neural networks: advanced models

Patrick van der Smagt

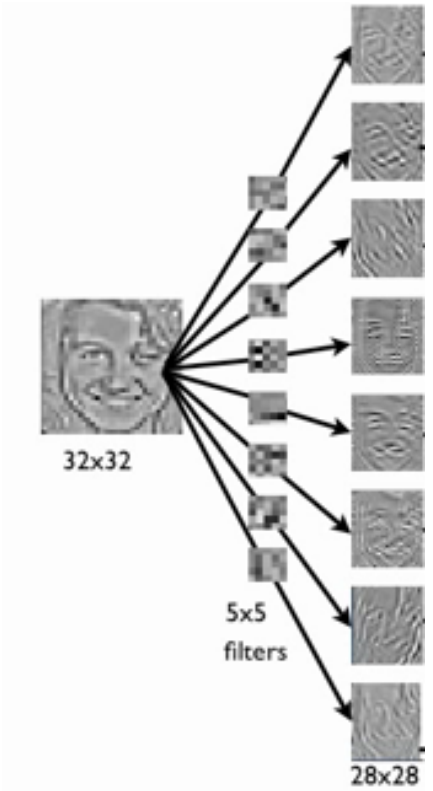
image processing with neural networks

Since 2009, convolutional neural networks (cNN aka ConvNet) have beaten many computer vision benchmarks. cNNs were somehow modelled after an abstract model of the human visual cortex, but have since progressed away from that.

A typical cNN consists of stacked convolutional and maxpooling layers.

- a convolutional layer consists of a number of $n \times m$ filters which are convolved on the image. For instance, a 5×5 filter is matched on all positions of the image, and the result of this match is the next image. This is done for many different filters (see figure);
- a maxpooling layer downsamples the resulting images, typically in 2×2 windows.

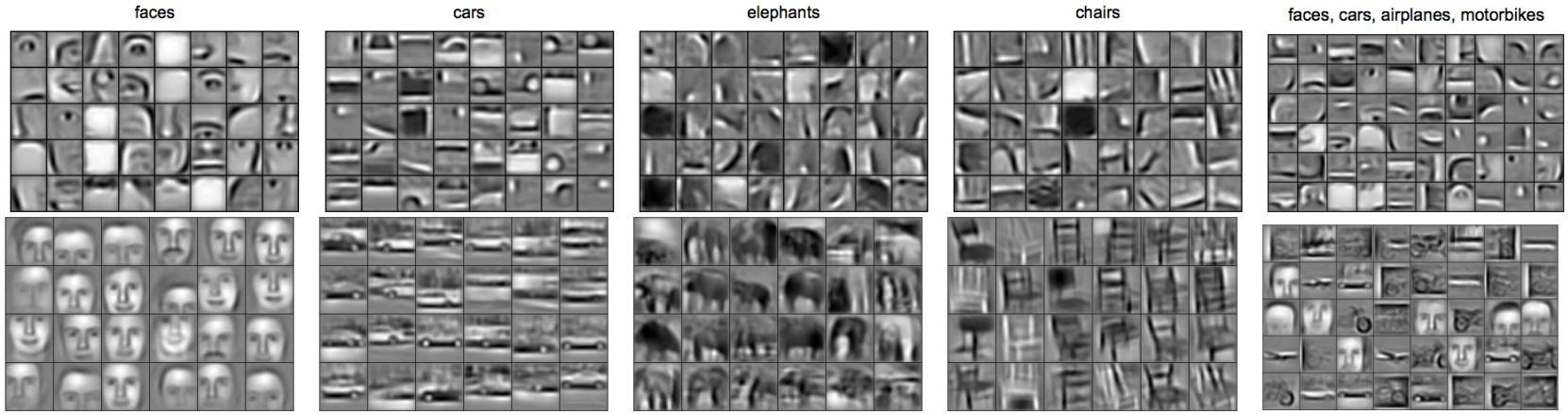
This structure is stacked, typically 2 to 4 times.



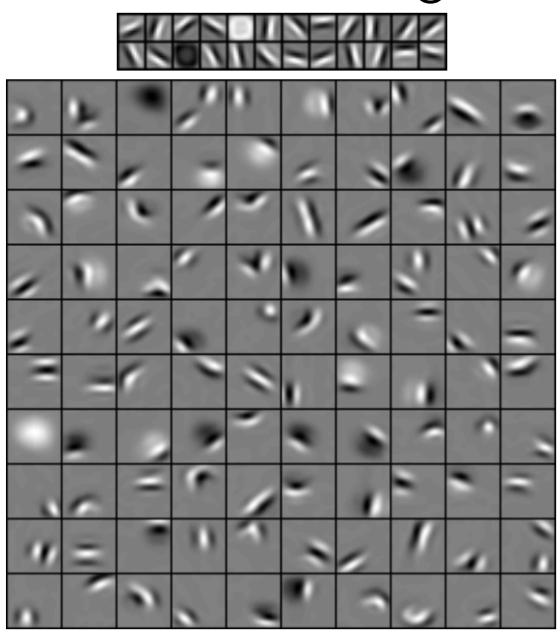
The power of a cNN is that the convolution filters are *learned*.

convolutional deep learning example (Ng 2009)

first layer (top) and second layer (bottom) weights for selected images



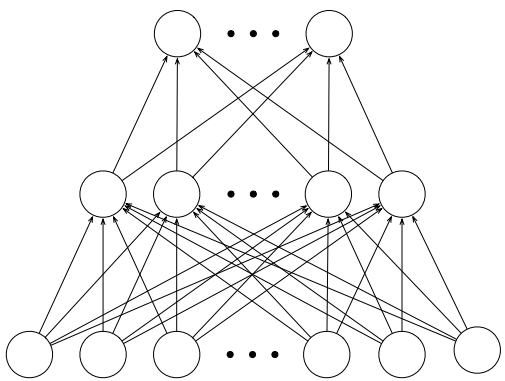
for natural images



unsupervised learning with neural networks

problem: often we have unlabelled data

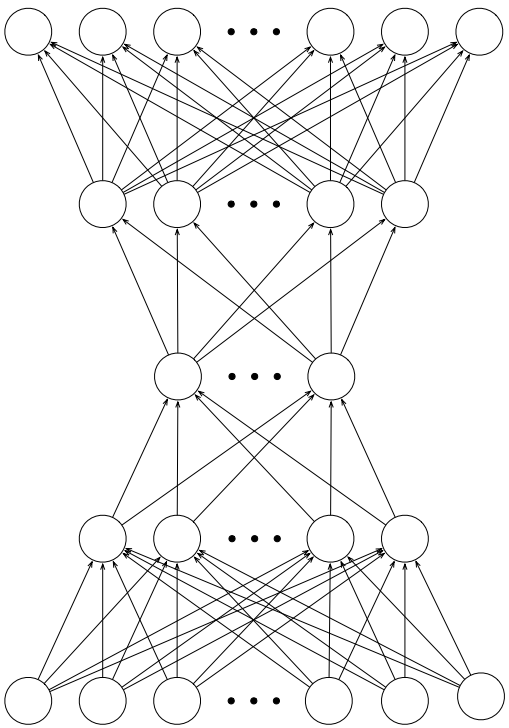
1. we have many data $\boldsymbol{x}$...
2. but no related output $\boldsymbol{z}$.



auto-encoder networks

idea: find compact representation of inputs (unsupervised!) by

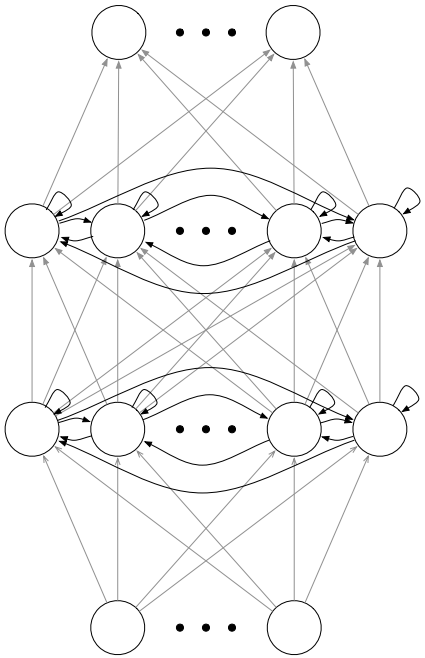
1. stacking a second neural network on top of the first
2. letting the whole NN compute $\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{x}$.



- middle layer ("latents") usually has fewer neurons
- latent representation $\boldsymbol{z}$ = compact code for $\boldsymbol{x}$

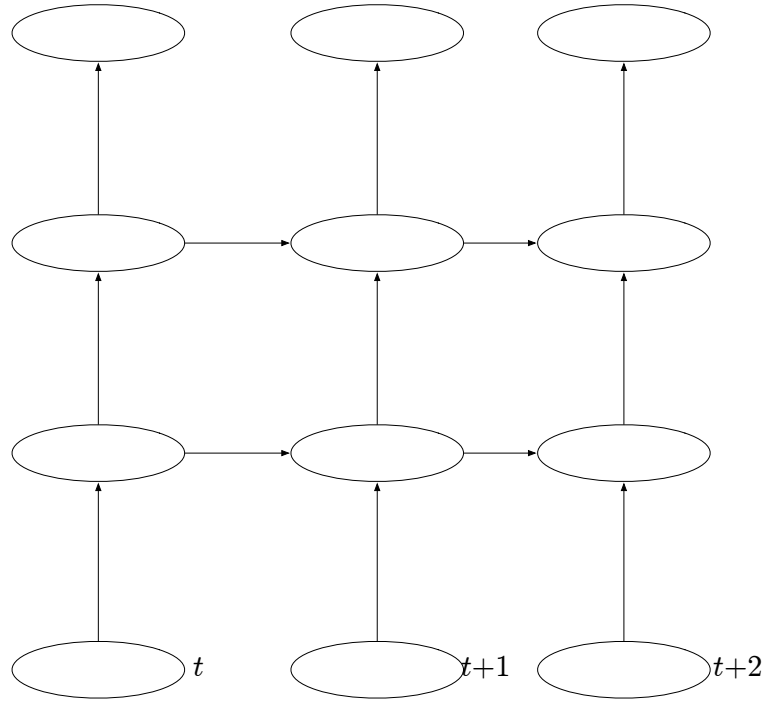
These networks make a compact representation of data ("dimensionality reduction"). However, you cannot control the representation!

recurrent neural networks



RNN

=



rolled out in time

Can be trained with backpropagation-through-time:

- input a finite sequence $\{\boldsymbol{x}(t)\}$, $t = 1, 2, \ldots T$ step by step
- back-propagate the error $y(\boldsymbol{x}(t)) - \boldsymbol{z}(t)$ timestep by timestep, starting at T down to 1, and sum the residuals;
- adapt the weights using the residuals computed above.

recurrent neural networks

RNNs have been successfully used for a number of serious applications.

These include

- robotics;
- handwritten character recognition and generation;
- text recognition, generation, annotation, etc.; speech recognition;
- machine translation

and so on.

A nice list of related resources can be found at
http://jiwonkim.org/awesome-rnn.

probabilistic neural networks

In recent years probabilistic neural networks have emerged.

In these, each neuron represents a random variable rather than a value.

The probabilistic neural network introduce many possibilities:

- prediction of confidence intervals at the output;
- probabilistic version of dropout;
- towards a full probabilistic interpretation;
- ...

getting started

Programming your own NN in Python is quite simple, but not efficient.

For efficient code, use one of many public domain libraries, e.g.

- Theano (from Université de Montreal) for Python
- TensorFlow (from Google) for Python and C++
- Chainer (not for Windows?) for Python
- ...

You will then find a number of implementations based on such libraries.

Our own library `breze` is a good example.

For cNNs, Berkeley's Caffe is the most widely used library.