(brief) intro to neural networks

image classification

X (matrix)	Y (vector)
Row 1 (30*30 x 1)	0 (cat)
Row 2 (30*30 x 1)	1 (dog)
Row 3 (30*30 x 1)	0 (cat)

Goal: Learn function $f : \mathbb{R}^{900} \rightarrow \mathbb{R}^{1}$

(Deep Learning may be a better term)

Typical Statistics----------Artificial Intelligence

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

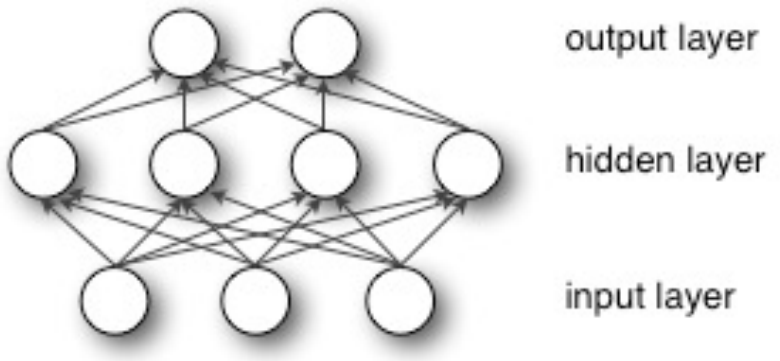
- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure so that it can be learned.

G. Hinton

Recall logistic regression

$$p_i = \sigma(w_1 x_1^i + \ldots . + w_n x_n^i)$$

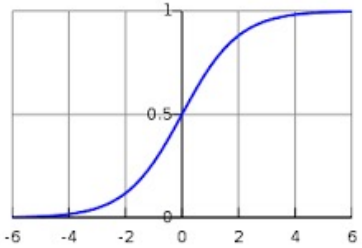
Simple “feed-forward” network (multilayer perceptron)



$$\vec{h} = f(W\vec{x})$$

$$\vec{y} = f(V\vec{h})$$

- W, V are called weight matrices
- f is “activation” function (sigmoid, ReLU)



“Train” model with stochastic gradient descent

Model

$$p_i = \sigma(w_1 x_1^i + \ldots . + w_n x_n^i)$$

Likelihood L

$$\max_{\vec{w}} \sum_i y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

“Train” model with stochastic gradient descent

Model

$$p_i = \sigma(w_1 x_1^i + \ldots . + w_n x_n^i)$$

Likelihood L

$$\max_{\vec{w}} \sum_i y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

Optimization: stochastic gradient descent (iteratively finds max where derivative is 0)

$$\vec{w}^{k+1} = \vec{w}^k + \gamma \nabla_{\vec{w}} L$$

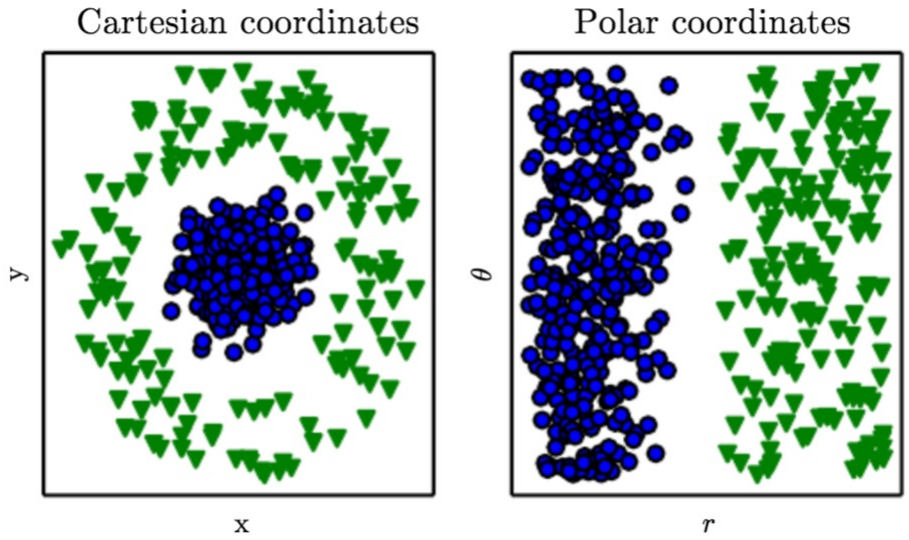
Training Neural Networks

- Gradient based optimization
  - "Back-propagate" gradients through network…this is just the chain rule for differentiation (can efficiently compute recursively)
  - Stochastic Gradient Descent (SGD), ADAM (adaptive gradients…uses info on second derivative)

- Many parameters (Google/FB ~billion)
  - can get stuck in local minima
  - can overfit
  - need to use ReLU (trains faster), regularization (L1/L2), dropout

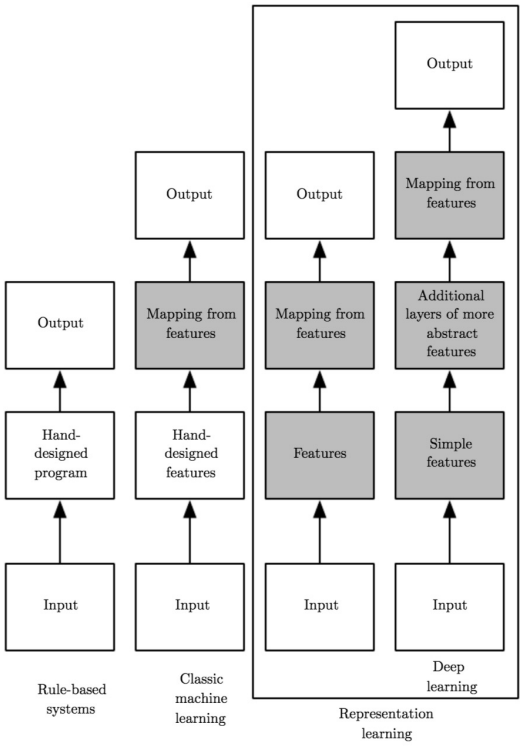
- Lots of linear algebra for large networks
  - there are linear algebra libraries that handle back-propagation "algebra" and use GPUs to speed up training
  - tensorflow, pytorch

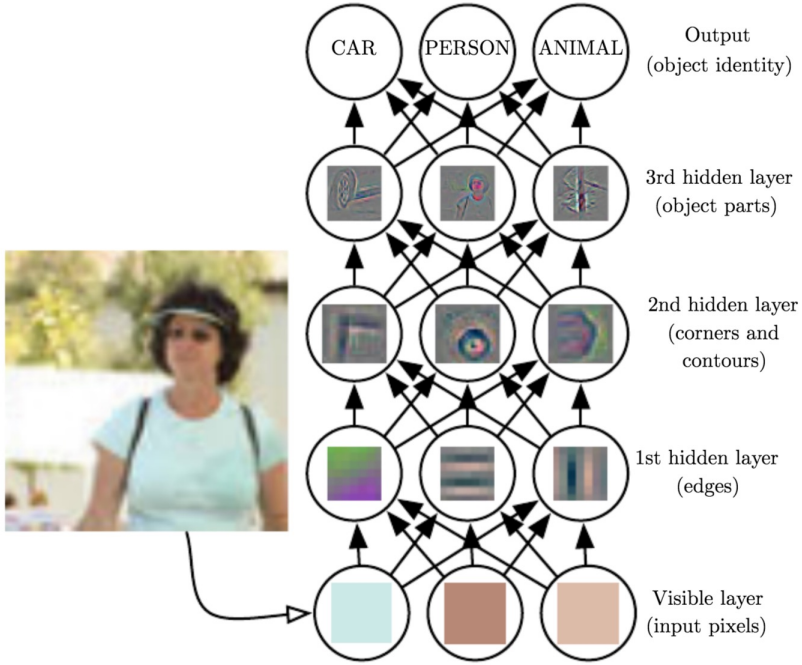
Learning representations



https://www.deeplearningbook.org/

Learning (deep) representations

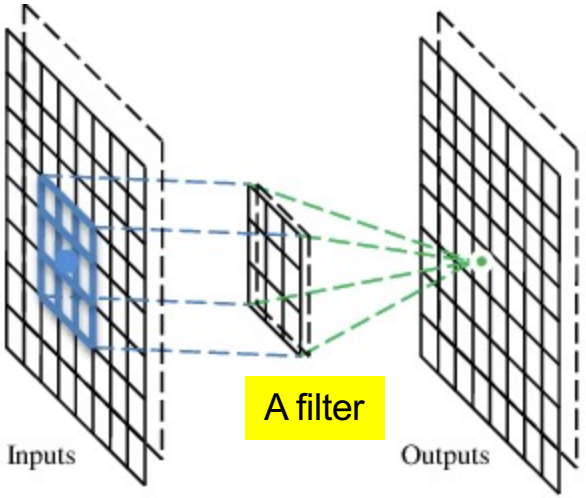




https://www.deeplearningbook.org/

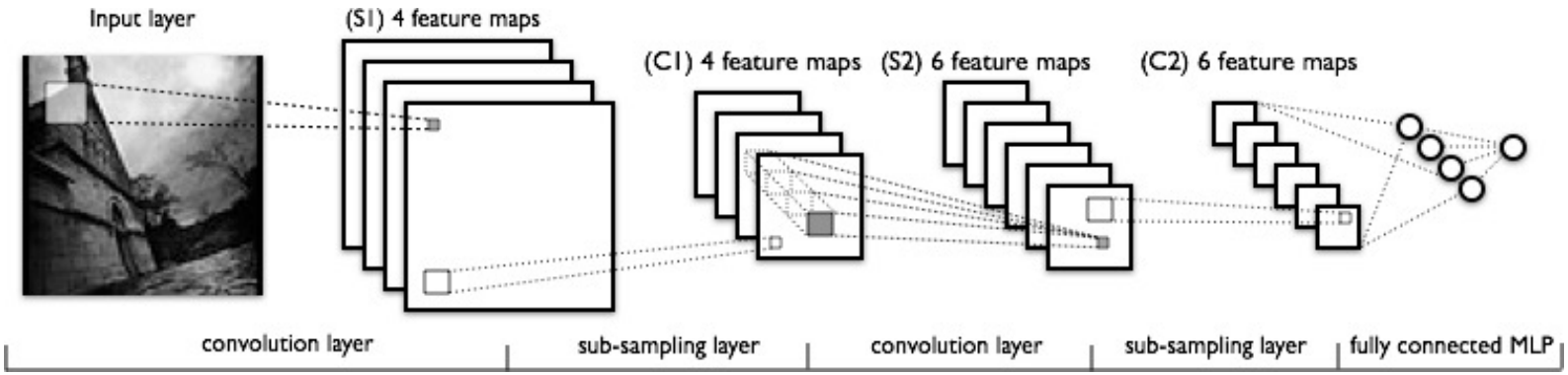
Convolution neural network (CNN)

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



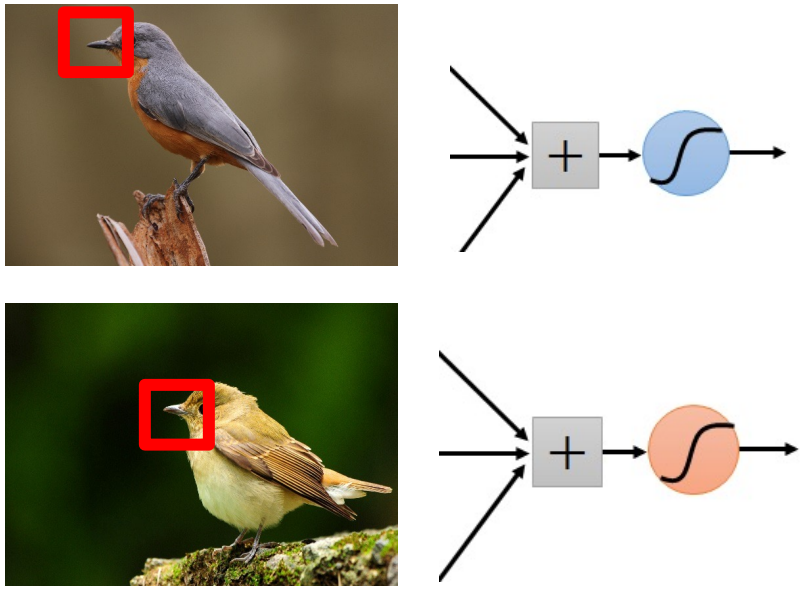
Convolution neural network (CNN)

$$Y_n^{ij} = \sum_{i'=-2,j'=-2}^{2} w_n^{i',j'} X_n^{i-i',j-j'}$$



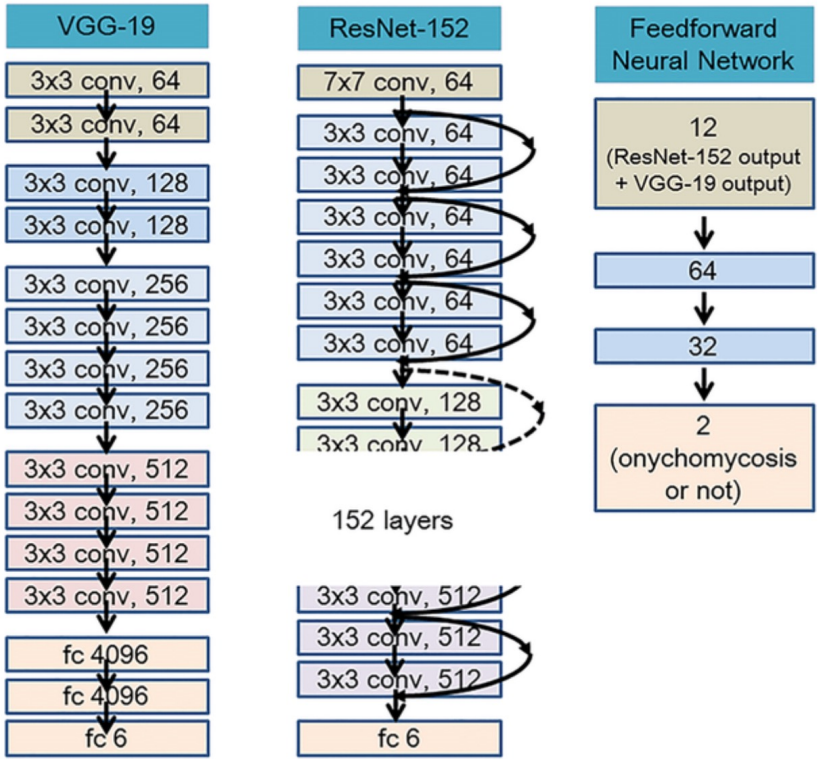
Example Architecture

CNNs “learn” features

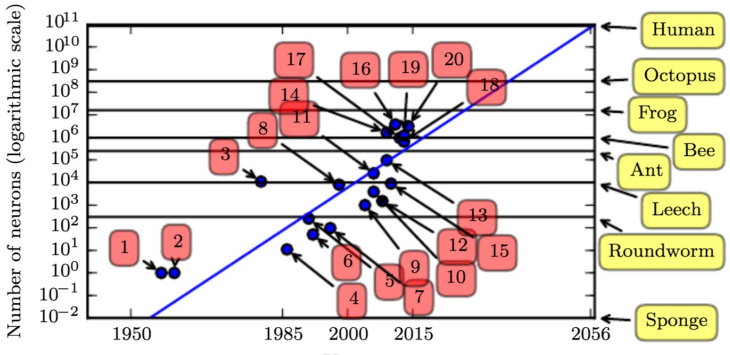


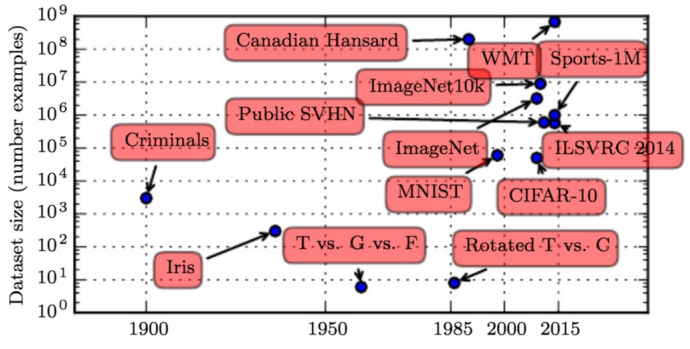
CNNs then reduce dimension by max pooling

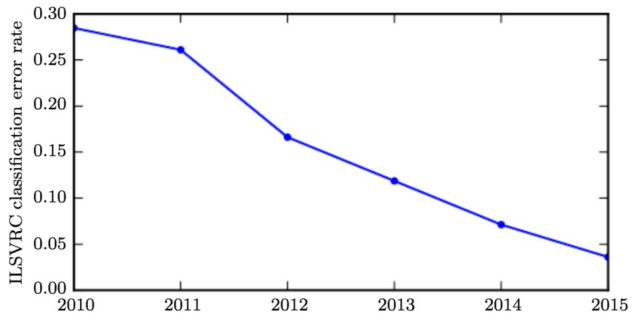
Resnets: very “deep” networks with skip/residual connections



Recent progress: larger nets, more data, higher accuracy (along with problem specific representations)







https://www.deeplearningbook.org/