Neural networks

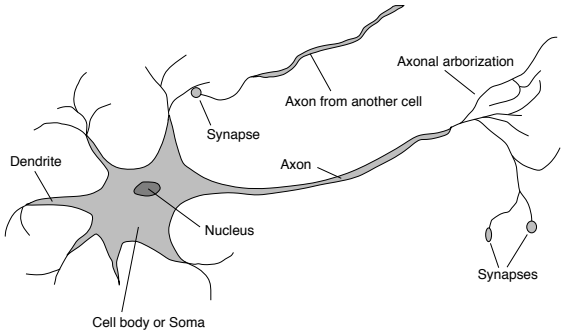
Chapter 20, Section 5

Outline

- ♢ Brains
- ♢ Neural networks
- ♢ Perceptrons
- ♢ Multilayer perceptrons
- ♢ Applications of neural networks

Brains

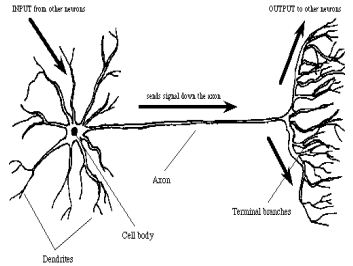
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



Neural networks

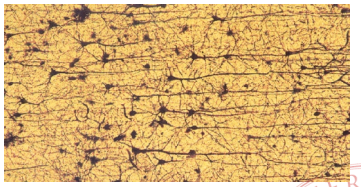
Inspired by the brain model. A network with:

- Many computational units (neurons) with low computational power
- many (weighted) connections
- Distributed control highly parallel



Approach completely different from the symbolic one:

- Not explicit knowledge
- Knowledge embedded int the structure of the network and the weights of the connections.



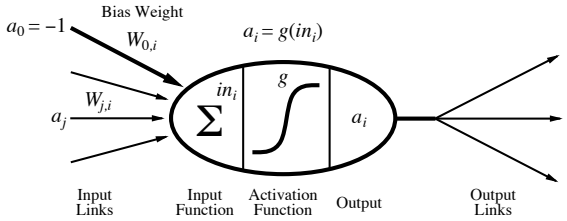
Learning ability

- Learning from differences between known target and observed facts (I/O)
- Learning mathematical functions associated to node computation

McCulloch–Pitts "unit"

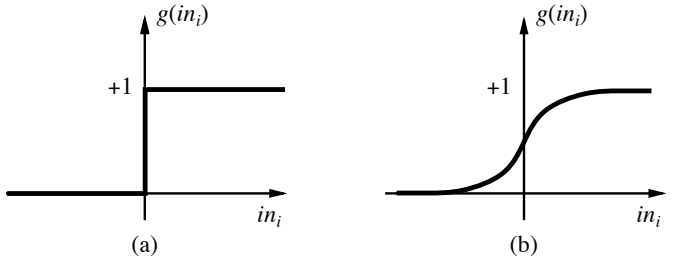
Output is a "squashed" linear function of the inputs:

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions



(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

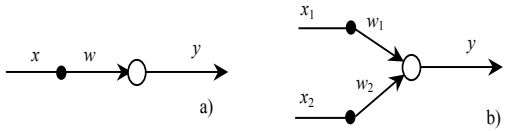
An example: representation of boolena functions

We consider elementary Boolean functions NOT, OR, AND, XOR. Table 1 shows these functions.

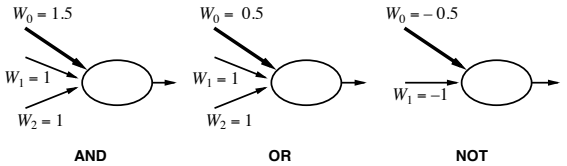
Table 1. Elementary Boolean functions

x_1	x_2	$\overline{x_1}$	$x_1 + x_2$	$x_1 \cdot x_2$	$x_1 \oplus x_2$
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

The basic question is, whether one neuron is enough to represent each of these Boolean functions. The neural networks to represent the first three functions are shown in figure 4.



Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

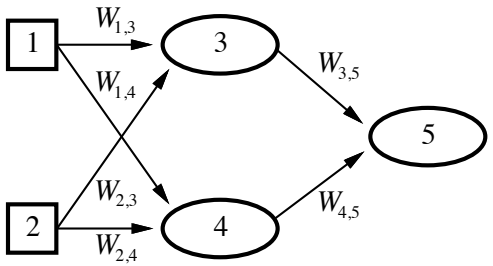
Network structures

Feed-forward networks:
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:
- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  $g(x) = \text{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**
- Boltzmann machines use stochastic activation functions,
  $\approx$ MCMC in Bayes nets
- recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

Feed-forward example

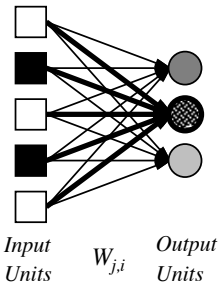


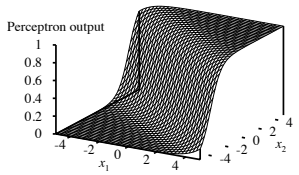
Feed-forward network = a parameterized family of nonlinear functions:

$$a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4)$$
$$= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))$$

Adjusting weights changes the function: do learning this way!

Single-layer perceptrons





Output units all operate separately—no shared weights

Adjusting weights moves the location, orientation, and steepness of cliff

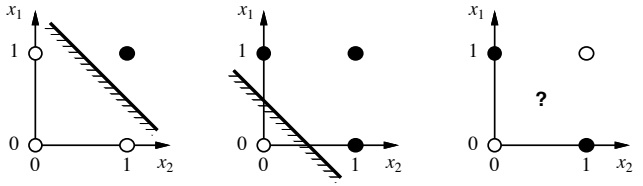
Expressiveness of perceptrons

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$\Sigma_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Minsky & Papert (1969) pricked the neural network balloon

Perception Learning rule

$h_W(x)$ = treshold $(W \bullet x)$ (or equivalently = treshold $(\Sigma_i w_i x_i)$

Error = $y - h_W(x)$

Idea of perception learning: update the weights to minimize the error.

We cannot use gradient descent because treshold is not differentiable in 0. We use the following rule

$W_i \leftarrow W_i + \alpha(y - h_W(x)) * x_i$

This is essentially linear regression, even though here we have a classification problem (output is 0 or 1). Intuition:

1) Y = 1 and $h_W(x) = 0$: w_i is increased if x_i is positive and w_i is decreased if x_i is negative. We want to increase the output $h_W(x)$
2) Y = 0 and $h_W(x) = i$: w_i is increased if x_i is negative and w_i is decreased if x_i is positive. We want to decrease the output $h_W(x)$

Univariate linear regression

Univariate linear function (= straight line)

$y = w_1 x + w_o$

Using the notation w for the vector $[w_0, w_1]$ we can write previous equation as follows

$h_W(x) = w_1 x + w_o$

Given a training set of n points (x,y) in the plane linear regression consists finding the best h which fit the data.

We can do so minimizing the error treshold $(W \bullet x)$ (or equivalently = treshold $(\Sigma_i w_i x_i)$

$Error(h_W) = \Sigma_j (y_j - h_W(x_j))^2 = \Sigma_j (y_j - w_1 x + w_o)^2$

This function can be minimized by considering the values of w_0 and w_1 for which the partial derivatives w.r.t. w_0 and w_1 are 0.

The equations $\delta\, Error(h_W) / \delta\, w_1 = 0$ and $\delta\, Error(h_W) / \delta\, w_1$ have a unique solution for every linear regression problem with the above error function.

This end the story for linear models.

Gradent descent

To go beyond linear models we can use gradent descent: we start from a point in the weight space (here the (w_0, w_1) plane and we move to a neighboring point that is "downhill" w.r.t. the Error function, repeating untill we converge

$W \leftarrow$ any point in the space

Loop until convergence do

For each w_i in w do

$w_i \leftarrow w_i - \alpha * \delta\, Error(h_W) / \delta\, w_i$

Perceptron learning

Learn by adjusting weights to reduce error on training set

The squared error for an example with input $\mathbf{x}$ and true output y is

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

Perform optimization search by gradient descent:

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\Sigma_{j=0}^{n} W_j x_j)\right)$$
$$= -Err \times g'(in) \times x_j$$

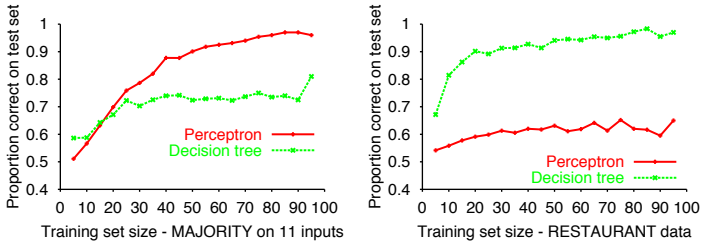
Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

E.g., +ve error $\Rightarrow$ increase network output
$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning contd.

Perceptron learning rule converges to a consistent function
for any linearly separable data set

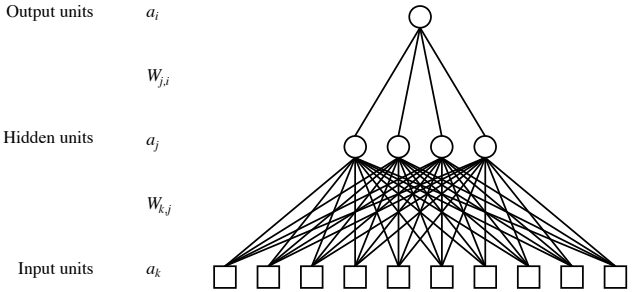


Perceptron learns majority function easily, DTL is hopeless

DTL learns restaurant function easily, perceptron cannot represent it

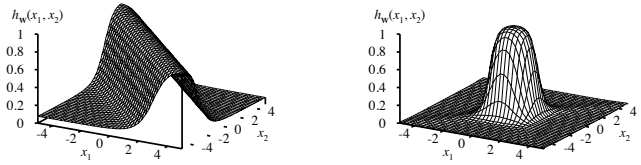
Multilayer perceptrons

Layers are usually fully connected;
numbers of hidden units typically chosen by hand



Expressiveness of MLPs

All continuous functions w/ 2 layers, all functions w/ 3 layers



Combine two opposite-facing threshold functions to make a ridge

Combine two perpendicular ridges to make a bump

Add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units (cf DTL proof)

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 \ ,$$

where the sum is over the nodes in the output layer.

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\ &= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i\end{aligned}$$

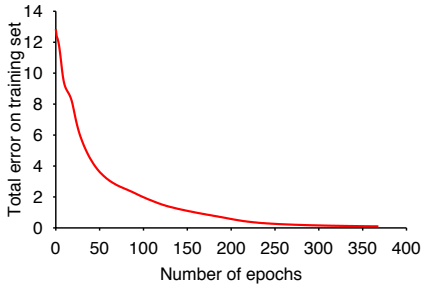
Back-propagation derivation contd.

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} \\ &= -\sum_i (y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i}a_j\right) \\ &= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial in_j}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j}a_k\right) \\ &= -\sum_i \Delta_i W_{j,i}g'(in_j)a_k = -a_k\Delta_j\end{aligned}$$

Back-propagation learning contd.

At each epoch, sum gradient updates for all examples and apply

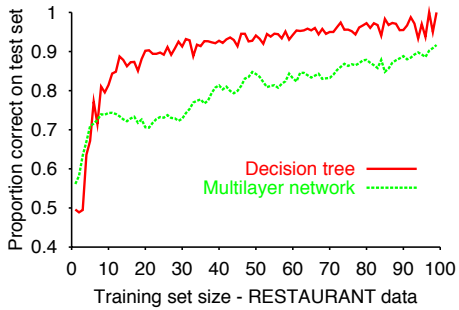
Training curve for 100 restaurant examples: finds exact fit



Typical problems: slow convergence, local minima

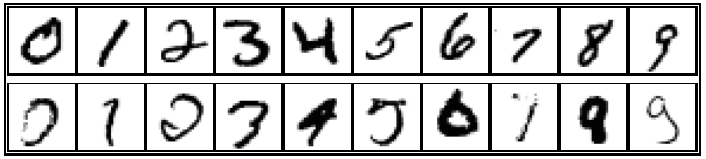
Back-propagation learning contd.

Learning curve for MLP with 4 hidden units:



MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily

Handwritten digit recognition



3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error

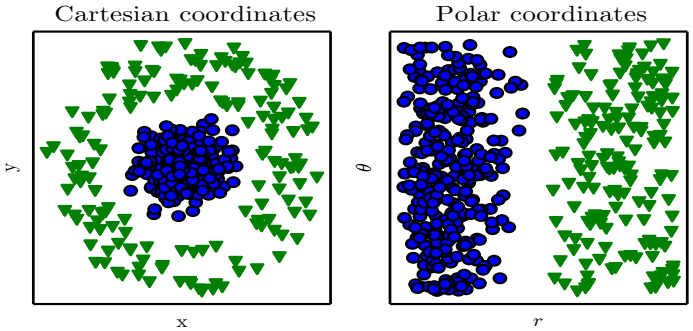
Current best (kernel machines, vision algorithms) ≈ 0.6% error

Including material from the book:

Deep Learning.
Ian Goodfellow , Yoshua Bengio and Aaron Courville.
MIT Press. 2016

http://www.deeplearningbook.org

Representation of data is important



Polar coordinates allow to separate data by using a simple vertical line, this is impossible with cartesian coodinates.

Not all the details of data are really important, we need to extract some features

Feature

A feature is an individual measurable property or characteristic of a phenomenon being observed.

In other words, a piece of information included in the representation of data.

For example, when a ML technique (logistic regression) is used to recommend cesarean delivery, the AI system does not examine the patient directly, but several pieces of relevant information, such as the presence or absence of a uterine scar.

Selection of the right features is difficult and crucial.

Feature selection

Consider the task of detecting a car in a photography: certainly they have wheels, so we could think to use the presence of a wheel as a feature. However it is difficult to describe a wheel in terms of pixels

- The simple geometric shape can be complicated by shadows, sun, fender of the car etc.

A possible solution is representation learning: to use ML not only to discover the output but also to discover the representation of data (i.e. the features). Example, Autoencoder, that is encoder + decoder

encoder : original data -> representation

decoder: representation -> original data (with better properties)

Factors of variation

Those factors which explain the observed data (and influence our understanding of data).

- Often this are not quantities directly observed but concepts and abstractions.
- They can also be due to constructs in the human mind.
- For example, when considering a speech, FoV are speaker's age, sex, accent ..

  When considering the image of a car FoV are, position, color, angle w.r.t. the sun

The main difficutly is to disentangle the FoV and discard those which are not useful. Some FoV influence each piece of data we observe

And some of them are very difficult to extract (e.g. the accent is probably non important but it is difficult to extract from data).

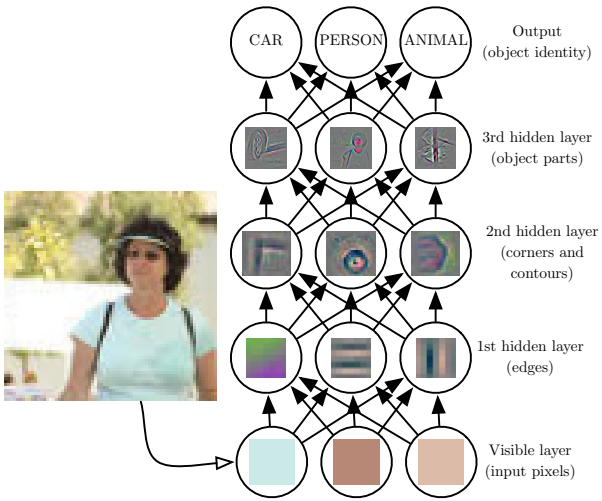
Deep learning

Disentangle the factors of variation and discard those which are not useful.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts. See next picture:

The input is presented at the visible layer so named because it contains the variables that we are able to observe. Then a series of hidden layers extracts increasingly abstract features from the image. These layers are called "hidden" because their values are not given in the data; instead the model must determine which concepts are useful for explaining

the relationships in the observed data. The images here are visualizations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image. Images reproduced from Zeller and Fergus, 2014.



Deep learning

A simple DL model is Multilayer Perceptron (MP) which is just a function mapping some set of input values to output values. The function is formed by composing many simpler functions.

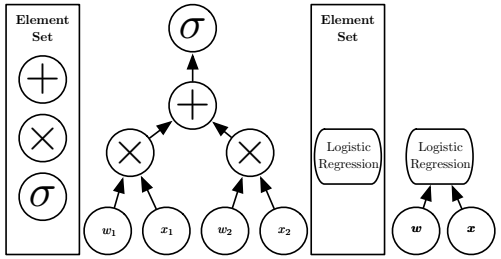
1) We can think of each application of a different function as providing a new representation of the input, so deep learning = learning the right representation of data.

2) We can consider depth as what enables the computer to learn a multistep computer program. Each layer of the representation is the state of the computer's memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence.

What is depth and when a model il deep ?

Two definitions of depth:

1) Number of sequential instructions that need to be executed in the model (i.e. number of nodes in a path in the computational graph. In this case the depths depends on the language, see below)
2) Depth of the graph describing how concepts are related to each other (usually smaller than the previous one)
3) No consensus on a number for "deep"



Why many levels

We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?

$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition

Can represent more complex functions with less "hardware"

We need an efficient parameterization of for functions that are useful for "AI" tasks (vision, audition, NLP...)

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

Revolution caused by feature learning

Speech Recognition I (late 1980s)
- Trained mid-level features with Gaussian mixtures (2-layer classifier)

Handwriting Recognition and OCR (late 1980s to mid 1990s)
- Supervised convolutional nets operating on pixels

Face & People Detection (early 1990s to mid 2000s)
- Supervised convolutional nets operating on pixels (YLC 1994, 2004, Garcia 2004)
- Haar features generation/selection (Viola-Jones 2001)

Object Recognition I (mid-to-late 2000s: Ponce, Schmid, Yu, Y le Cun....)
- Trainable mid-level features (K-means or sparse coding)

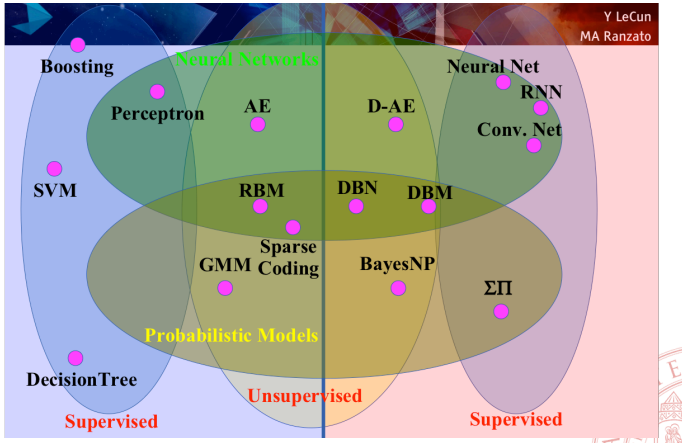
Low-Res Object Recognition: road signs, house numbers (early 2010's)
- Supervised convolutional net operating on pixels

Speech Recognition II (circa 2011)
- Deep neural nets for acoustic modeling

Object Recognition III, Semantic Labeling (2012, Hinton, YLC,...)
- Supervised convolutional nets operating on pixels

Learning models

Shallow Deep



Curse of Dimensionality

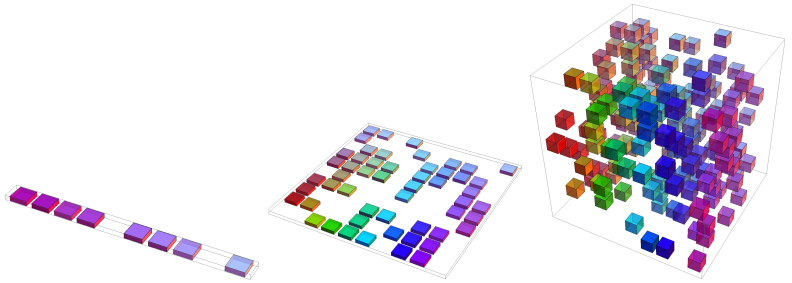
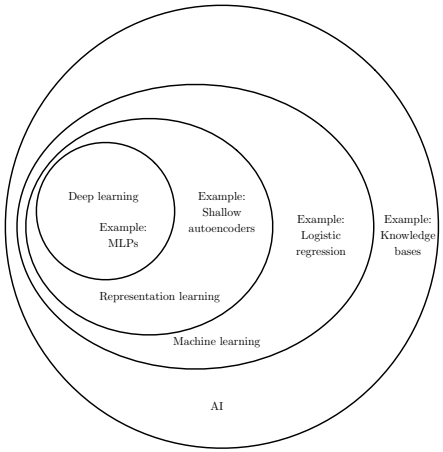


Figure 5.9

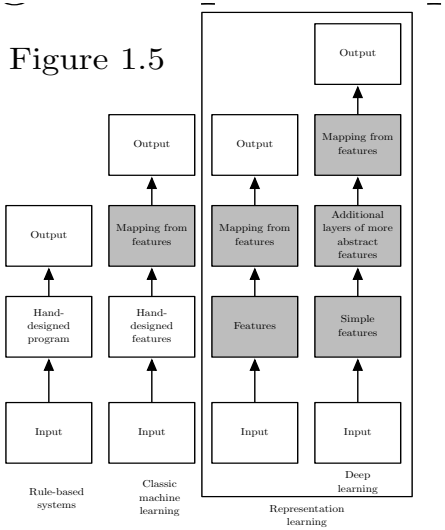
(Goodfellow 2016)

Machine learning and AI

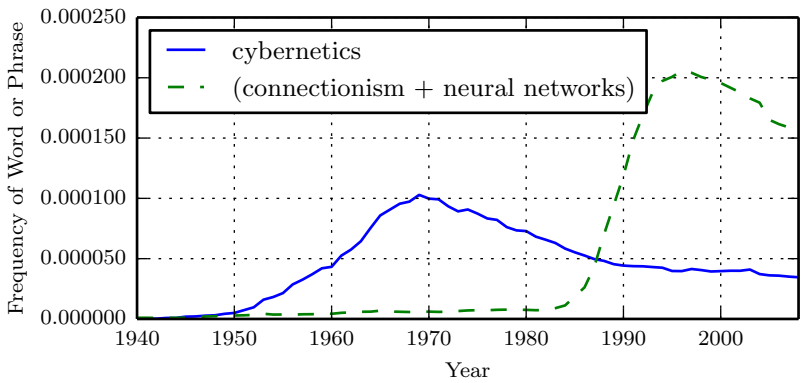


Deep learning vs. other AI systems

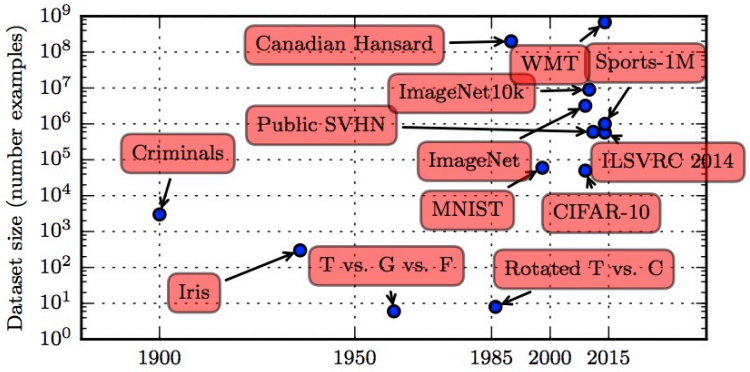
Figure 1.5



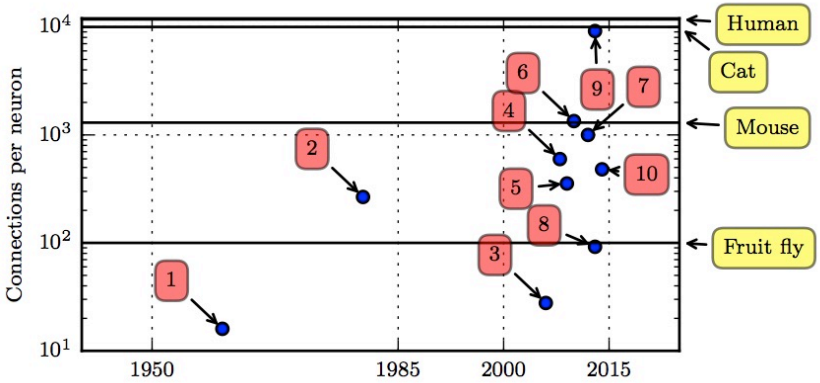
Trend



Dataset size



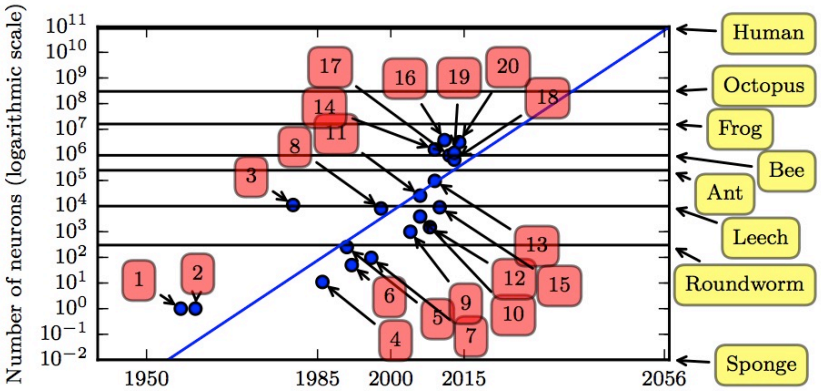
Connections per neuron



Legenda

1. Adaptive Linear element (Widrow and Hoff, 1960)
2. Neocognitron (Fukushima, 1980)
3. GPU-accelerated convolutional network (Chellapilla et al., 2006)
4. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
5. Unsupervised convolutional network (Jarrett et al., 2009)
6. GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
7. Distributed autoencoder (Le et al., 2012)
8. Multi-GPU convolutional network (Krizhevsky et al., 2012)
9. COTS HPC unsupervised
10. Goole Net

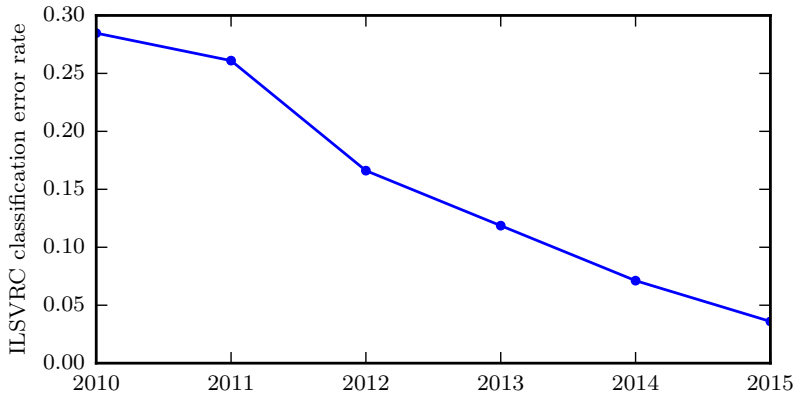
Number of neurons



Legenda

1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart et al., 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio et al., 1991)
7. Mean field sigmoid belief network (Saul et al., 1996)
8. LeNet-5 (LeCun et al., 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton et al., 2006)
11. GPU-accelerated convolutional network (Chellapilla et al., 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina et al., 2009)
14. Unsupervised convolutional network (Jarrett et al., 2009)
15. GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le et al., 2012)
18. Multi-GPU convolutional network (Krizhevsky et al., 2012)
19. COTS HPC unsupervised convolutional network (Coates et al., 2013)
20. GoogLeNet (Szegedy et al., 2014a)

Object recognition



Further reading

Deep Learning.
Ian Goodfellow , Yoshua Bengio and Aaron Courville.
MIT Press. 2016
http://www.deeplearningbook.org
(available on line, very good book on the topic)

Deep Learning: A Critical Appraisal

Gary Marcus. 2 january 2018.

https://arxiv.org/abs/1801.00631

(interesting for a different view on DL)

Weka

https://www.cs.waikato.ac.nz/ml/weka/