Machine Learning

Neural Networks and Deep Learning: Fundamentals

Gabriele Russo Russo   Francesco Lo Presti

Laurea Magistrale in Ingegneria Informatica – A.Y. 2023/24

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

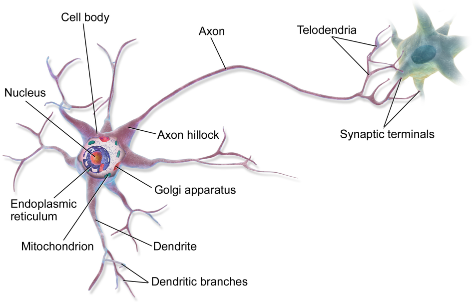
Dipartimento di Ingegneria Civile e Ingegneria Informatica

From Human Brain to ANNs

- Science and technology often inspired by nature
  - e.g., birds inspired us to build aircrafts
- From human brain to intelligent machines?
- Challenging task: computational power of brain much higher than modern computers
- Artificial neural network (ANN): popular ML technique that simulates the mechanism of learning in biological organisms

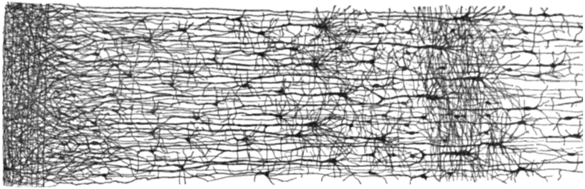
Biological Neurons

- Cell body + branching extensions (dendrites) + very long extension (axon)
- Axon splits off into branches (telodendria) at its extremity
- Synaptic terminals (or, synapses) at the tip of these branches, connected to dendrites or cell bodies of other neurons



Biological Neurons (2)

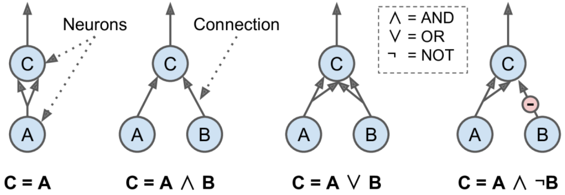
- Neurons produce short electrical impulses, which make synapses release chemical signals called neurotransmitters
- When a neuron receives a sufficient amount of specific neurotransmitters within a few milliseconds, it fires its own electrical impulses
- Single neurons are pretty simple, but they are organized in a network of billions
  - Each connected to 1,000+ neurons (often organized in layers)
  - Highly complex computations performed by such networks



History: McCulloch and Pitts (1943)

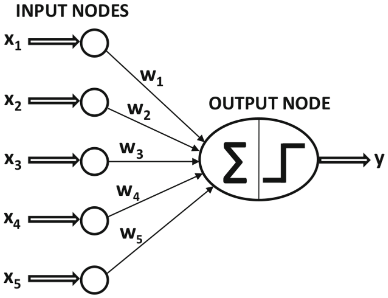
A Logical Calculus of Ideas Immanent in Nervous Activity

- Simplified computational model of how biological neurons might work (first ANN architecture)
- Neuron: 1+ binary inputs, 1 binary output
- Output activated when at least X inputs are active
- Example: activation with at least 2 active inputs



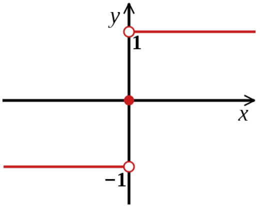
Perceptron

- Different model, proposed by F. Rosenblatt in 1957
- Inputs and outputs are real numbers
- Each input has an associated weight
- Output computed through an activation function ϕ (e.g., the sign function)



spesso possiamo usare, in modo equivalente, i termini
PESI o PARAMETRI,
anche il bias ne fa parte!

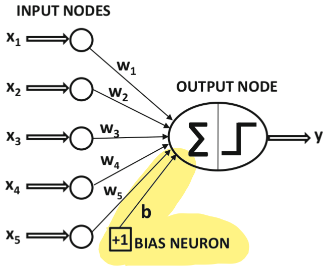
$$y = \phi\left(\sum_{j=1}^{d} w_j x_j\right) = \phi(\mathrm{w}^T \mathrm{x}) = sgn(\mathrm{w}^T \mathrm{x}) \quad (1)$$



Perceptron with Bias

Quando eseguiamo la predizione, c'è una componente indipendente dagli input che entra nella operazioni di Sommatoria: il bias, che si aggiunge alla somma pesata.

- There is often an invariant part of the prediction, called bias
  - e.g., you need to predict a positive value when $x_j = 0\ \forall j$
- We consider an additional input node that always transmits a constant value 1 with connection weight b (bias variable)



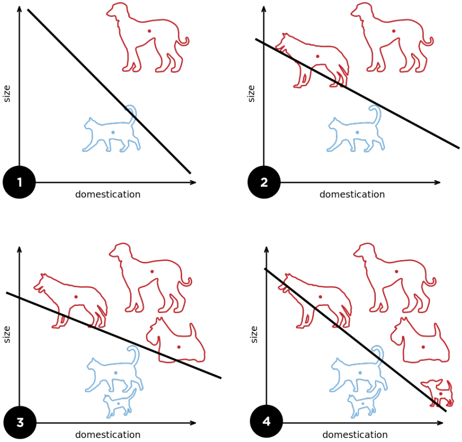
$$y = \phi\left(\sum_{j=1}^{d} w_j x_j + b\right) = \phi(\mathrm{w}^T \mathrm{x} + b) \tag{2}$$

Note: Hereafter, to simplify notation, we will often omit b in the equations

Perceptron for Classification

Inizialmente mappo su "-1" o "+1", basandosi su feature <dimensione animale, addomesticabile>
Ciò che ottengo è una retta, con una certa inclinazione, in funzione dei pesi "w" e bias.

- Perceptron can be seen as a binary classifier
- It suffices to map the output values $\{-1, 1\}$ to the target classes



Perceptron: Training

Rosenblatt proposed a heuristic algorithm for training

- Weights initialized arbitrarily (e.g., $\mathrm{w} = 0$)
- Perceptron is given one training instance $\mathrm{x}^{(i)} \in \mathcal{D}$ at a time to make a prediction y
- In case of error ($y^{(i)} \neq t^{(i)}$), “reinforce” connections that would contribute to a correct prediction

$$w_j \leftarrow w_j + (t^{(i)} - y^{(i)})x_j^{(i)} \tag{3}$$

- Iterate for a fixed number of epochs (or, other stopping criteria)

Il perceptron da come predizione "-1" o "+1", che devo confrontare con la soluzione "t".
Se toppo, vado a modificare il peso di quella istanza. Se la predizione è corretta, t(i) e y(i) sono uguali, e quindi non cambio il peso. Se predizione è "+1", e sbaglio "-1", aumento peso (1-(-1)) = 2.
Nel caso contrario diminuisco il peso.

Example

Classifying Iris Setosa flowers using a Perceptron.

perceptron.ipynb

Perceptron and SGD

- The original paper by Rosenblatt did not consider any explicit loss function to optimize and only proposed a training heuristic

$$\mathrm{w} \leftarrow \mathrm{w} + (t^{(i)} - y^{(i)})x^{(i)} \tag{4}$$

- Later, the algorithm was reverse-engineered and interpreted as an application of SGD
- Keep in mind that the original algorithm is not SGD though!
  - e.g., Perceptron can be fed training data without any randomization, it is not stochastic!

Non abbiamo stocasticità!

Perceptron and SGD (2)

- Consider a 0-1 loss function for the instance $(\mathrm{x}^{(i)}, t^{(i)})$

$$\mathcal{L}^{(i)}_{0/1} = \frac{1}{2}(t^{(i)} - sgn\{\mathrm{w}^T\mathrm{x}^{(i)}\})^2 = (1 - t^{(i)} sgn\{\mathrm{w}^T\mathrm{x}^{(i)}\}) \quad (5)$$

0 ok, 1 se sbaglio. Lavoro svolto dal Perceptron mediante vettori "x" e "w trasposto"

- The sign function is a problem for differentiability and, thus, SGD application Se il gradiente è sempre 0, non mi muovo!
- We consider a smoothed surrogate loss function:

$$\mathcal{L}^{(i)} = \max\{0, -t^{(i)}(\mathrm{w}^T\mathrm{x}^{(i)})\} \quad (6)$$

Usiamo una funzione "surrogata", che prende il massimo tra i due valori proposti. L'idea è che se t >0, allora segno è negativo, e vince 0.
Se t<0, col segno meno ottengo un valore >0, e quindi assumiamo questo valore come risultato.
Graficamente, questa funzione non è troppo diversa dall'originale, infatti ne mantiene lo stesso punto di minimo.

perceptron-sgd.ipynb

Perceptron and SGD (3)

$$\mathcal{L} = \sum_{i=1}^{N} \max\{0, -t^{(i)}(\mathrm{w}^T \mathrm{x}^{(i)})\} \tag{7}$$

- We can compute the gradient and the SGD update of the smoothed surrogate loss function:

$$\nabla_{\mathrm{w}} \mathcal{L}^{(i)} = (y^{(i)} - t^{(i)})\mathrm{x}^{(i)} \tag{8}$$

$$\mathrm{w} \leftarrow \mathrm{w} - \eta \nabla_{\mathrm{w}} \mathcal{L}^{(i)} = \mathrm{w} + \eta(t^{(i)} - y^{(i)})\mathrm{x}^{(i)} \tag{9}$$

With $\eta = 1$, we get back the Perceptron update.

Questa variabile (eta) rappresenta il "learning rate".
Se eta = 1, torniamo al caso originale del Perceptron.

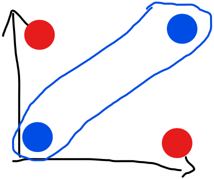
Beyond the Perceptron

- The Perceptron is a neural network with a single computational layer
- It can only learn linear decision boundaries

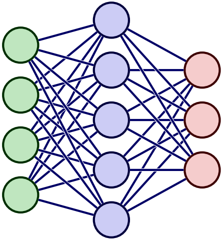
  perceptron-xor.ipynb

l'unico output che da è:
>0 o <0

- Much more powerful models can been obtained composing neurons into an artificial neural network
  - Feedforward (no cycles) vs Recurrent neural networks



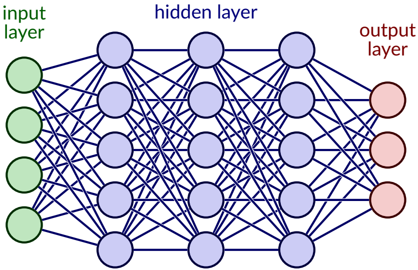
es: prendiamo XOR
(è 1 se c'è SOLO un 1)
(0 xor 0 ->0), (0 xor 1 -> 1)
(1 xor 0 ->1) (1 xor 1 ->0)



il grafico a destra non è tagliabile in due, la soluzione sarebbe un ellisse intorno ad uno dei due colori.
Scikit-learn ottiene, in questo caso, 50% di accuratezza, perchè dicendo sempre "0" (o sempre "1) ci prende nel 50% dei casi.

Feedforward Neural Network

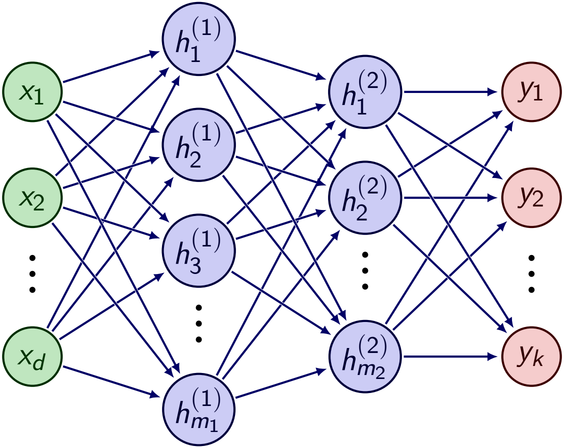
- Neurons (or, “units”) organized into layers:
  - A passthrough input layer
  - 1+ hidden layers
  - 1 output layer
- Fully connected layers
- Feedforward: information flows in one direction, from the input layer to the output layer; no feedback connections



Note: MLP

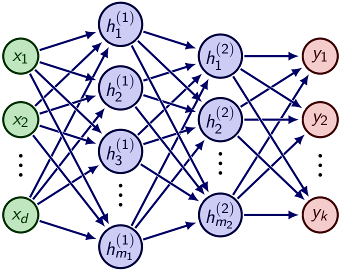
- Feedforward NNs have been originally referred to as multilayer perceptrons (MLP)
  - e.g., scikit-learn provides the `MLPClassifier` class
- While still used, MLP is considered to be a *misnomer*
- Besides the use of artificial neurons, modern feedforward NNs profoundly differ from the Perceptron model w.r.t. activation functions, loss function, training algorithm, regularization strategies, …

Notation and Terminology



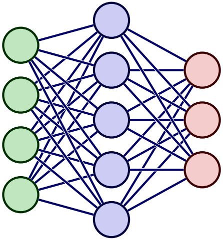
- x is the input vector
- $\mathrm{h}^{(i)}$ is the output of the i-th hidden layer
- y is the output vector

Notation and Terminology (2)



- the number of layers (excluding the input layer) L is the depth of the network
  - deep learning involves NNs with many layers
- the number of units in a layer is known as the width

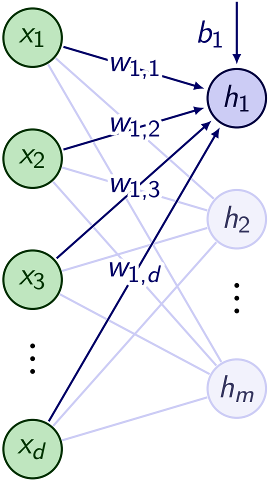
Activation Functions



- As for the Perceptron, units in the input layer simply output their own input unmodified
- Units in the hidden and output layers instead apply an activation function to their input
- We assume activation functions to be identical for all the units in the same layer, but they can differ across layers

Hidden Units

Let's consider the case of a single hidden layer:



$$h_1 = \phi(\sum_{j=1}^{d} w_{1,j} * x_j + b_i)$$

For the whole layer:

$$\mathrm{h} = \phi(\mathrm{Wx} + \mathrm{b})$$

where $\mathrm{W} \in \mathbb{R}^{m \times d}$ is a matrix of weights, and $\mathrm{b} \in \mathbb{R}^m$ is the bias vector.

Hidden Units (2)

In general, we have multiple hidden layers:

- $h_i^{(\ell)}$ denotes the i-th unit of the ℓ-th layer
- $w_{i,j}^{(\ell)}$ denotes the weight of the connection from the j-th unit of the $(\ell - 1)$-th layer to the i-th unit of the ℓ-th layer

So, for $\ell > 1$:

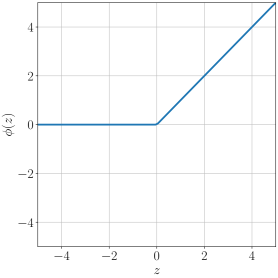
$$\mathsf{h}^{(1)} = \phi^{(1)}(\mathsf{W}^{(1)}\mathsf{x} + \mathsf{b}^1)$$

$$\mathsf{h}^{(\ell)} = \phi^{(\ell)}(\mathsf{W}^{(\ell)}\mathsf{h}^{(\ell-1)} + \mathsf{b}^\ell)$$

Activation Function: Examples

ReLU (Rectified Linear Unit)

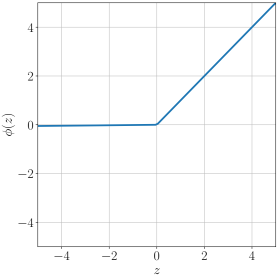
$$\phi(z) = \max\{0, z\}$$



Parametric ReLU (PReLU)

$$\phi(z) = \begin{cases} z & z > 0 \\ pz & z \leq 0 \end{cases}$$

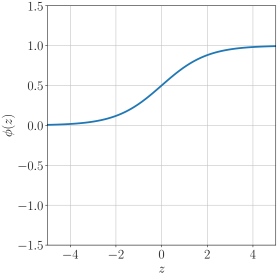
e.g., $p = 0.01$



Activation Function: Examples (2)

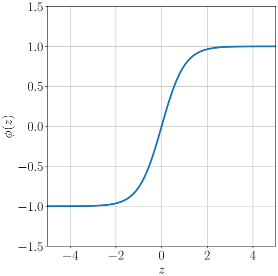
Logistic sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Output Units

- Different output units are used depending on the task
- A linear output layer can be used for regression:

$$\mathrm{y} = \mathrm{W}^{(L)}\mathrm{h}^{(L-1)} + \mathrm{b}^{(L)}$$

- A sigmoid unit can be used for binary classification:

$$\mathrm{y} = \sigma(\mathrm{W}^{(L)}\mathrm{h}^{(L-1)} + \mathrm{b}^{(L)})$$

- A softmax layer can be used for multiclass classification:

$$\mathrm{z} = \mathrm{W}^{(L)}\mathrm{h}^{(L-1)} + \mathrm{b}^{(L)}$$

$$y_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

NN and Function Composition

Let $f^{(\ell)}$ be the function implemented by the ℓ-layer, e.g.:

$$h^{(1)} = f^{(1)}(x) = \phi^{(1)}(W^{(1)}x + b^{1})$$

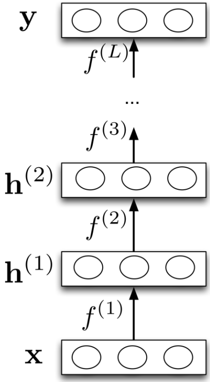
$$h^{(2)} = f^{(2)}(h^{(1)})$$

$$\dots$$

$$y = f^{(L)}(h^{(L-1)})$$

The NN computes the composite function:

$$y = f^{(L)} \circ f^{(L-1)} \circ \dots f^{(1)}$$

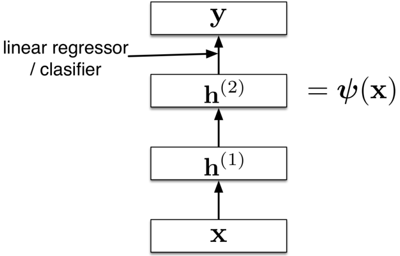


NN and Function Composition (2)

- The NN computes the composite function:

$$y = f^{(L)} \circ f^{(L-1)} \circ \dots f^{(1)}$$

- Output unit is often a linear function (regression) or a sigmoid (binary classification), similar to the Perceptron step function
- It is like applying a linear model to features $\psi(\mathrm{x})$ computed by the first $(L-1)$ layers



From Linear to Nonlinear

Is it so important to use nonlinear activation functions?

Let's consider a model with a single hidden layer h, a linear activation function, and a linear output function:

$$\mathsf{h} = \mathsf{W}^{(1)}\mathsf{x} + \mathsf{b}^{(1)} \tag{10}$$

$$\mathsf{y} = \mathsf{W}^{(2)}\mathsf{h} + \mathsf{b}^{(2)} \tag{11}$$

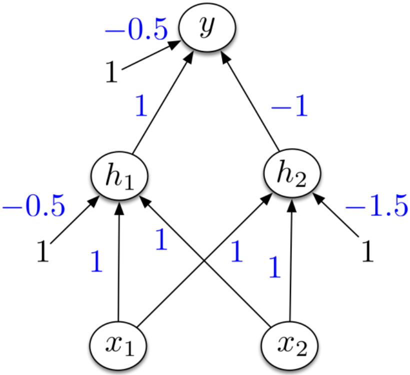
We get:

$$\begin{aligned} \mathsf{y} &= \mathsf{W}^{(2)}(\mathsf{W}^{(1)}\mathsf{x} + \mathsf{b}^{(1)}) + \mathsf{b}^{(2)} = && (12) \\ &= \mathsf{W}^{(2)}\mathsf{W}^{(1)}\mathsf{x} + \mathsf{W}^{(2)}\mathsf{b}^{(1)} + \mathsf{b}^{(2)} = && (13) \\ &= \mathsf{W}'\mathsf{x} + \mathsf{b}' && (14) \end{aligned}$$

With linear activations, you end up with a linear model, regardless of how many layers you use!

Example: XOR

Let's consider the following NN, with the Heaviside step function $H(z)$ for activation:



$$H(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

xor.ipynb

Expressivity

How "powerful" ANNs can be?

- We saw that linear models are easy to train, but they can only represent linear functions
- NNs overcome this limitation introducing nonlinearity
- Question: do we need a specialized model family for each nonlinear function to learn?

Universal Approximation Theorem

Theorem

A feedforward NN with a linear output layer, any “squashing” activation function, and enough hidden units can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error.

- *G. Cybenko, “Approximation by superpositions of a sigmoidal function” (1989)*
- *squashing*: activation output is a finite interval (e.g., $[0, 1]$)
- *Borel measurable function*, TL;DR: any continuous function on a closed and bounded subset of $\mathbb{R}^n$
- Initially proven for the sigmoid activation, then proven for various functions, including ReLU

Universal Approximation Theorem (2)

But, don't be too enthusiastic!

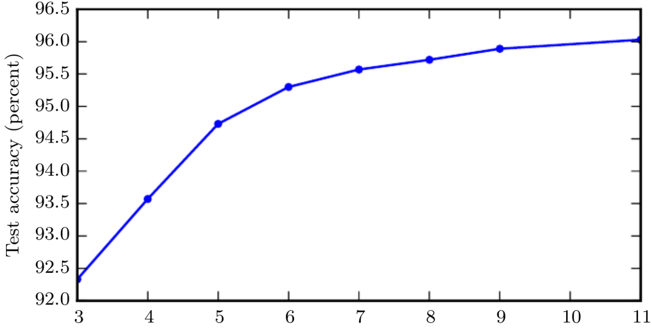
- The theorem guarantees that a large enough NN can **represent** any function, but we are not given a training algorithm that can **learn** any function
- Training can fail for two reasons
  1. we completely fail to compute the parameters corresponding to the desired function
  2. as a result of overfitting, we learn the wrong function
- Moreover, the theorem does not say how large the hidden layer has to be!

Deep Neural Networks (DNN)

- In practice, better results can be achieved with more layers (i.e., deeper networks) rather than larger layers
- Traditional distinction between shallow and deep NNs
- No standard definition of “deep”
  - Popular one: “NNs with more than 1 hidden layer are deep”
  - Sometimes: “NNs with more than 2 hidden layers are deep”
- Just pick the definition you prefer…

Impact of Depth: Example

Multidigit number transcription from Street View address photographs

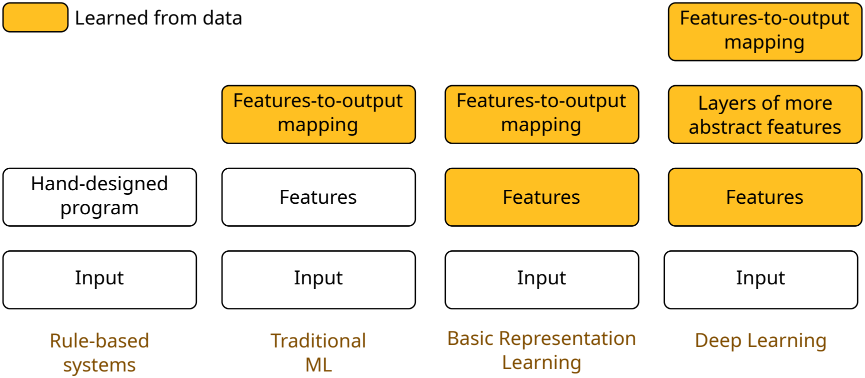


Goodfellow et al., "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks", 2014

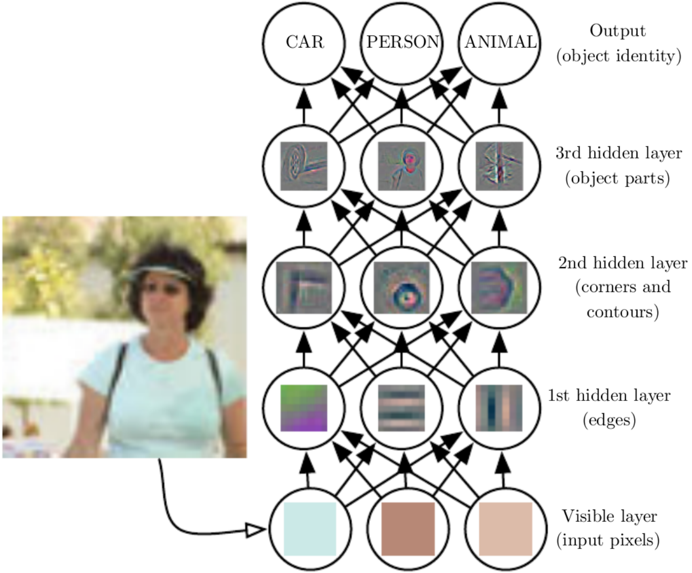
Deep Learning

- Deep learning (DL) is a collection of ML methods based on DNNs and representation learning, which can be adopted for (semi-)supervised, unsupervised and reinforcement learning tasks
- Not just a matter of having “more layers”
- Traditional ML: given a representation of the input data (i.e., features), learn a mapping from representation to output
- DL aims to learn both the representation and the mapping!
- Multiple layers $\rightarrow$ learning multiple levels of composition
  - discovering representations of the input expressed in terms of other, simpler representations

Comparison of AI Approaches



Example


CAR
PERSON
ANIMAL
Output
(object identity)
3rd hidden layer
(object parts)
2nd hidden layer
(corners and
contours)
1st hidden layer
(edges)
Visible layer
(input pixels)

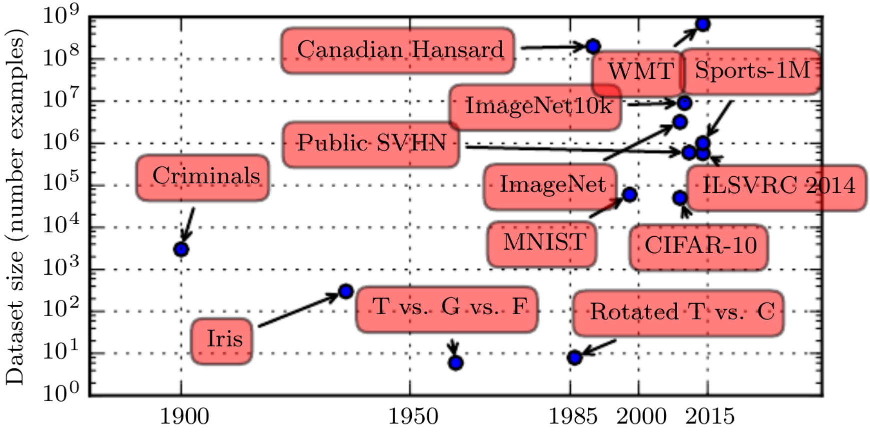
Back to History

- ANNs had 3 waves of popularity
  - 1950s-1960s: Perceptron, only linear models
  - 1980s-1990s: MLPs, backpropagation, difficulties to train large models
  - 2006–today: deep learning
- Current renaissance began in 2006, when Hinton showed that a DNN could be efficiently trained to outperform a SVM-based solution on the MNIST benchmark.
  - *Hinton et al., "A Fast Learning Algorithm for Deep Belief Nets"*
- In general, DL "revolution" enabled by multiple factors:
  - more data ("Big Data")
  - more and better hardware for parallel computing (e.g., GPGPUs)
  - new/improved software libraries

Increasing Dataset Sizes

- First DL applications date back to 1990s, but more as an art rather than an accessible technology (due to very difficult training)
- Nearly identical learning algorithms today reach human performance on complex tasks (though the models have undergone changes that simplify the training of very deep architectures)
- What else has changed? We have much more data for training!
  - …and the computational resources to store and process those data

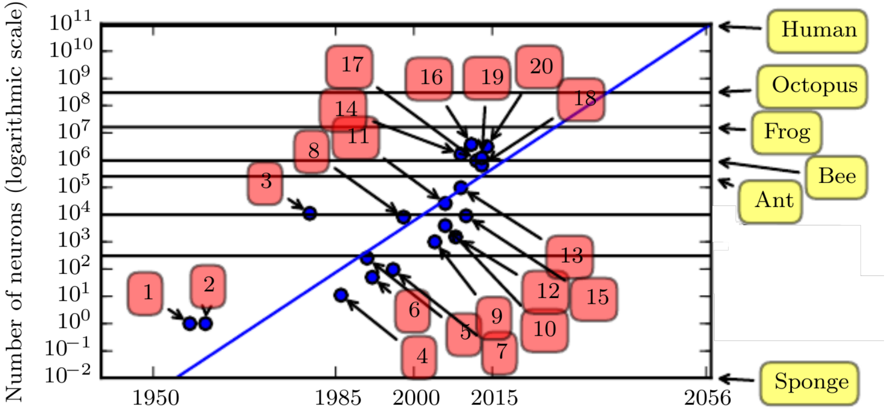
Example: Popular Datasets



Increasing Model Sizes

- Besides handling more data, we have the resources to run much larger models today
  - mainly faster CPUs, GPGPUs, better software infrastructures for parallel and distributed computing
  - …and enough data to train them
- Unless new technologies enable significantly faster scaling, ANNs will reach the same number of neurons of the human brain not earlier than 2050

Example: Neural Network Size



1: Perceptron (1962) 4: Backpropagation NN (1986)
10: Deep Belief Network by Hinton (2006)
20: GoogLeNet (2014)

Increasing Impact

- The spectacular success of DL in several domains has further dramatically increased the interest of researchers and industries in the last decades
  - e.g, object recognition, speech recognition, image/video generation, ...
- Since 2015, DL has been also successfully and widely applied to reinforcement learning tasks

Recommended Readings

- Goodfellow et al., Chapter 6 (§6.1–§6.4)
- Dive into DL (`d2l.ai`): §5.1