neural networks: advanced models

Patrick van der Smagt

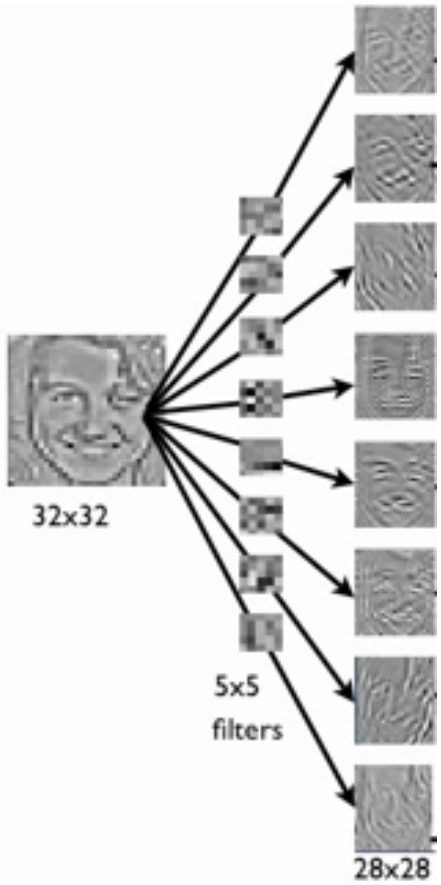
image processing with neural networks

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.

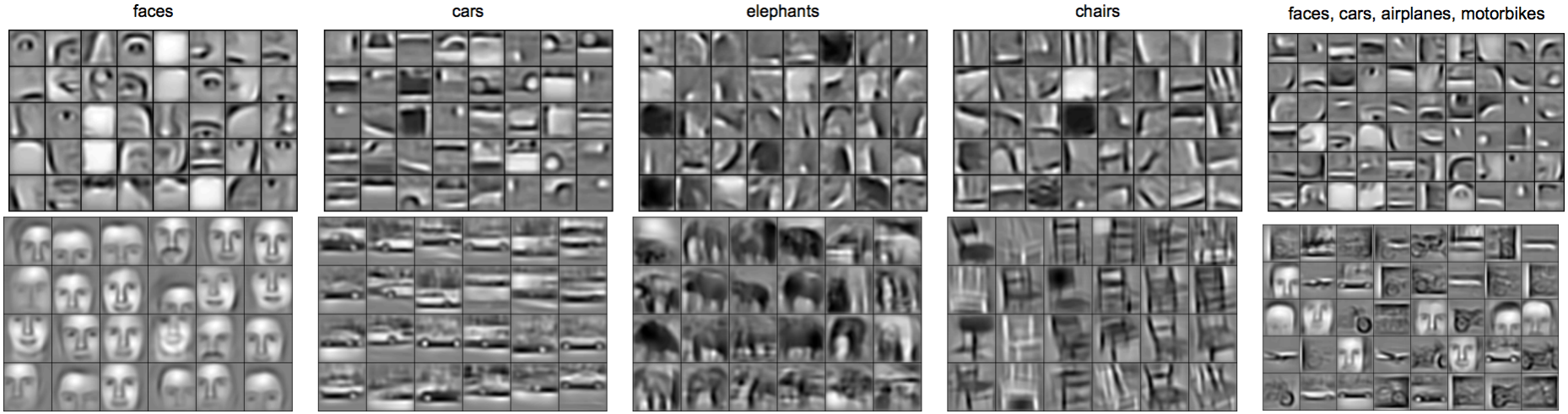
This structure is stacked, typically 2 to 4 times.



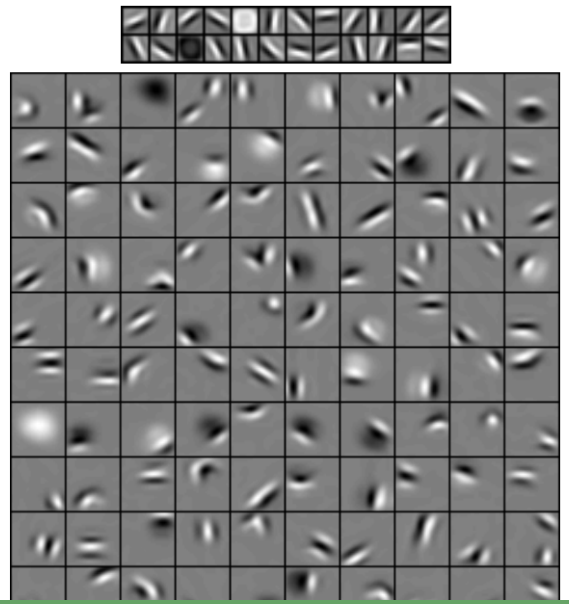
The power of a cNN is that the convolution filters are *learned*.

convolutional deep learning example (Ng 2009)

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



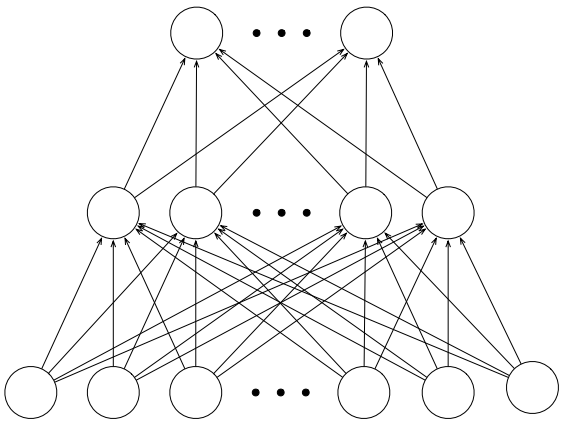
for natural images



unsupervised learning with neural networks

problem: often we have unlabelled data

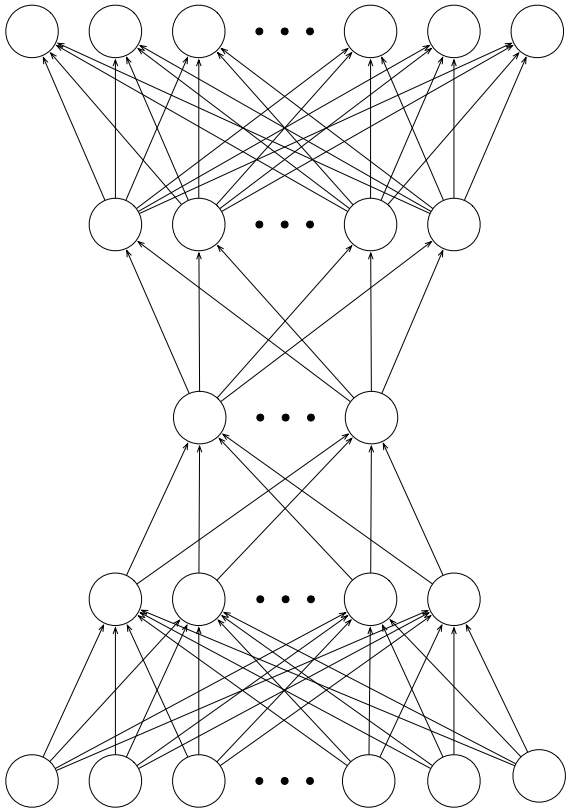
1. we have many data $\boldsymbol{x}$...
2. but no related output $\boldsymbol{z}$.



auto-encoder networks

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

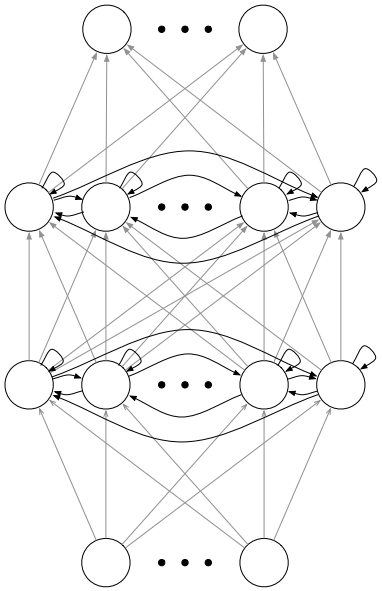
1. stacking a second neural network on top of the first
2. letting the whole NN compute $\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{x}$.



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

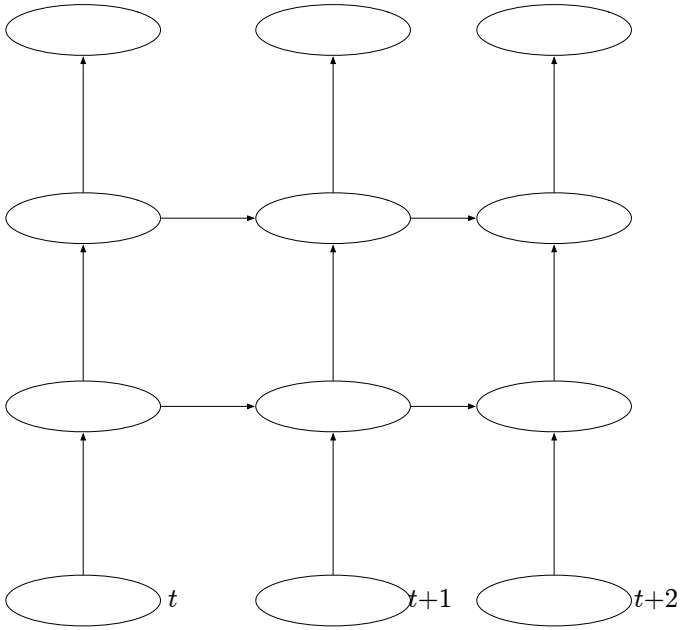
These networks make a compact representation of data ("dimensionality reduction"). However, you cannot control the representation!

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.

probabilistic neural networks

Neural networks become much more powerful if they are probabilistic. In these, each neuron represents a random variable rather than a value. In cleartext, a neuron in fact represents a mean and a variance, i.e., two values rather than one are learned.

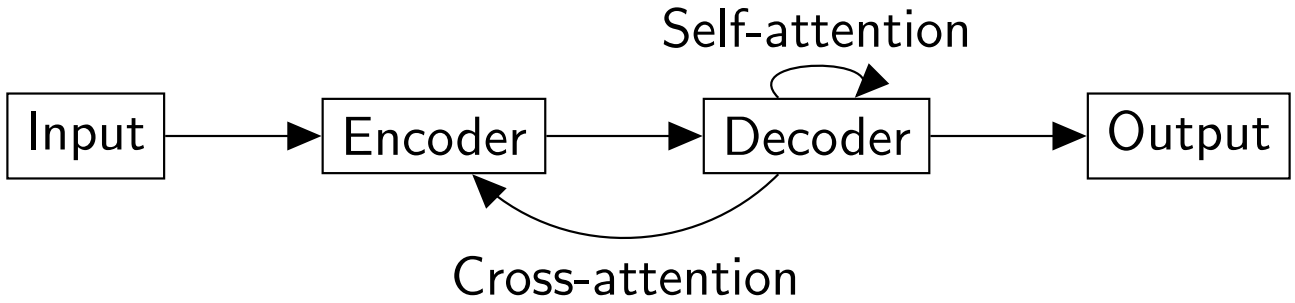
The probabilistic neural network introduce many possibilities:

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

What is a transformer?

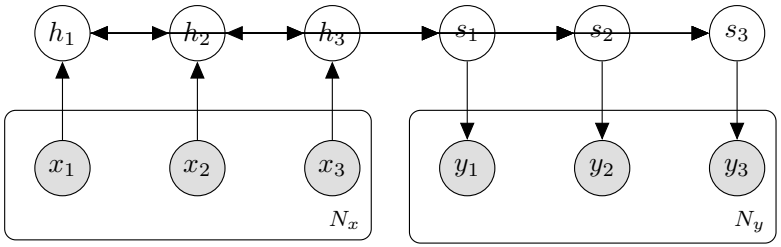
- A transformer is a neural network architecture that was proposed by Vaswani et al. (2017) for natural language processing tasks.
- It is based on the idea of self-attention, which is a mechanism that allows the network to learn the relationships between different parts of the input sequence.
- A transformer consists of two main components: an encoder and a decoder. The encoder processes the input sequence and generates a representation for each token. The decoder generates the output sequence by attending to the encoder representation and its own previous outputs.
- A transformer can handle variable-length inputs and outputs without using recurrent or convolutional layers, which makes it more efficient and scalable.

How does a transformer work?



- The encoder and the decoder are composed of multiple layers, each containing a multi-head self-attention sublayer and a feed-forward sublayer with residual connections and layer normalisation.
- The self-attention sublayer allows each token to attend to all other tokens in the same sequence, and computes a weighted sum of their representations. The multi-head mechanism allows the network to learn different aspects of attention from different subspaces of the representation.
- The cross-attention sublayer allows each token in the decoder to attend to all tokens in the encoder, and computes a weighted sum of their representations. This enables the decoder to generate outputs that are relevant to the input.
- The feed-forward sublayer consists of two linear transformations with a non-linear activation function in between. It uses the same function to each token independently, and enhances the representation capacity of the network.

transformers graphical model



The nodes represent random variables and the edges represent conditional dependencies. The shaded nodes are observed variables and the unshaded nodes are latent variables. The plates indicate repeated variables.

In this graphical model, the input sequence is denoted by $x_1, x_2, \ldots, x_{N_x}$ and the output sequence is denoted by $y_1, y_2, \ldots, y_{N_y}$. The encoder generates a representation for each input token, denoted by $h_1, h_2, \ldots, h_{N_x}$. The decoder generates a hidden state for each output token, denoted by $s_1, s_2, \ldots, s_{N_y}$.

The edges from the input tokens to the encoder representations indicate that each h_i depends on x_i. The edges from the encoder representations to the decoder hidden states indicate that each s_j depends on all h_i's. The edges from the decoder hidden states to the output tokens indicate that each y_j depends on s_j. The edges from the decoder hidden states to the next decoder hidden states indicate that each s_j also depends on the previous s_{j-1}.

The graphical model captures the conditional distribution of the output sequence given the input sequence:

$$p(y_1, y_2, \ldots, y_{N_y} | x_1, x_2, \ldots, x_{N_x})$$

getting started

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

For efficient code, use one of many public domain libraries, nowadays evolving around PyTorch, Tensorflow, JAX. Which one will you choose?

Zillions of tutorials to be found. For instance, check this recurrent neural network to translate between languages. `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`