Artificial Neural Networks

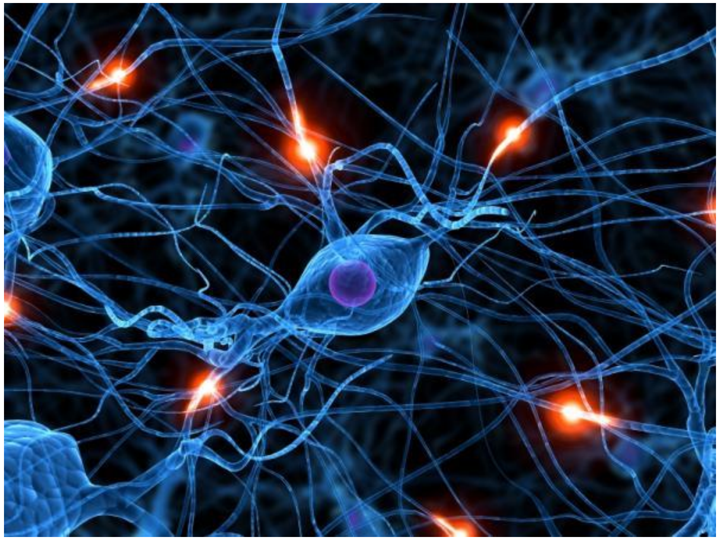
Data Base and Data Mining Group of Politecnico di Torino

Elena Baralis

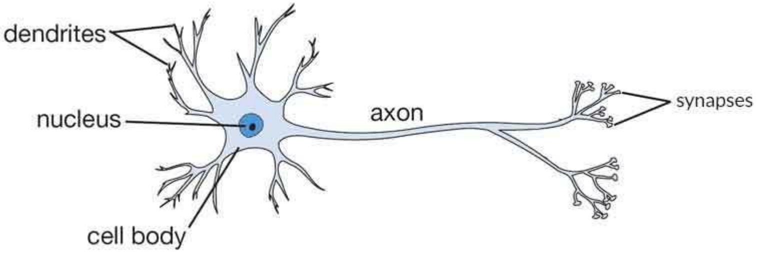
Politecnico di Torino

Artificial Neural Networks

- Inspired to the structure of the human brain
  - Neurons as elaboration units
  - Synapses as connection network



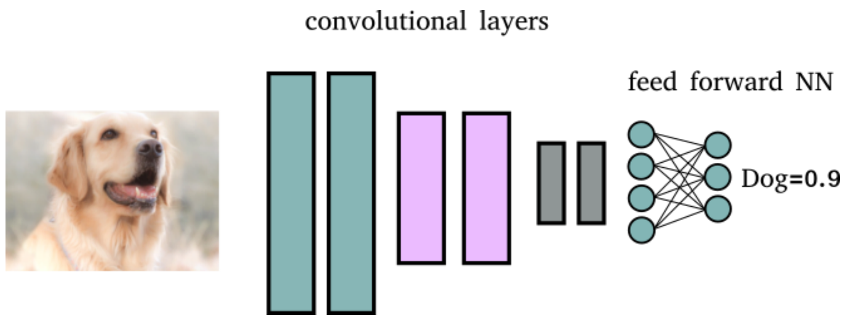
Biological Neuron



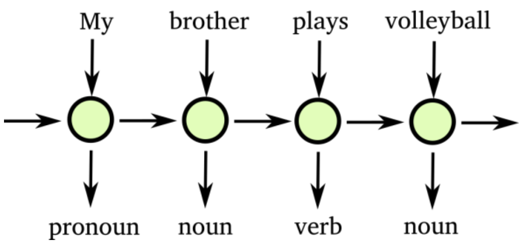
Artificial Neural Networks

- Different tasks, different architectures

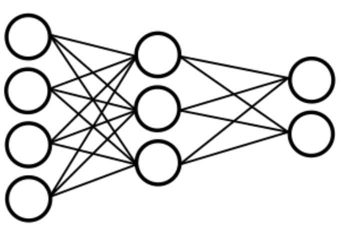
image understanding: convolutional NN (CNN)



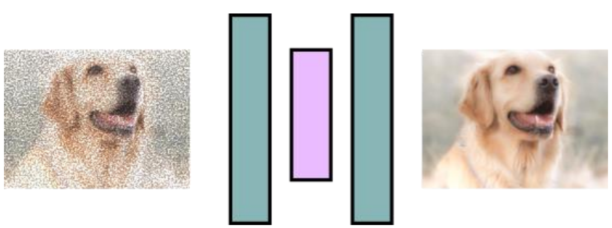
time series analysis: recurrent NN (RNN)



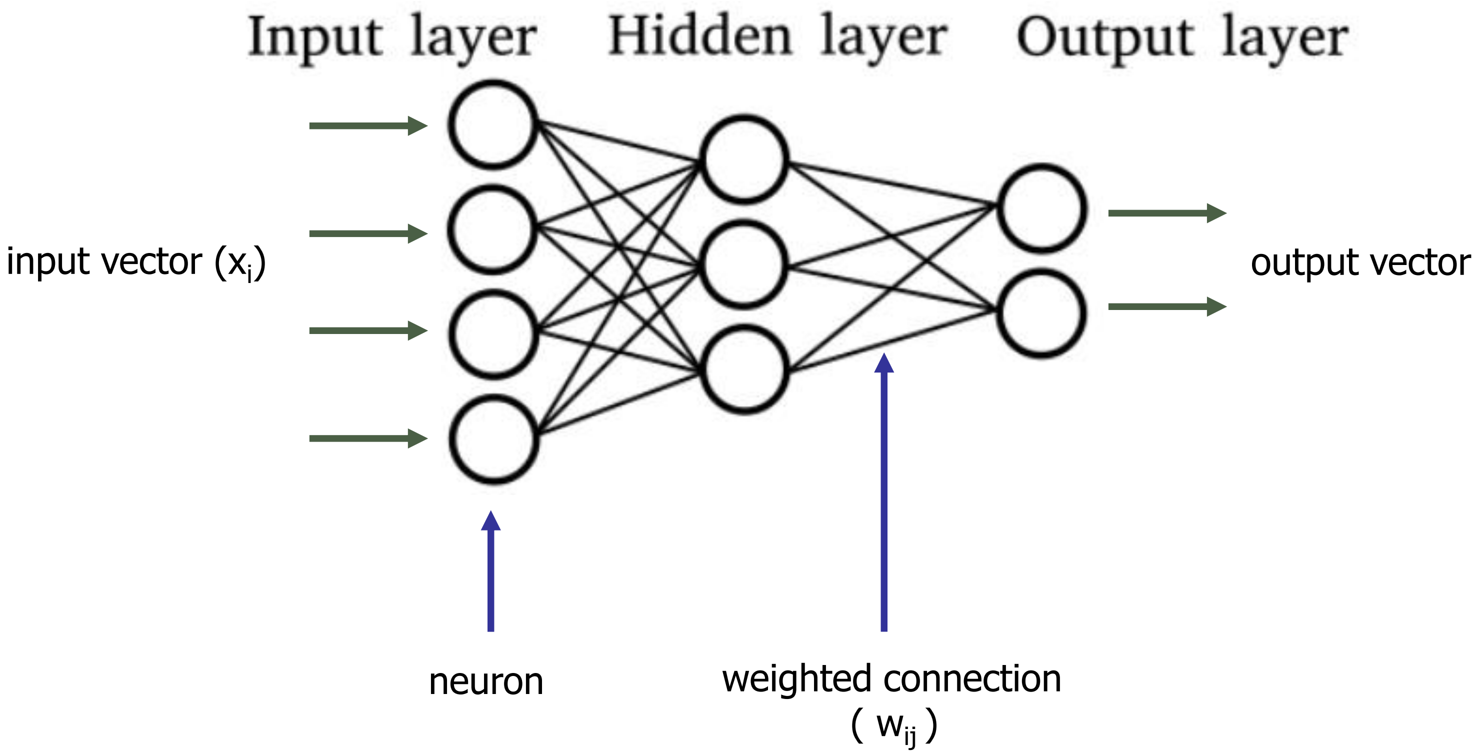
numerical vectors classification: feed forward NN (FFNN)



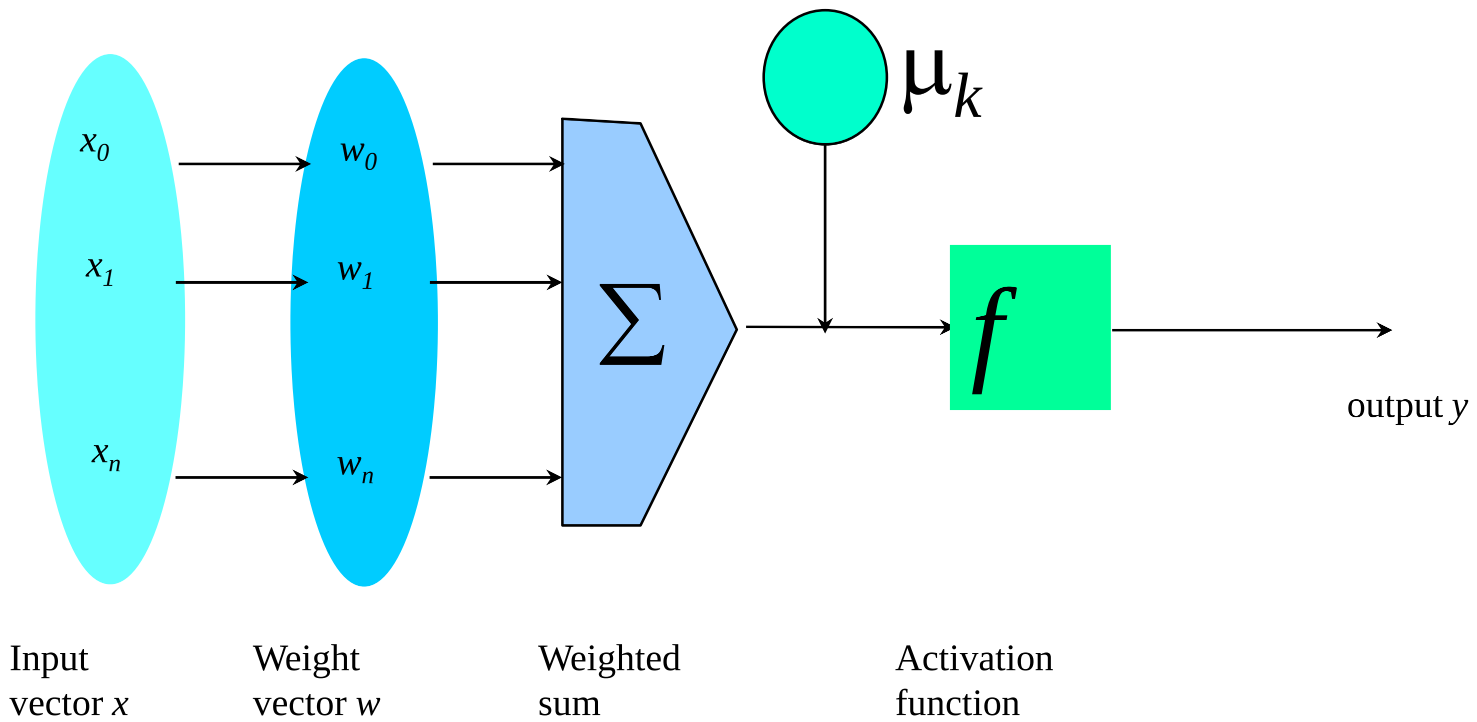
denoising: auto-encoders



Feed Forward Neural Network



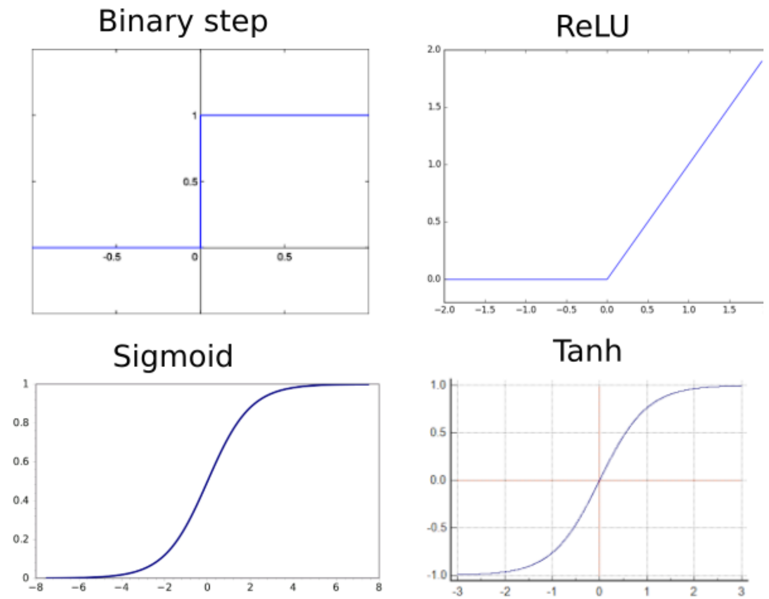
Structure of a neuron



From: Han, Kamber,"Data mining; Concepts and Techniques", Morgan Kaufmann 2006

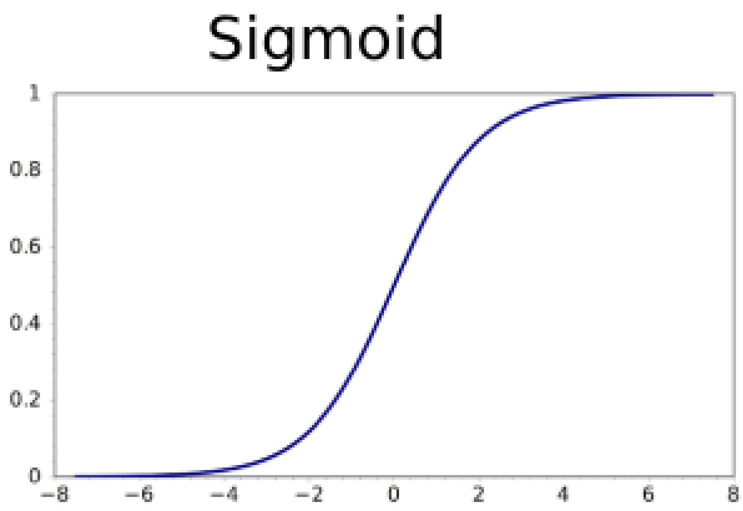
Activation Functions

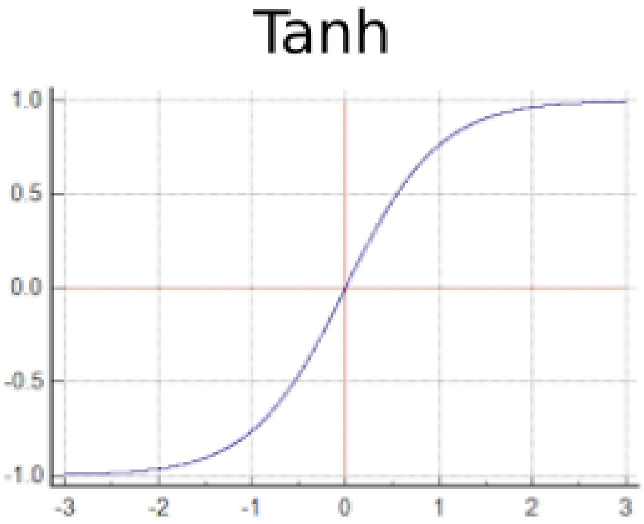
- Activation
  - simulates biological activation to input stymula
  - provides non-linearity to the computation
  - may help to saturate neuron outputs in fixed ranges



Activation Functions

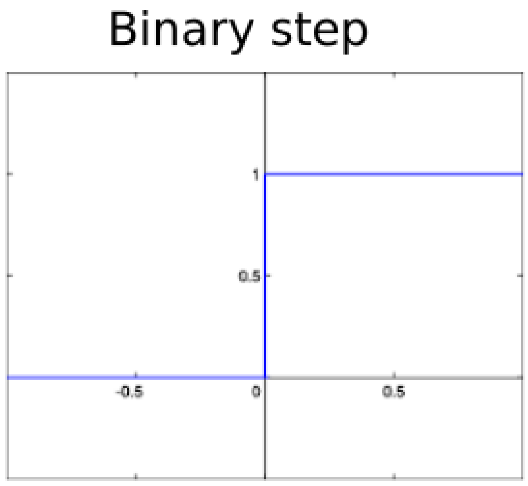
- Sigmoid, tanh
  - saturate input value in a fixed range
  - non linear for all the input scale
  - typically used by FFNNs for both hidden and output layers
    - E.g. *sigmoid* in output layers allows generating values between 0 and 1 (useful when output must be interpreted as likelihood)



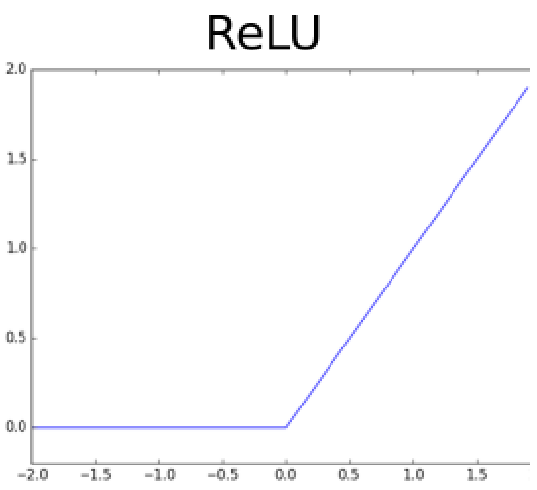


Activation Functions

- Binary Step
  - outputs 1 when input is non-zero
  - useful for binary outputs
  - **issues**: not appropriate for gradient descent
    - derivative not defined in x=0
    - derivative equal to 0 in every other position



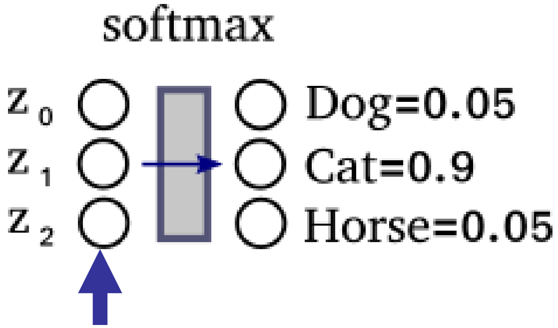
- ReLU (Rectified Linear Unit)
  - used in deep networks (e.g. CNNs)
    - avoids vanishing gradient
    - does not saturate
  - neurons activate linearly only for positive input



Activation Functions

- Softmax
  - differently to other activation functions
    - it is applied only to the **output** layer
    - works by considering **all the neurons** in the layer
  - after softmax, the output vector can be interpreted as a discrete **distribution of probabilities**
    - e.g. the probabilities for the input pattern of belonging to each class

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{i=0}^{N-1} e^{z_i}}$$



output layer

Building a FFNN

- For each node, definition of
  - set of weights
  - offset value

  providing the highest accuracy on the training data
- Iterative approach on training data instances

Building a FFNN

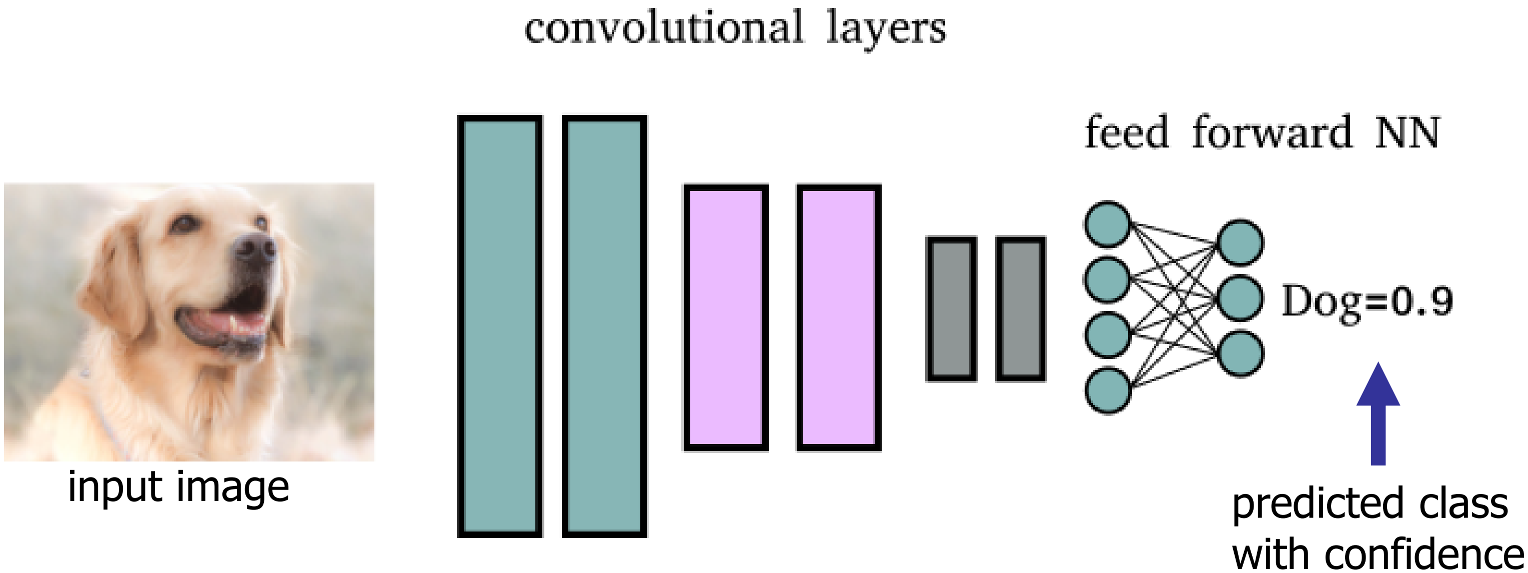
- Base algorithm
  - Initially assign random values to weights and offsets
  - Process instances in the training set one at a time
    - For each neuron, compute the result when applying weights, offset and activation function for the instance
    - Forward propagation until the output is computed
    - Compare the computed output with the expected output, and evaluate error
    - Backpropagation of the error, by updating weights and offset for each neuron
  - The process ends when
    - % of accuracy above a given threshold
    - % of parameter variation (error) below a given threshold
    - The maximum number of epochs is reached

Feed Forward Neural Networks

- Strong points
  - High accuracy
  - Robust to noise and outliers
  - Supports both discrete and continuous output
  - Efficient during classification
- Weak points
  - Long training time
    - weakly scalable in training data size
    - complex configuration
  - Not interpretable model
    - application domain knowledge cannot be exploited in the model

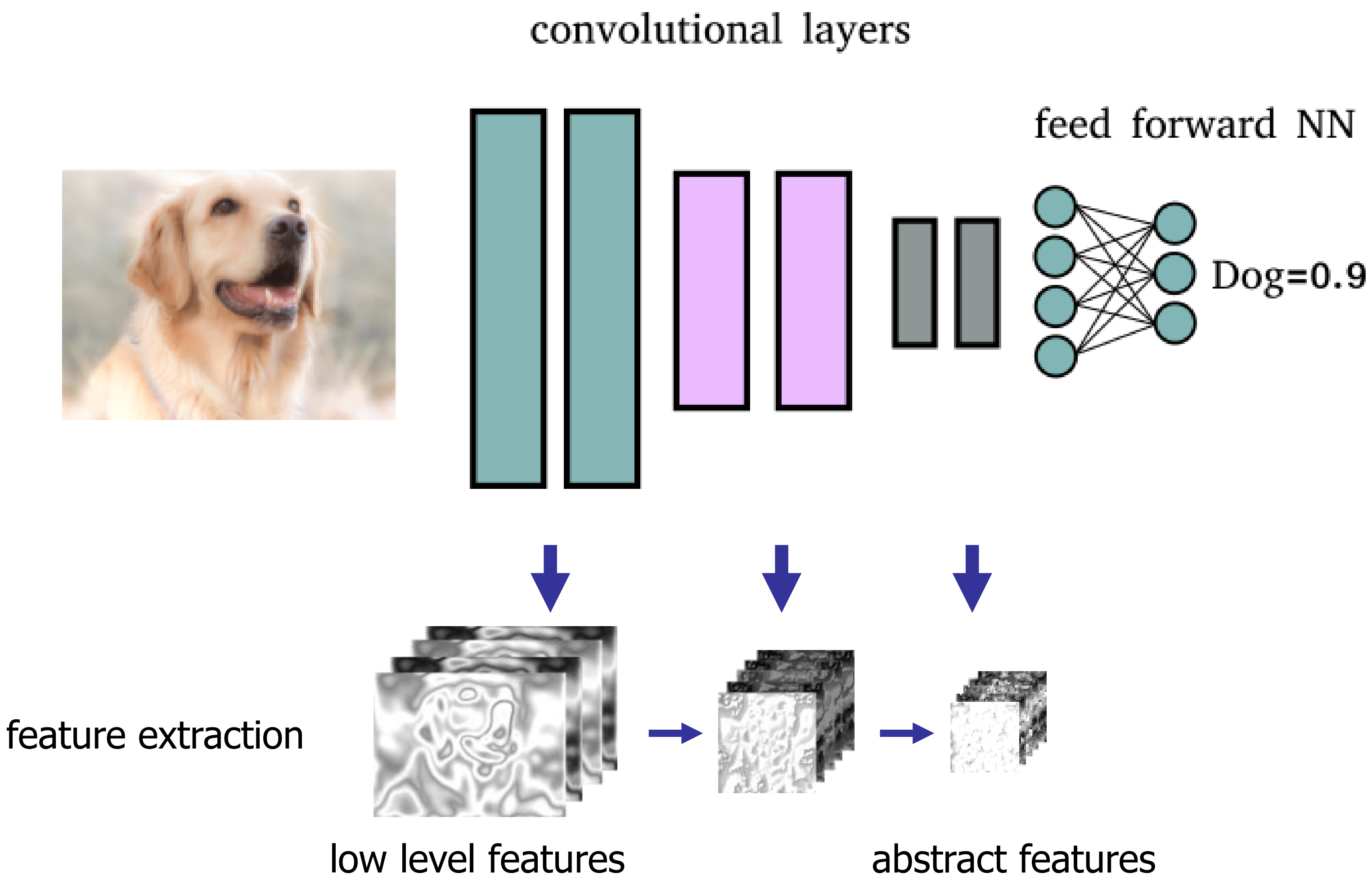
Convolutional Neural Networks

- Allow automatically extracting **features** from images and performing **classification**

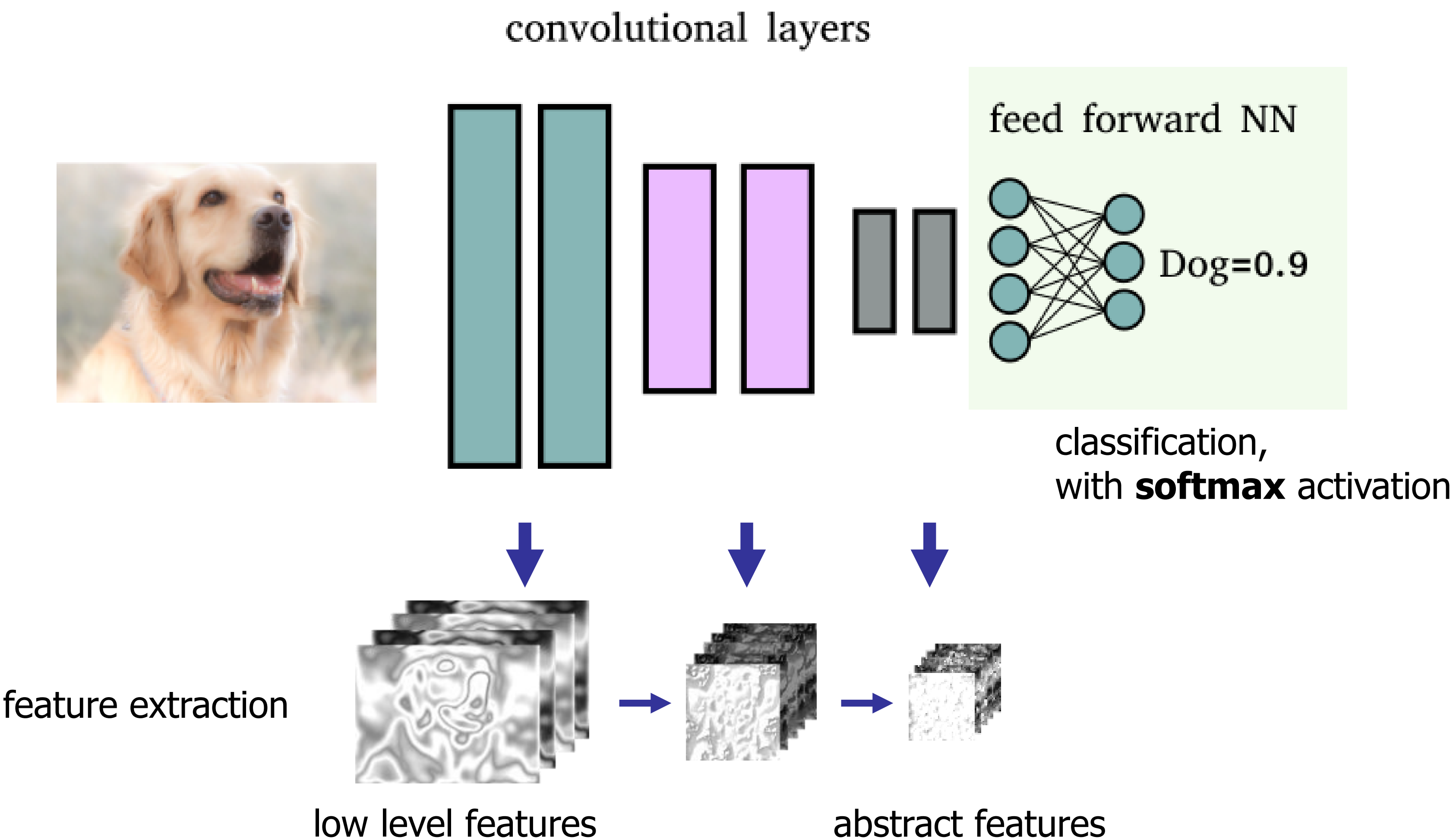


Convolutional Neural Network (CNN) Architecture

Convolutional Neural Networks

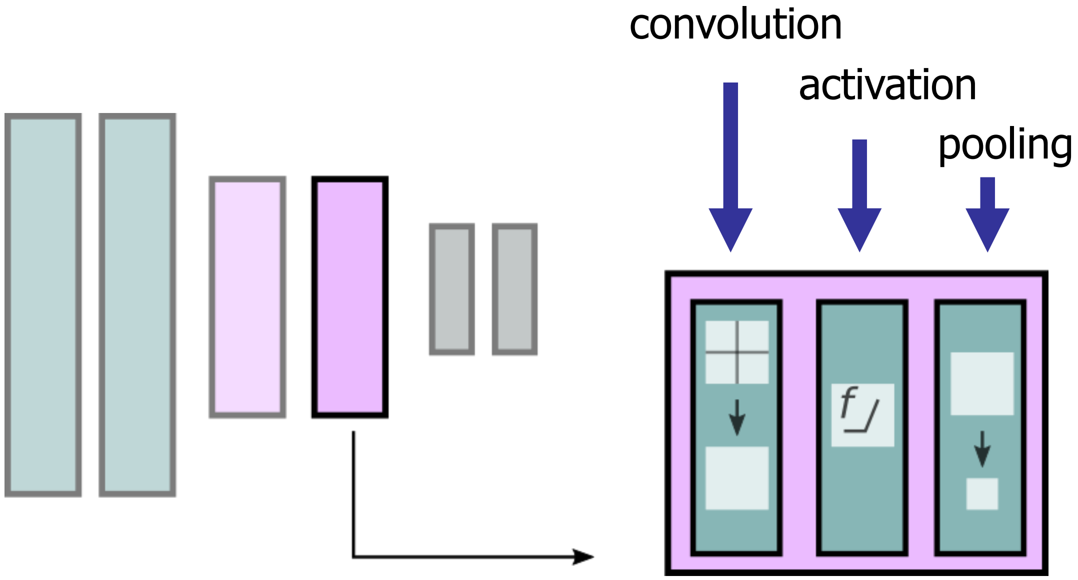


Convolutional Neural Networks



Convolutional Neural Networks

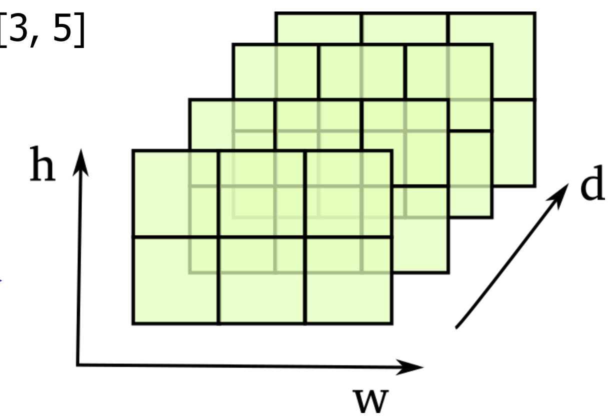
- Typical convolutional layer
  - *convolution* stage: feature extraction by means of (hundreds to thousands) sliding filters
  - sliding filters *activation*: apply activation functions to input tensor
  - *pooling*: tensor downsampling



Convolutional Neural Networks

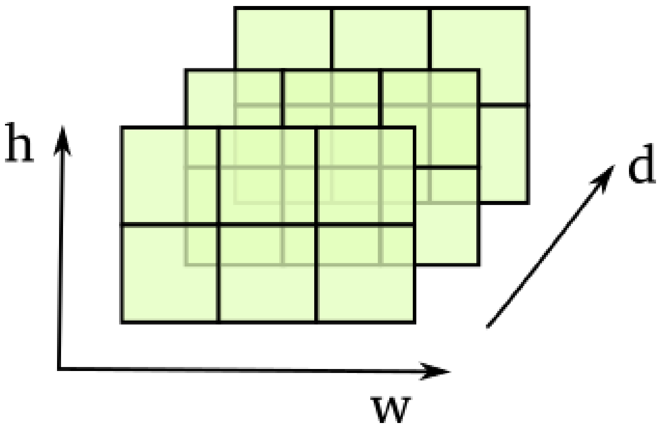
- Tensors
  - data flowing through CNN layers is represented in the form of *tensors*
  - *Tensor* = N-dimensional vector
  - *Rank* = number of dimensions
    - scalar: rank 0
    - 1-D vector: rank 1
    - 2-D matrix: rank 2
  - *Shape* = number of elements for each dimension
    - e.g. a vector of length 5 has shape [5]
    - e.g. a matrix w x h, w=5, h=3 has shape [h, w] = [3, 5]

rank-3 tensor with shape
[d,h,w] = [4,2,3]



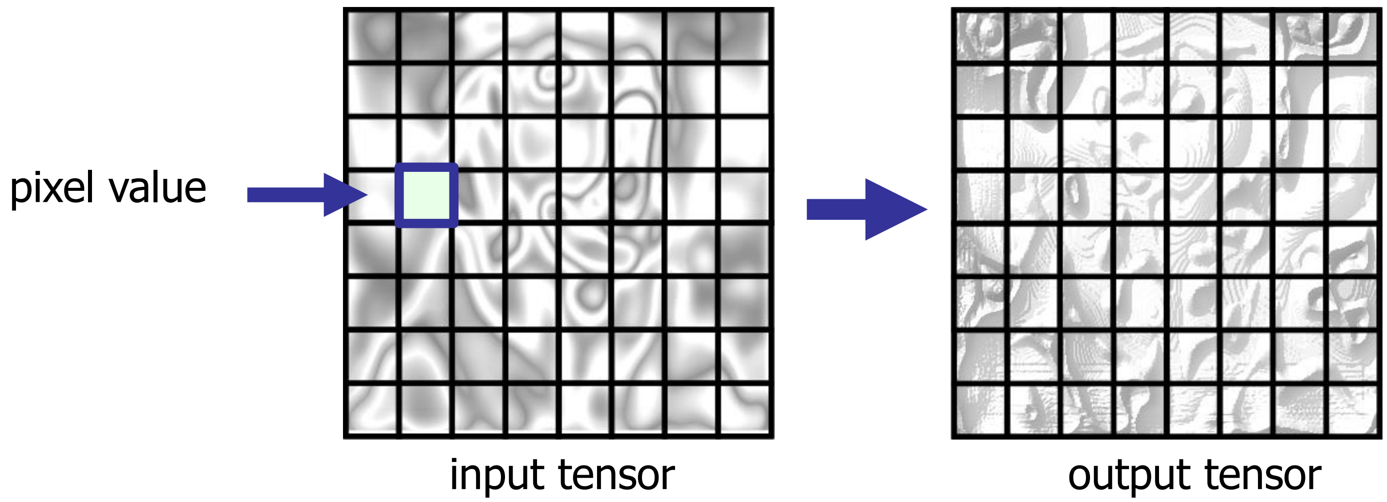
Convolutional Neural Networks

- Images
  - rank-3 tensors with shape [d,h,w]
  - where h=height, w=width, d=image depth (1 for grayscale, 3 for RGB colors)



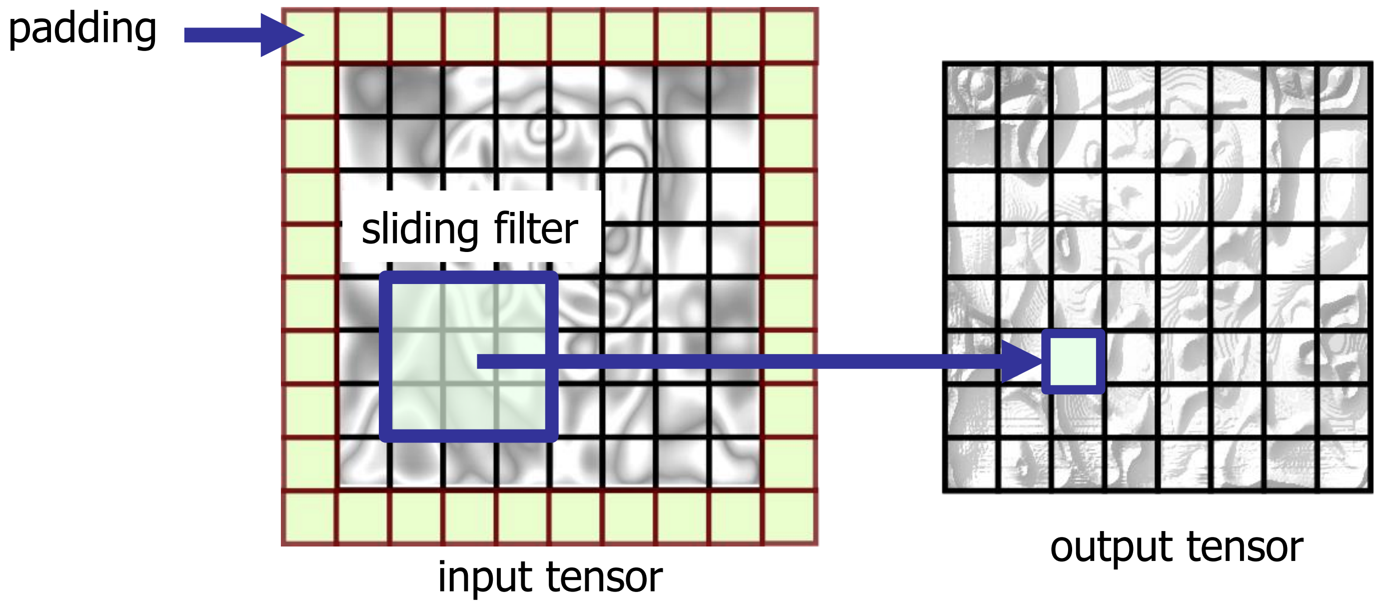
Convolutional Neural Networks

- Convolution
  - processes data in form of *tensors* (multi-dimensional matrixes)
  - **input**: input image or intermediate features (tensor)
  - **output**: a tensor with the extracted features

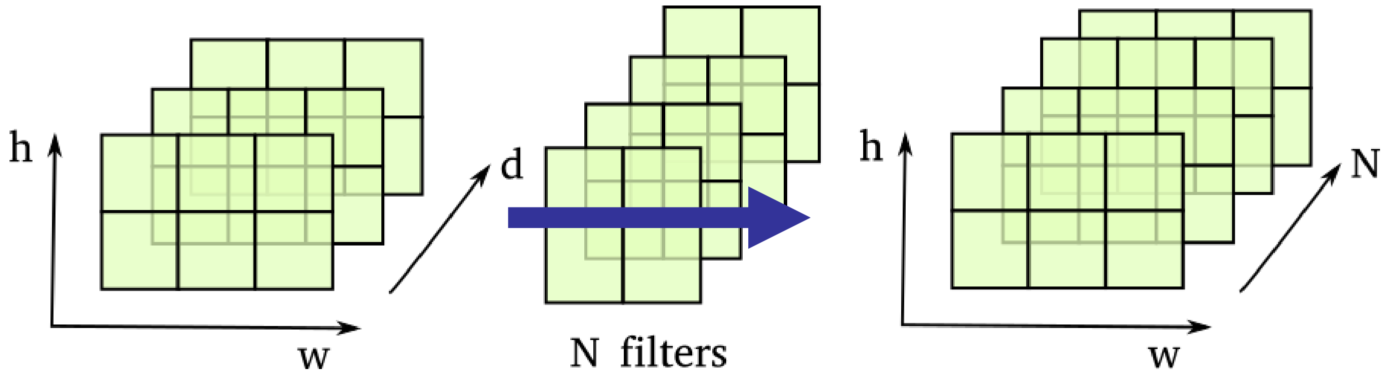


Convolutional Neural Networks

- Convolution
  - a *sliding filter* produces the values of the output tensor
  - sliding filters contain the trainable *weights* of the neural network
  - each convolutional layer contains many (hundreds) filters

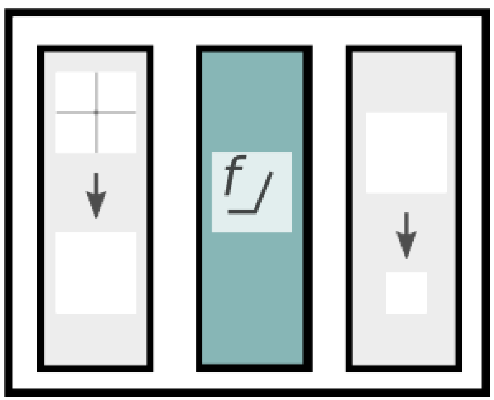


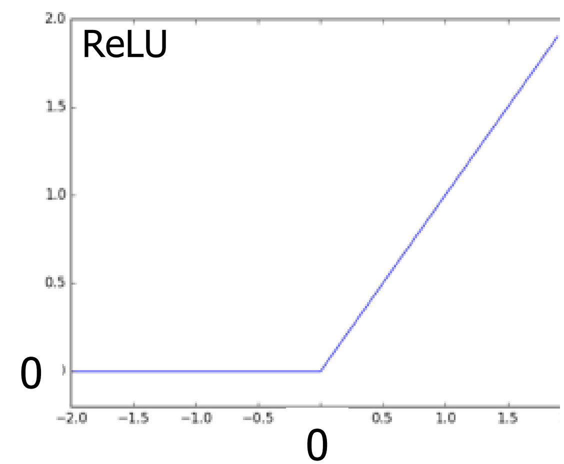
- Convolution
  - images are transformed into features by convolutional filters
  - after convolving a tensor [d,h,w] with *N filters* we obtain
    - a rank-3 tensor with shape [N,h,w]
    - hence, each filter generates a layer in the depth of the output tensor



Convolutional Neural Networks

- Activation
  - symulates biological activation to input stymula
  - provides non-linearity to the computation
  - ReLU is typically used for CNNs
    - faster training (no vanishing gradients)
    - does not saturate
    - faster computation of derivatives for backpropagation





- Pooling
  - performs tensor *downsampling*
  - *sliding filter* which replaces tensor values with a *summary* statistic of the nearby outputs
  - *maxpool* is the most common: computes the maximum value as statistic

sliding filter

0	1	0	2
0	2	0	3
1	0	0	2
2	0	0	4

max=2

0	1	0	2
0	2	0	3
1	0	0	2
2	0	0	4

max=3

0	1	0	2
0	2	0	3
1	0	0	2
2	0	0	4

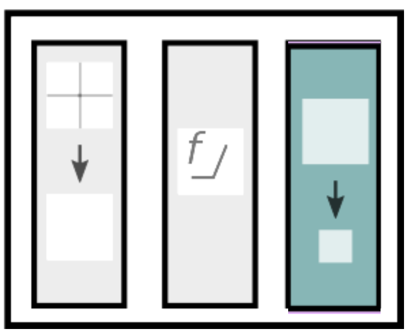
max=2

0	1	0	2
0	2	0	3
1	0	0	2
2	0	0	4

max=4

2	3
2	4

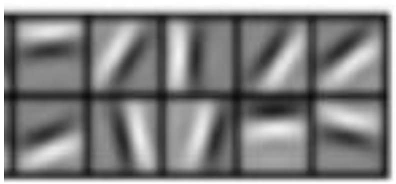
output tensor



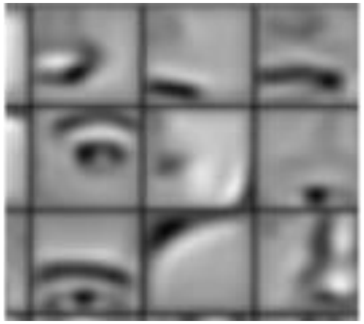
Convolutional Neural Networks

- Convolutional layers training
  - during training each sliding filter learns to recognize a particular *pattern* in the input tensor
  - filters in *shallow layers* recognize textures and edges
  - filters in *deeper layers* can recognize objects and parts (e.g. eye, ear or even faces)

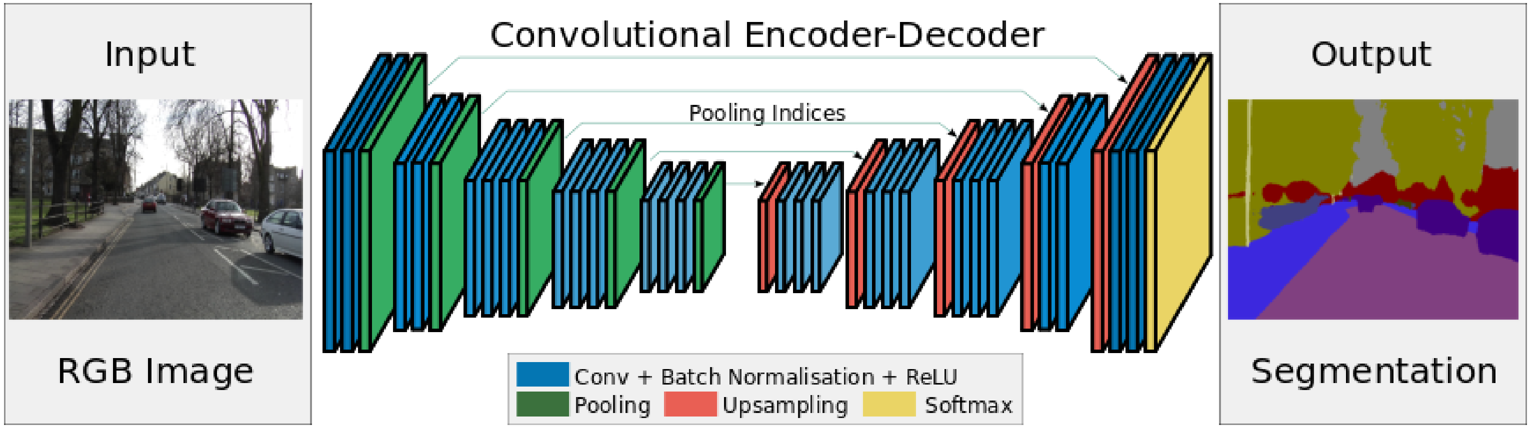
shallow filters



deeper filters



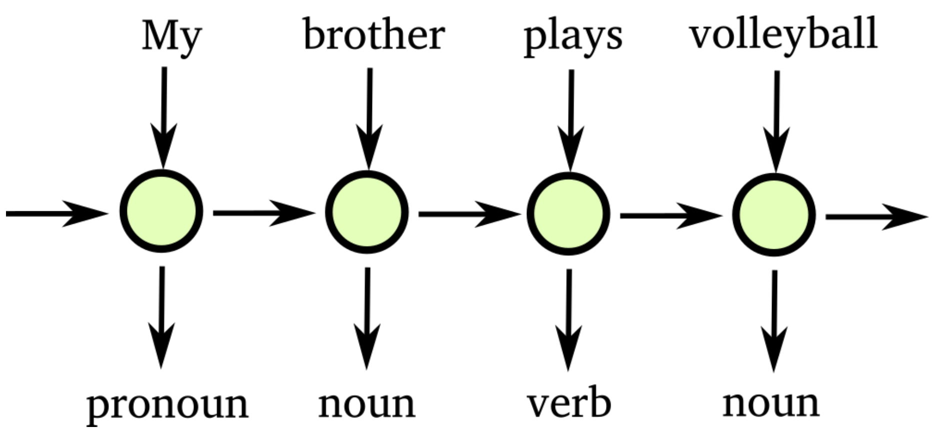
- Semantic segmentation CNNs
  - allow assigning a class to each pixel of the input image
  - composed of 2 parts
    - **encoder network**: convolutional layers to extract abstract features
    - **decoder network**: deconvolutional layers to obtain the output image from the extracted features



SegNet neural network

Recurrent Neural Networks

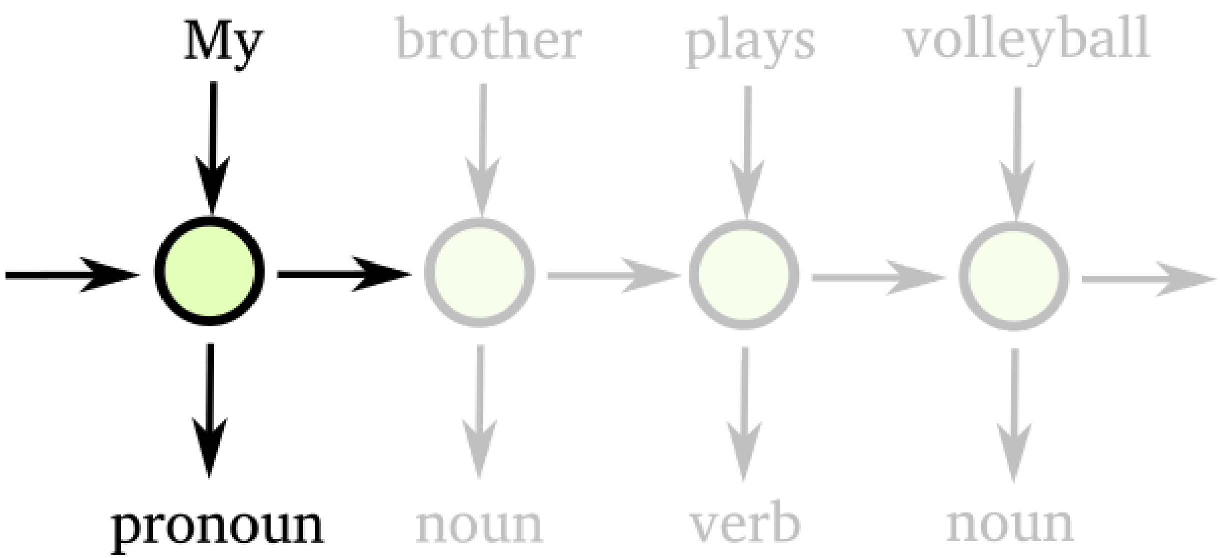
- Allow processing *sequential* data x(t)
- Differently from normal FFNN they are able to keep a *state* which evolves during time
- Applications
  - machine translation
  - time series prediction
  - speech recognition
  - part of speech (POS) tagging



Recurrent Neural Networks

- RNN execution during time

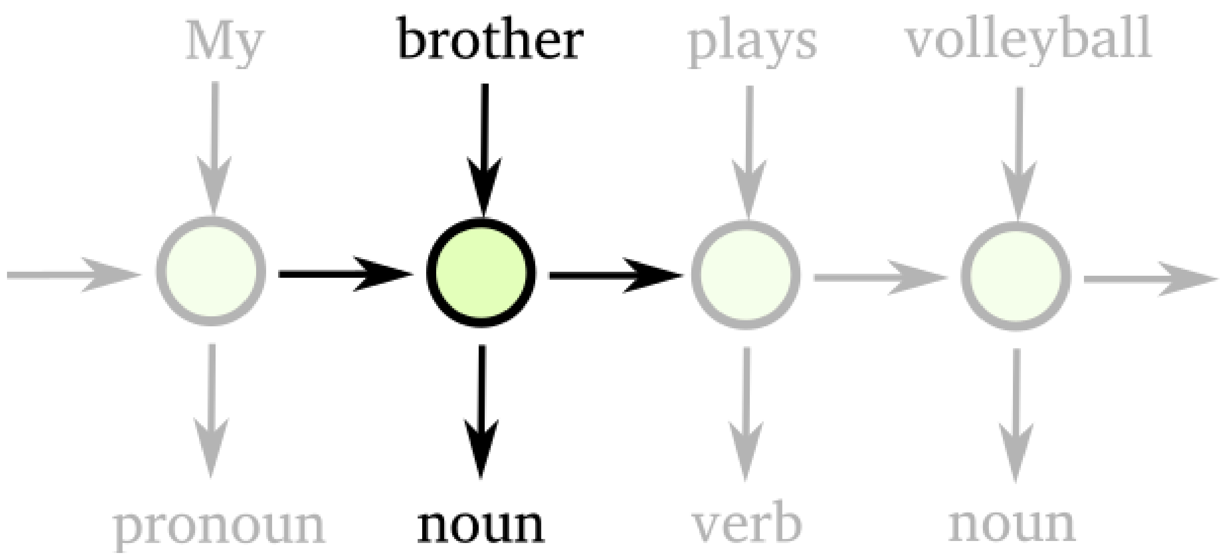
instance of the RNN at time t1



Recurrent Neural Networks

- RNN execution during time

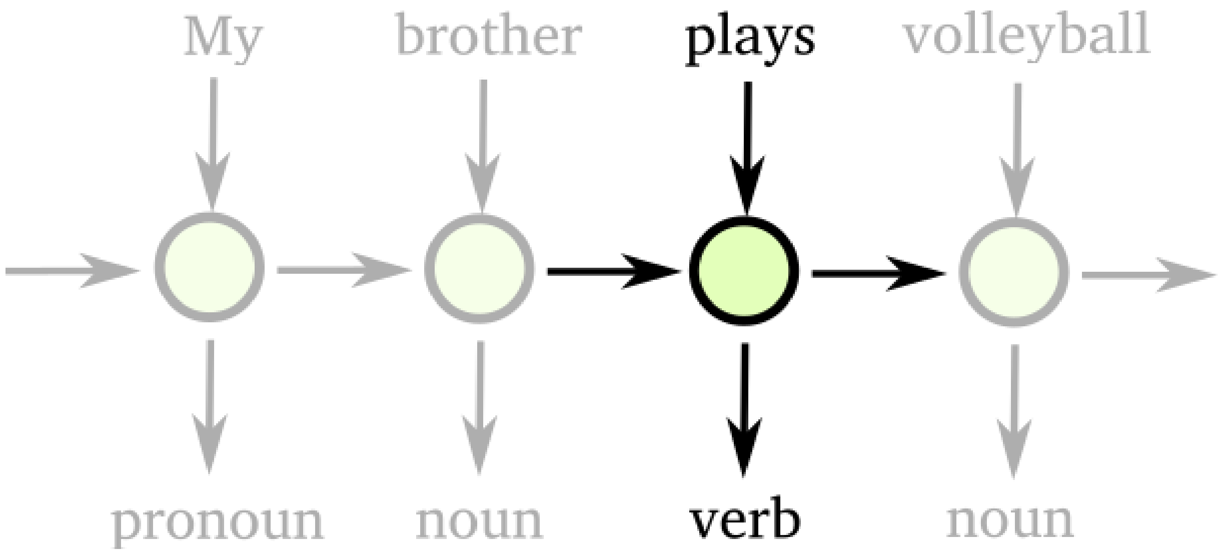
instance of the RNN at time t2



Recurrent Neural Networks

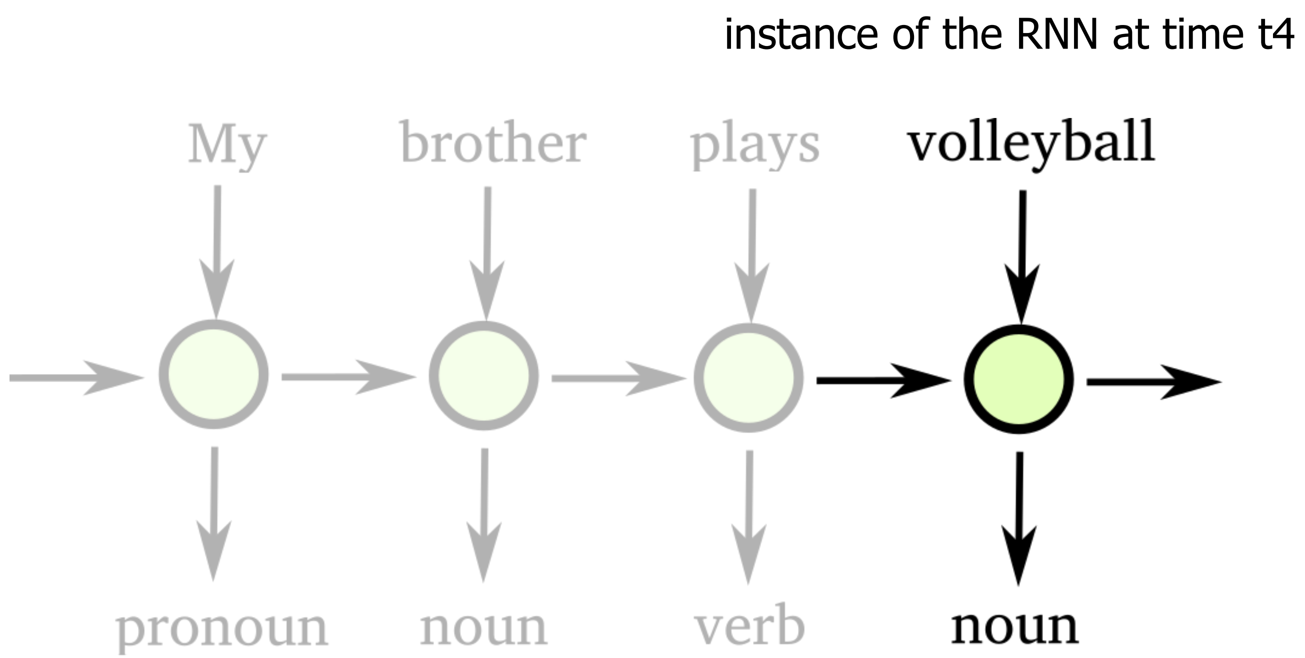
- RNN execution during time

instance of the RNN at time t3



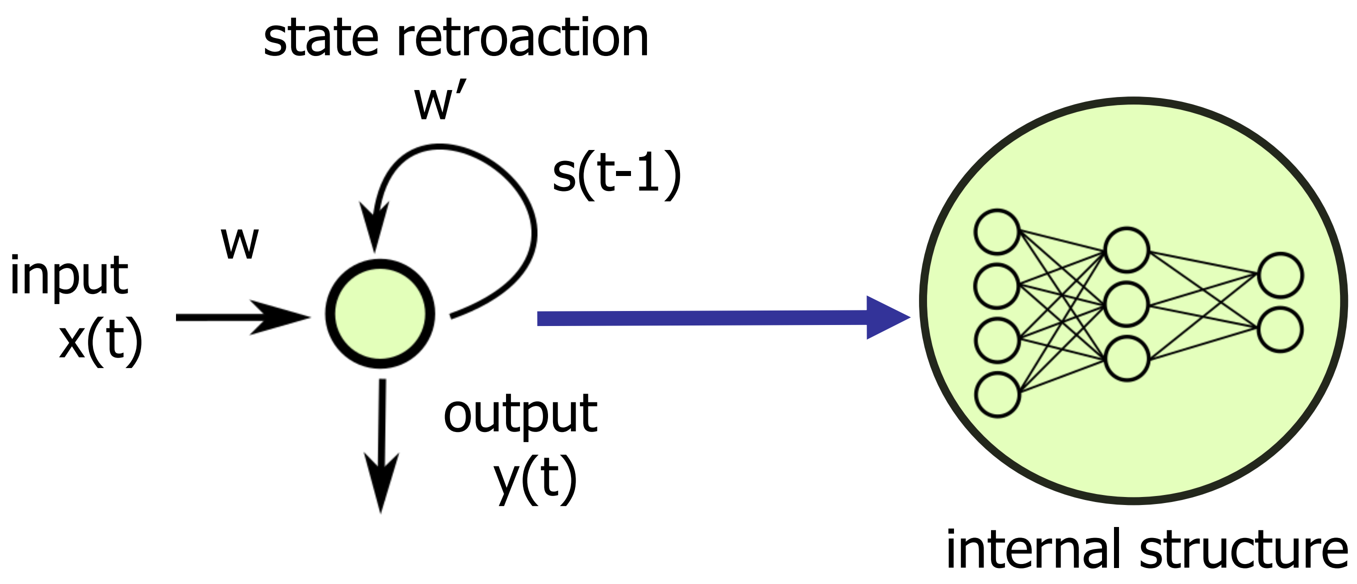
Recurrent Neural Networks

- RNN execution during time



Recurrent Neural Networks

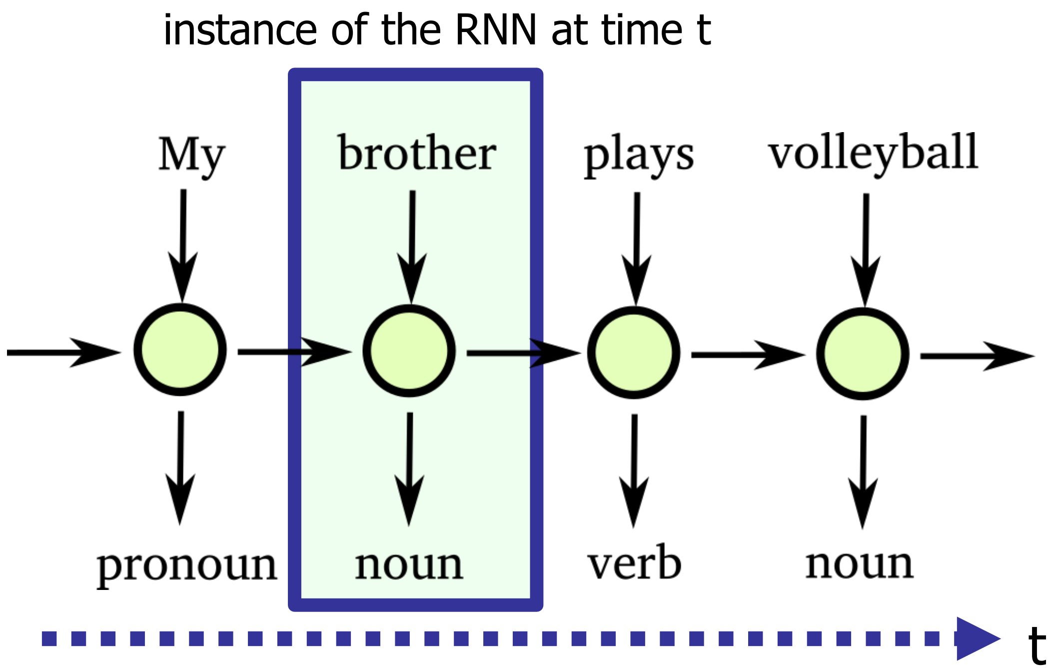
- A RNN receives as input a vector x(t) and the state at previous time step s(t-1)
- A RNN typically contains many *neurons organized in different layers*



Recurrent Neural Networks

- Training is performed with *Backpropagation Through Time*
- Given a pair training sequence x(t) and expected output y(t)
  - error is propagated through time
  - weights are updated to minimize the error across all the time steps

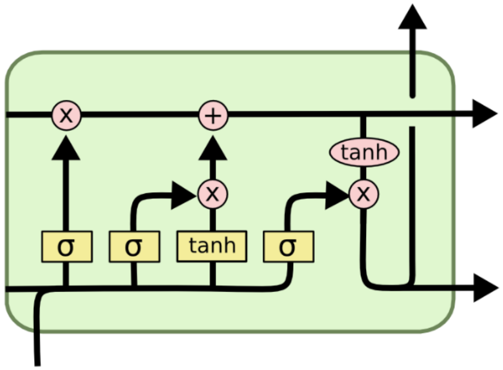
unrolled RNN diagram: shows the **same** neural network at different time steps



Recurrent Neural Networks

- Issues
  - *vanishing gradient*: error gradient decreases rapidly over time, weights are not properly updated
  - this makes harder having RNN with *long-term* memories

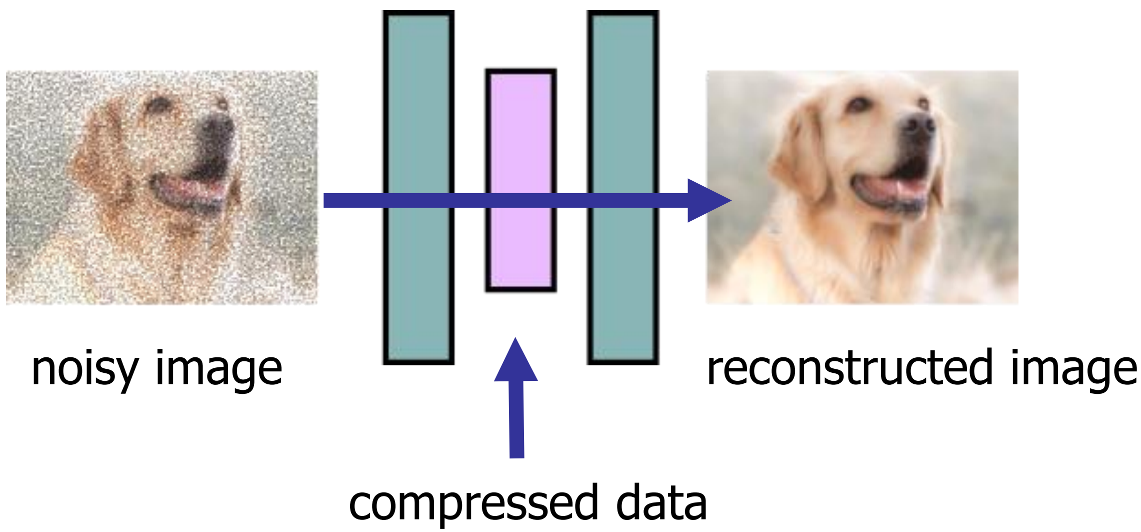
- Solution: *LSTM* (Long Short Term Memories)
  - RNN with "gates" which encourage the state information to flow through long time intervals



LSTM stage

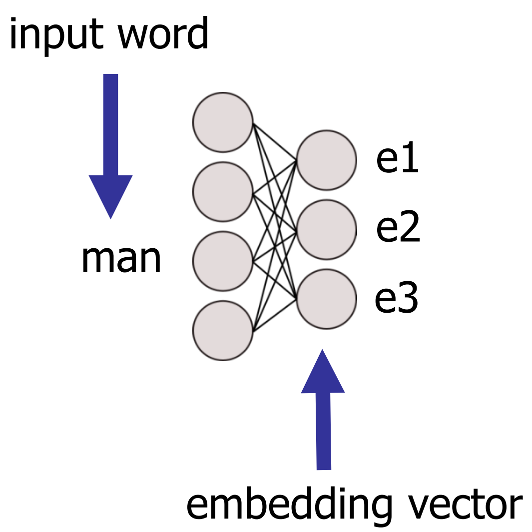
Autoencoders

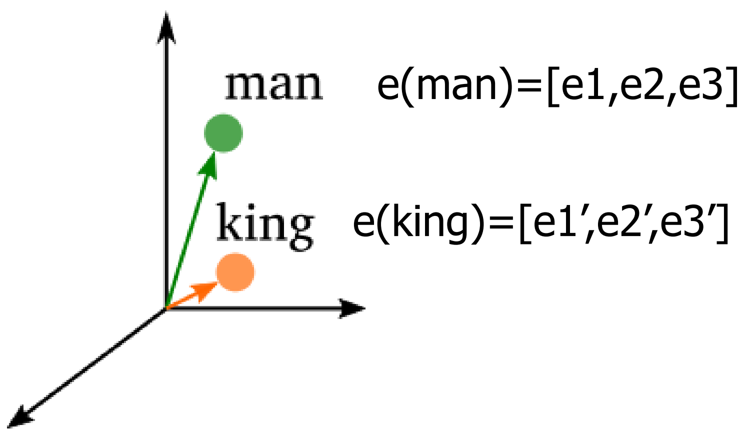
- Autoencoders allow *compressing* input data by means of compact representations and from them *reconstruct* the initial input
  - for feature extraction: the compressed representation can be used as significant set of features representing input data
  - for image (or signal) *denoising*: the image reconstructed from the abstract representation is denoised with respect to the original one



Word Embeddings (Word2Vec)

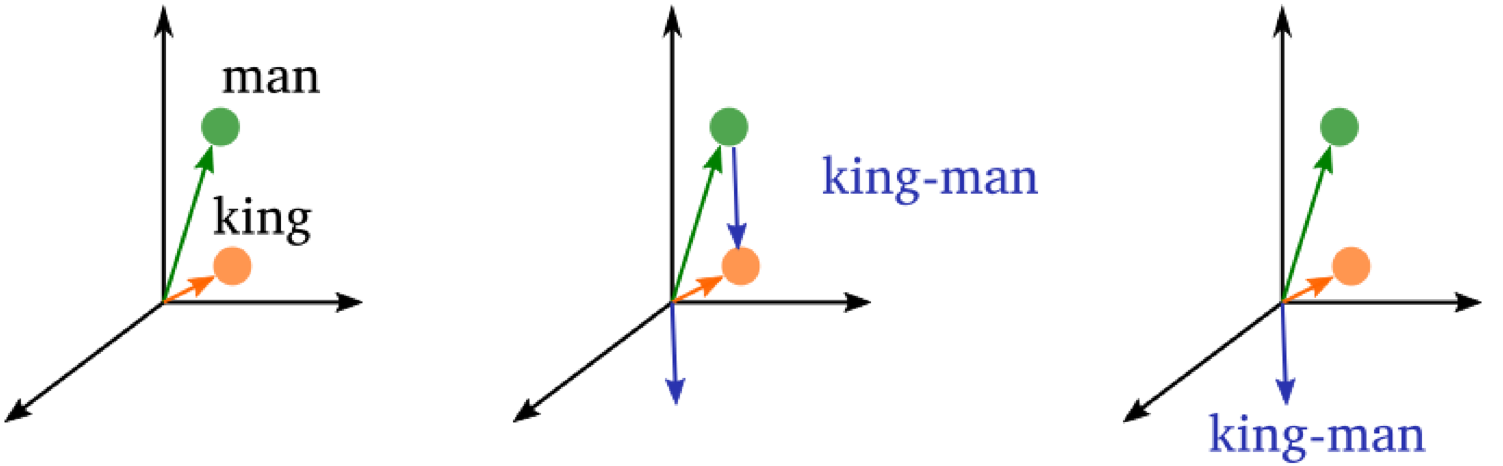
- Word *embeddings* associate words to n-dimensional vectors
  - trained on big text collections to model the word distributions in different sentences and contexts
  - able to capture the *semantic* information of each word
  - words with similar *meaning* share vectors with similar characteristics





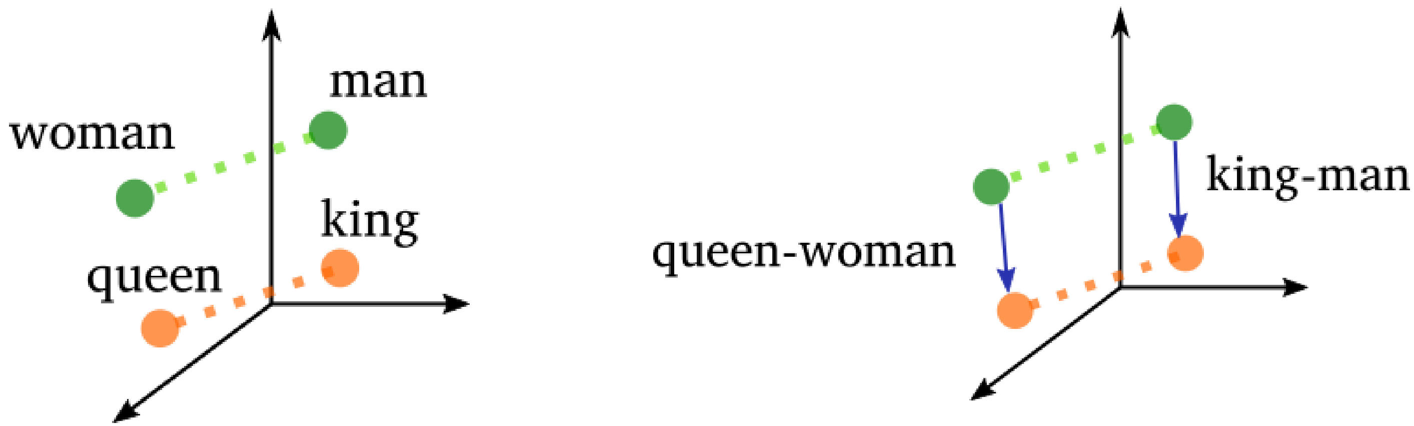
Word Embeddings (Word2Vec)

- Since each word is represented with a vector, operations among words (e.g. difference, addition) are allowed



Word Embeddings (Word2Vec)

- Semantic relationships among words are captured by vector positions



king - man = queen - woman
king - man + woman = queen