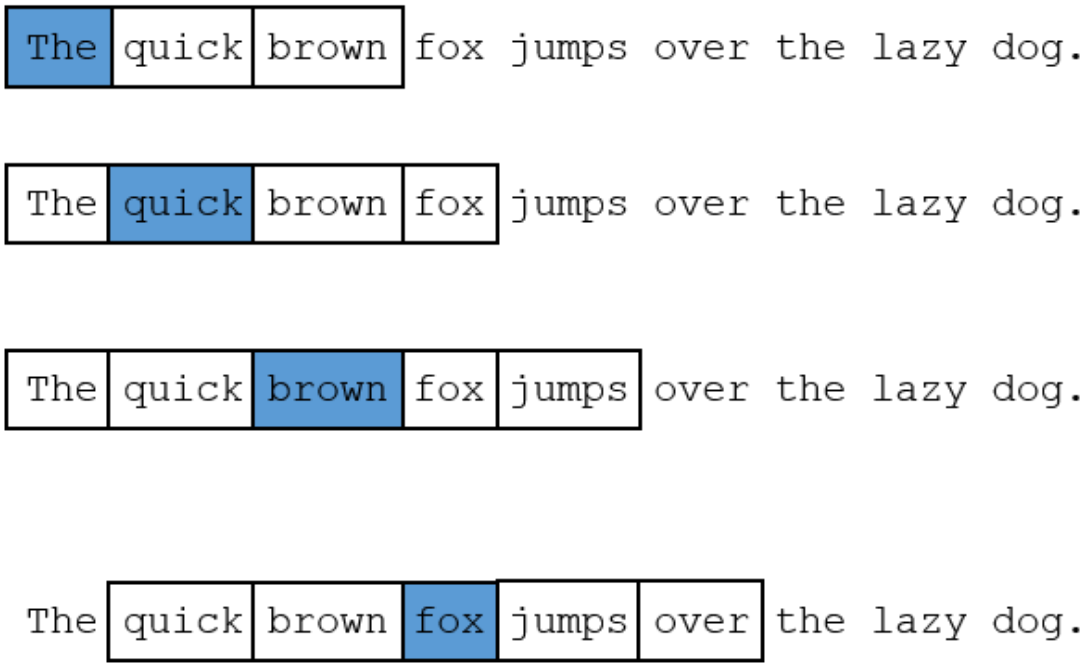


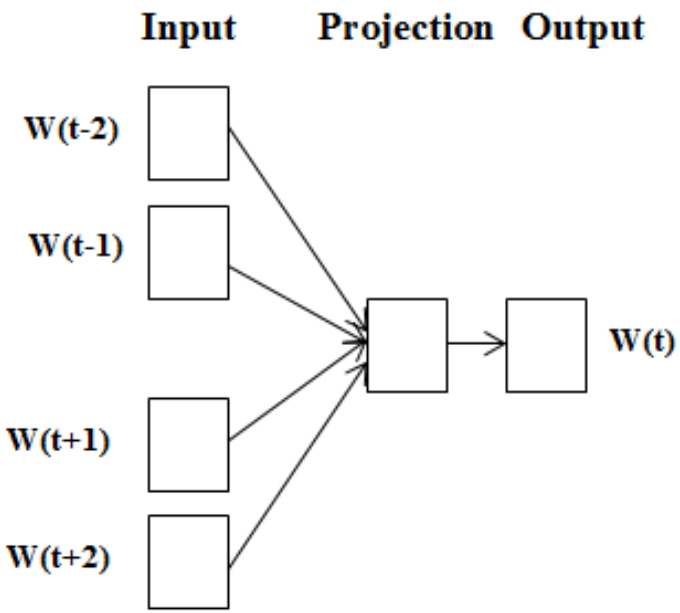
Figure: Example of word context: window size = 2 - both the **forward context** and the **backward context** are composed of 2 words.

Word2Vec - Fake tasks

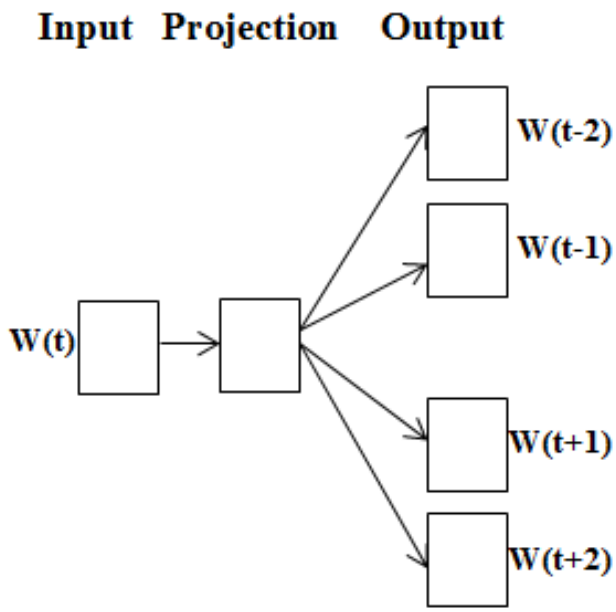
By using this concept of context word2vec can learn word embeddings in two different ways:

CBOW (Continuous Bag Of Words) in which the network uses the context to predict a target word;

Skip-gram in which it uses the target word to predict a target context.



CBOW



Skip-gram

Word2Vec - Skip-gram

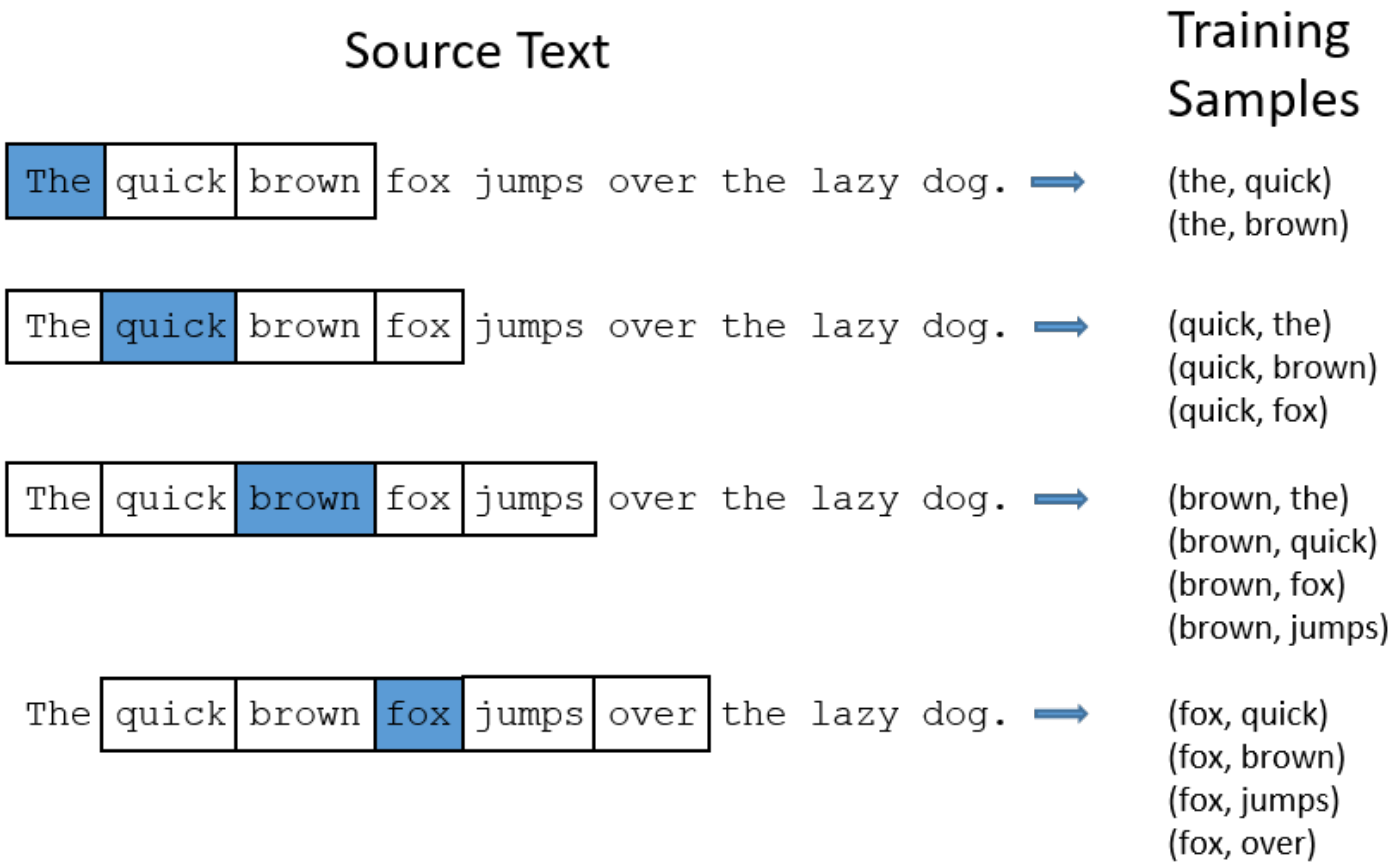


Figure: Example of training samples for the skip-gram model with context size =2.

Word2Vec w/ skip-gram - NN architecture

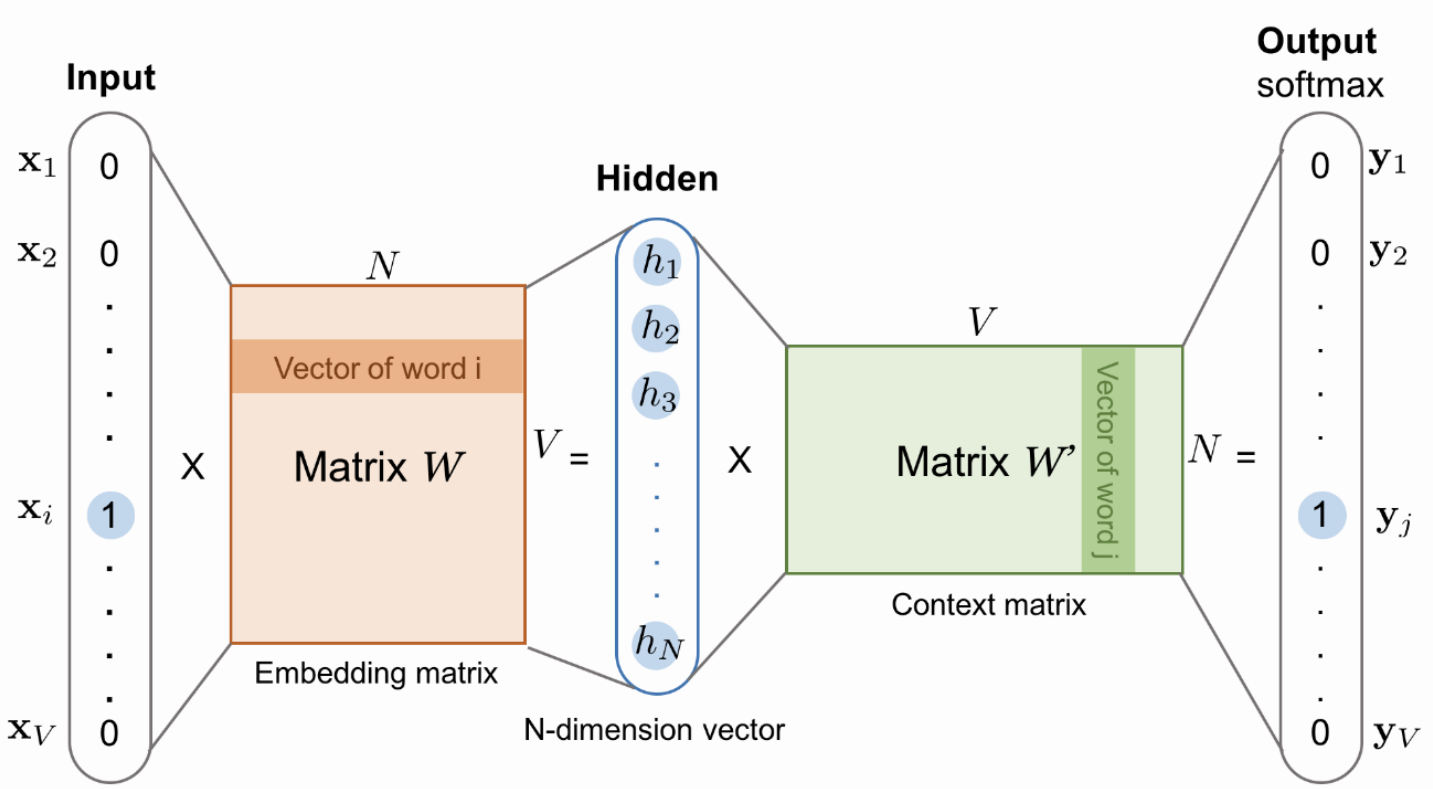


Figure: Example of architecture for word2vec based on the *skip-gram* fake task.

Word2Vec w/ skip-gram - Model overview

- Input: one-hot encoding of the word according to a **vocabulary**;
- Output: distribution over the words of being in the context of the target word;
- Formal model ($\sigma(\cdot)$ is the *softmax* function):

$$\mathbf{h} = \mathbf{W}^\top \mathbf{x}$$
$$\mathbf{u} = \mathbf{W}'^\top \mathbf{h} = \mathbf{W}'^\top \mathbf{W}^\top \mathbf{x}$$
$$\mathbf{y} = \sigma(\mathbf{u}) = \sigma(\mathbf{W}'^\top \mathbf{W}^\top \mathbf{x})$$

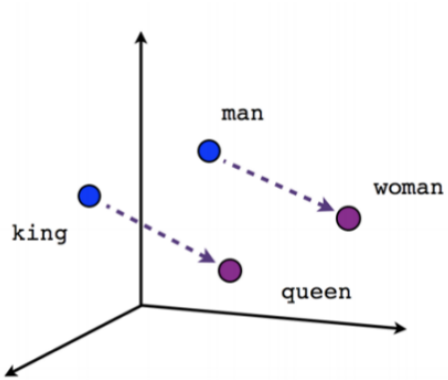
- The objective of the skip-gram model is to maximize the average log probability, i.e.,

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} \mid w_t).$$

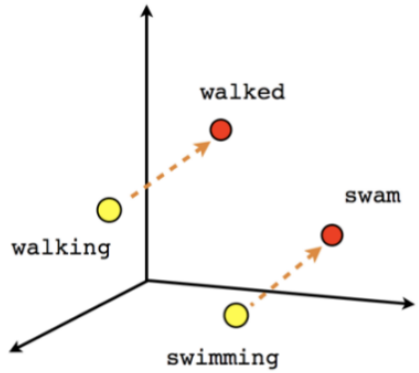
Word2Vec - Semantic and syntactic relations

- Word2vec captures multiple different degrees of similarity between words;
- Patterns such as "*Man is to Woman as Brother is to Sister*" can be generated through algebraic operations on the vector representations

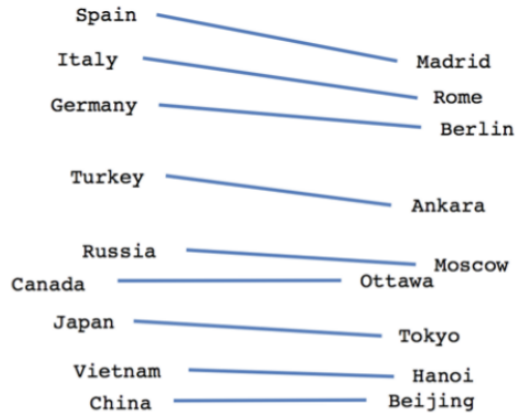
$$\text{"}Brother\text{"} - \text{"}Man\text{"} + \text{"}Woman\text{"} \approx \text{"}Sister\text{"}.$$



Male-Female



Verb tense



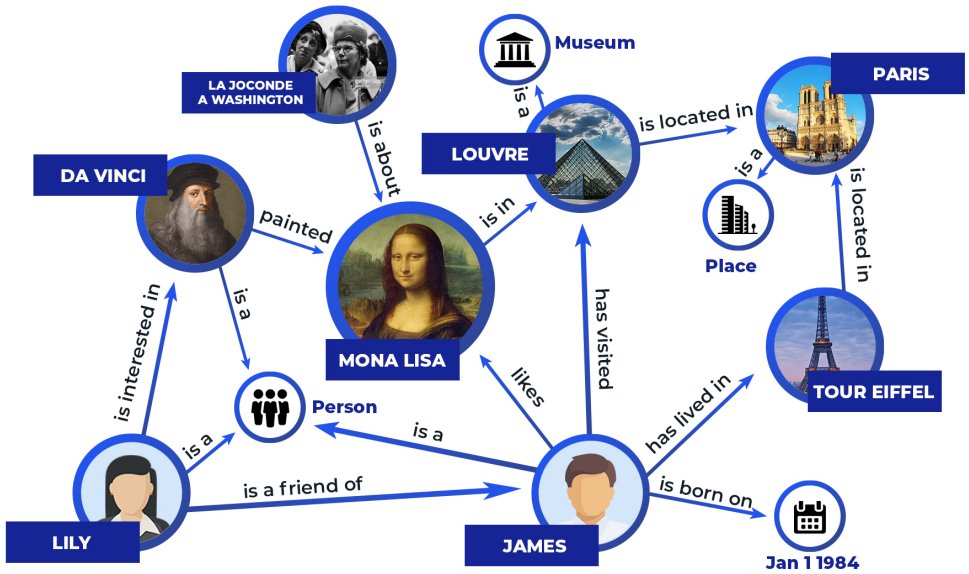
Country-Capital

Knowledge graph

- A Knowledge graph (KG) is a multi-relational graph composed of entities (nodes) and relations (edges);
- Edges are represented as triples of the form

$$(\textit{head entity}, \textit{relation}, \textit{tail entity}),$$

also called facts, indicating that two entities are connected by a specific relation, e.g., ('Hitchcock', 'DirectorOf', 'Psycho').



Formal Knowledge graph

Given a KG of fact $\mathcal{S}$, a relation is denoted by a triplet $(h, r, t) \in \mathcal{S}$, where

- h is the **head** (a.k.a. left entity) of the relation;
- r is the kind (a.k.a. label) of **relation**, and
- t is the **tail** (a.k.a. right entity) of the relation.

We assume that $h, t \in \mathcal{E}$ where $\mathcal{E}$ is the set of all possible entities and $r \in \mathcal{R}$, where $\mathcal{R}$ is the set of all possible relations.

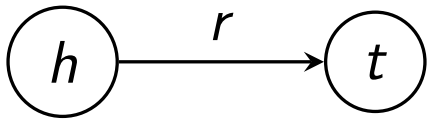


Figure: Graphical depiction of a relation r between the entities h and t, which is represented by the triplet (h, r, t).

Knowledge graph embedding

Although effective in representing structured data, the underlying symbolic nature of such triples usually makes KGs hard to manipulate.

$\Rightarrow$ Knowledge Graph Embedding (KGE) has shown of being a possible direction for tackling this issue.

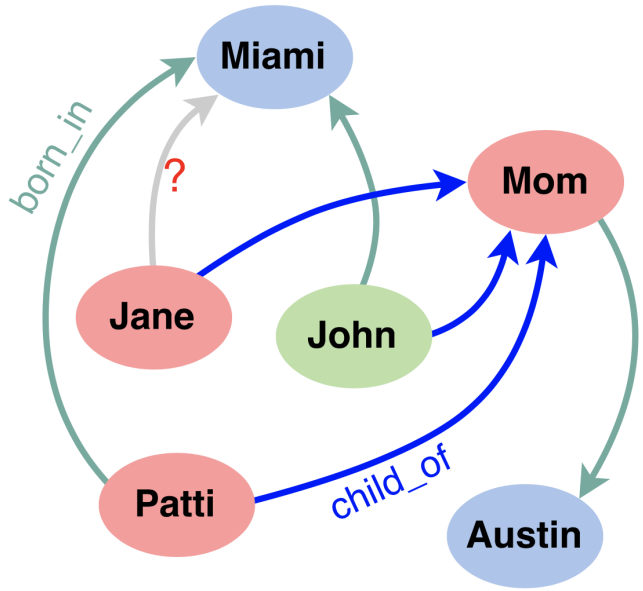
Goal of KGE

Embed components of a KG including entities and relations into continuous vector spaces, so as to simplify the manipulation while preserving the inherent structure of the KG.

Those embeddings can further be used to benefit all kinds of tasks such as Knowledge Graph Completion.

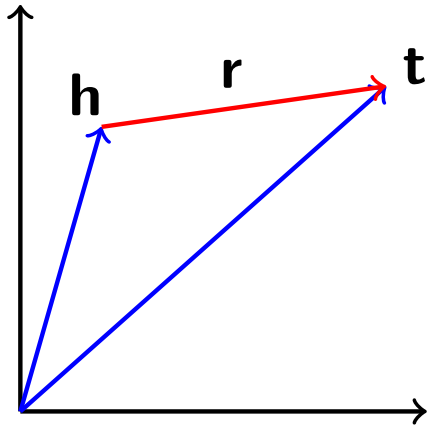
Knowledge graph completion

- KGs are typically incomplete, hence it is useful to be able to perform knowledge base completion (a.k.a. link prediction);
- Link prediction is concerned with predicting the existence (or probability of correctness) of (typed) edges in the graph (i.e., triples);
- This is important since existing **knowledge graphs are often missing many facts**, and some of the edges they contain are incorrect.



Trans-E

- The basic idea behind Trans-E is that a relation corresponds to a linear translation in the embedding space (similar to the "side-effect" of word2vec);
- The method assumes that entities can be mapped into a k-dimensional embedding space, i.e., $h, t \rightarrow \mathbf{h}, \mathbf{t} \in \mathbb{R}^k$ in which when the relation (h, r, t) holds then $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$, otherwise $\mathbf{t}$ should be far away from $\mathbf{h} + \mathbf{r}$.



Trans-E - Model

In order to learn these embeddings, Trans-E minimizes the following margin-based ranking criterion:

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{S}} \sum_{(h',r,t')\in\mathcal{S}'_{(h,r,t)}} [f_r(h,t) - f_r(h',t') + \gamma]_+,$$

subject to $\forall \mathbf{h}, \|\mathbf{h}\|_2 = 1, \quad \forall \mathbf{t}, \|\mathbf{t}\|_2 = 1$ where

- $[x]_+$ denotes the positive part of x;
- γ is a margin hyper-parameters;
- $\mathcal{S}'_{(h,r,t)}$ is the set of corrupted triplets

$$\mathcal{S}'_{(h,r,t)} \equiv \{(h',r,t) \mid h' \in \mathcal{E}\} \cup \{(h,r,t') \mid t' \in \mathcal{E}\}.$$

- The scoring function is then defined as the (negative) distance between $\mathbf{h} + \mathbf{r}$ and , i.e.,

$$f_r(h,t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_2.$$

Optimization and prediction with Trans-E

Optimization

- Random initialization of **h**, **r** and **t**;
- Optimization is carried out by SGD (minibatch mode);
- At each iteration:
  - embeddings of the entities are normalized;
  - for each example in the minibatch a corrupted triple is sampled.
- stop criterion based on validation performance.

Prediction

- For each test triple (h, r, t) the head h is removed;
- Trans-E scores each $(h', r, t) \mid h' \in \mathcal{E}$ and rank them accordingly.

Try this at home

- Try PCA on the IRIS dataset
- Autoencoders on the MNIST dataset
- CNNs on CIFAR dataset (Keras implementations)
- W2V on the WIKIPEDIA dataset
- TransE on WORDNET dataset