Machine Learning

Università Roma Tre
Dipartimento di Ingegneria
Anno Accademico 2022 - 2023

Deep Learning: Introduzione

Sommario

- Informazioni sul corso
  - Programma, testi consigliati, precondizioni
- Software tools
- Cos'è il deep learning
  - Representation learning
  - Autoencoders
  - Factors of variation
- Approcci di IA ed evoluzione delle architetture
- Curse dimensionality
- Local constancy & smoothness regularization
- Libreria D2L

Il corso

- L'obiettivo del **corso di Deep Learning (DL)** di 6 CFU è fornire competenze avanzate e specifiche nell'ambito delle architetture di reti neurali Deep.
- Il corso è strutturato in una parte teorica e *metodologica* sui concetti fondamentali, e da una *attività di programmazione* in cui tali concetti sono applicati nella risoluzione di problemi mediante recenti framework di sviluppo (Keras & PyTorch).
- Al termine del corso lo studente sarà in grado di:
  - addestrare e ottimizzare in maniera adeguata reti neurali Deep;
  - saper distinguere tra diverse soluzioni, e saper selezionare e personalizzare le architetture di reti più efficaci da utilizzare in ambiti applicativi reali, supervised, unsupervised o seguendo un approccio basato su un apprendimento per rinforzo.
- Il corso prevede lo svolgimento di progetti.

Il programma

1. **Introduzione al Corso**

   - Introduzione al deep learning.
   - Elementi di Algebra lineare e di calcolo differenziale
   - Algoritmi specifici per l'ottimizzazione e la regolarizzazione
   - Richiami di feature engineering: One-hot encoding, Binning, Normalization,
   - Valutazione delle performance dei modelli di DL
   - Tuning degli iperparametri
   - Tecniche di regolarizzazione (es. Dropout)

2. **Convolutional Neural Networks (CNN)**

   - Concetti base: Local receptive fields, shared weights, pooling
   - Architettura LeNet
   - Very deep convolutional networks per la Computer vision
   - Architetture CNN, es. AlexNet, Residual Networks, VGG

3. **Reti Ricorrenti (RNN)**

   - Cella RNN e backpropagation through time (BPTT)
   - Bidirectional RNNs e Stateful RNNs
   - Architetture Encoder-Decoder
   - Architettura Transformer
   - Attention mechanism

4. **Autoencoders**

   - Vanilla autoencoders
   - Sparse autoencoder
   - Denoising autoencoders
   - Stacked autoencoder

5. **Generative Adversarial Networks (GAN)**

   - Architetture GAN: SRGAN, CycleGAN, InfoGAN
   - Applicazioni con le GAN

6. **Word embeddings**

   - Static embeddings: Word2Vec, GloVe
   - Neural embeddings: Item2Vec, node2vec
   - Language model-based embeddings, es. BERT
   - Creazione di word embeddings

7. **Deep Reinforcement Learning**

   - OpenAI Gym
   - Deep Q-Networks

8. **Casi di Studio e Progetti**

Testi consigliati e altri riferimenti

- • I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning", MIT Press, 2016.
- • A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into Deep Learning", 2020 (free online).
- • A. Geron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”, O'Reilly Media, Inc, USA, 2019.
- • M. Nielsen, "Neural Networks and Deep Learning", 2019 (free online).

- Altri riferimenti a codice, tutorial, e altre fonti saranno dati durante il corso.

Precondizioni

- Le seguenti lezioni del corso di Machine Learning sono requisiti per il corso di DL:
  - *Introduzione alla Regressione*
  - *La Valutazione nella Regressione*
  - *Overfitting, Cross Validation*
  - *Introduzione alle Reti Neurali Artificiali (es. algoritmo di backpropagation)*

- Sebbene alcuni dei concetti saranno ripresi per introdurre i formalismi necessari al resto del corso.

Software Tools

- Docker
  - https://www.docker.com/community-edition#/download

- Jupyter Notebook Scientific Python Stack + Tensorflow + Tensorboard
  - https://github.com/lspvic/jupyter_tensorboard

```
docker pull lspvic/tensorboard-notebook
docker run -it --rm -p 8888:8888 lspvic/tensorboard-notebook
```

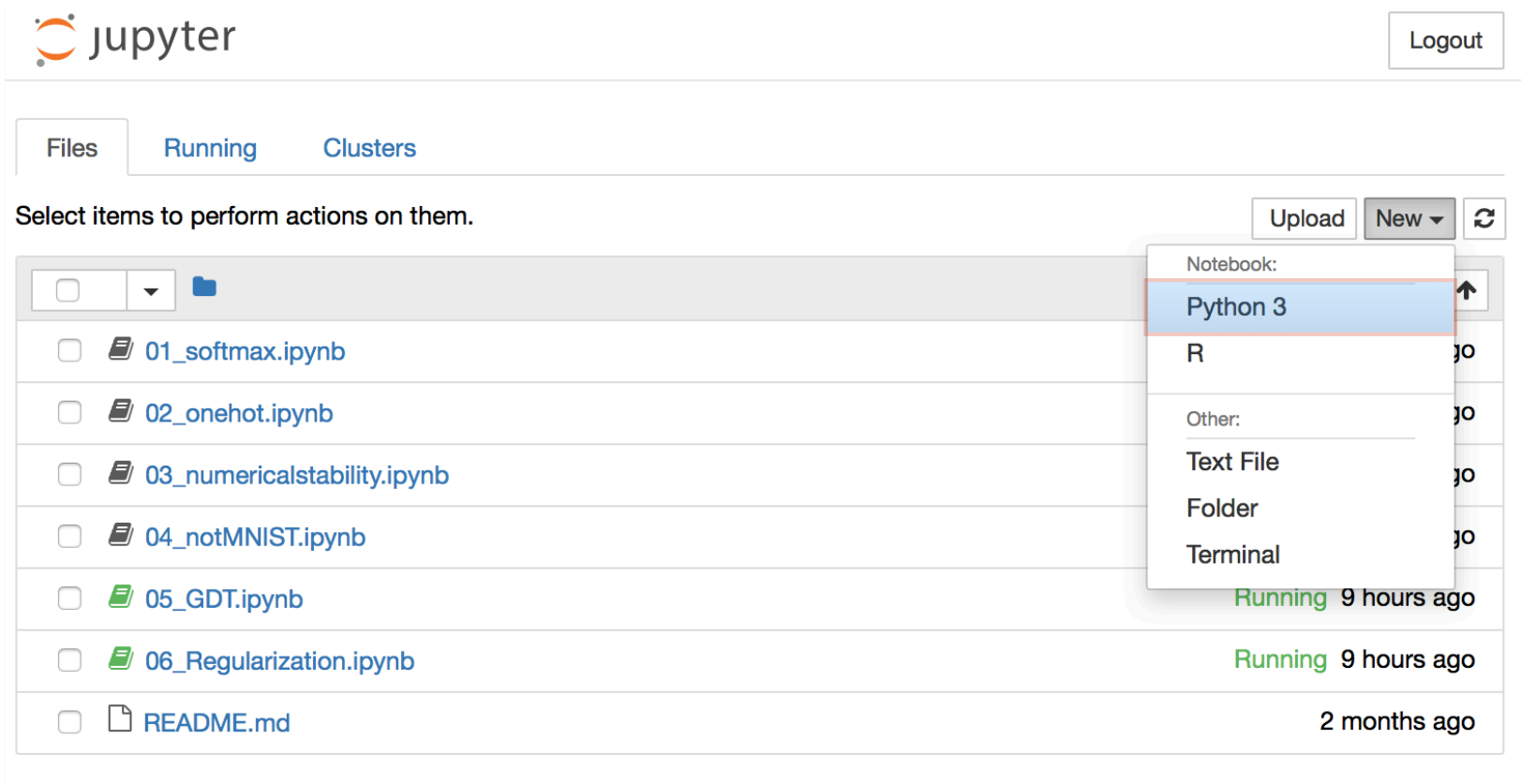
- Docker Engine Utility for NVIDIA GPUs
  - https://github.com/NVIDIA/nvidia-docker

- Anaconda + Tensorflow
  - https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

Jupiter

```
[C 19:12:58.853 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=afb2c2ddcde0e70e27382a243e3d3221b5ea38c8ed3e2be5
```



https://jupyter.readthedocs.io/en/latest/

Google Colaboratory (o Colab)

- Servizio di calcolo online basato su Nvidia Tesla T4 GPUs
  - 12 GB of RAM
  - fino a 12 ore di seguito
- Supporto multi-ambiente: TensorFlow, Keras, PyTorch, e OpenCV.
- Molti dataset disponibili nell'ambiente
  - https://www.tensorflow.org/datasets/catalog/overview
- Interfaccia Jupyter ben nota.
- Default: Runtime Python 3 e nessun acceleratore hardware.
  - Menu Runtime -> Change runtime type
- Possibilità di trasferire l'esecuzione in locale (per elaborazioni molto lunghe)
  - https://research.google.com/colaboratory/local-runtimes.html

Google Colaboratory (o Colab)

Acceleratore: GPU

```
!nvidia-smi
```

```
Mon Oct 26 12:23:58 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.23.05    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Google Colaboratory (o Colab)

```
!lscpu
```

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    2
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU @ 2.30GHz
Stepping:              0
CPU MHz:               2300.000
BogoMIPS:              4600.00
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              46080K
NUMA node0 CPU(s):     0,1
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
```

Google Colaboratory (o Colab)

GPUs includes Tesla **P100** (used in Colab), Tesla **V100** (equipped in Amazon EC2 P3 instance), and Tesla **T4** (equipped in Amazon EC2 G4 instance). **TPUs** are tensor processing units developed by Google to accelerate operations on a Tensorflow Graph. Each TPU packs up to 180 teraflops of floating-point performance and 64 GB of high-bandwidth memory onto a single board.

Hardware	Intel E5-2686 v4	Tesla P100	Tesla V100	Tesla T4
Clock rate (GHz)	**3**	1.48	1.53	1.59
# cores	16	56	**80**	40
# FP64 AUs per core	4	**32**	**32**	x
# FP32 AUs per core	8	**64**	**64**	**64**
# FP16 AUs per core	x	x*	**8**	**8**
cache per core (KB)	**320**	64	128	64
shared cache (MB)	**45**	4	6	6
Memory (GB)	**240**	16	16	16
Max memory bandwidth (GB/sec)	72	732	**900**	300
FP64 TFLOPS	0.38	4.7	**7.8**	x
FP32 TFLOPS	0.77	9.3	**15.7**	8.1
FP16 TFLOPS	x	18.7	**125.3**	65
:label: `tab_cpu_gpu_compare`				

Cos'è il Deep Learning

- Il *machine learning* riguarda tecnologie capaci di acquisire conoscenza relativamente all'ambiente di interesse allo scopo di risolvere problemi in modo automatizzato, cioè senza l'intervento dell'utente.
- Per problemi complessi occorre rappresentare la conoscenza come concetti su vari livelli di astrazione, creando dipendenze tra gli stessi, in modo simile a come avviene nella mente umana.
- Da queste strutture deriva il termine **deep learning**.
- Storicamente i computer sono stati impiegati per rappresentare conoscenza formale (es. regole per giocare a scacchi) su cui implementare meccanismi di *reasoning* (es. regole logiche) mentre è stato più difficile rappresentare la conoscenza informale.
  - Esempio: Cyc inference engine e "Fred shaving in the morning"

Representation learning (1)

- Successivamente sono state introdotte tecniche di machine learning per acquisire la conoscenza (*know-how*) estraendo *patterns* dai dati in modo automatico, es. logistic regression e naive Bayes.
- Le prestazioni di tali approcci dipendono dalla scelta con cui i dati sono rappresentati. È importanti scegliere le informazioni più rilevanti (*features*) per il task che si intende risolvere (approccio *hand-designed*).
  - LXI + XCIX = ?
- Per alcuni task definire una rappresentazione dei task è arduo.
  - Es. identificare un auto potrebbe ridursi al task di riconoscere le ruote; come puoi farlo a partire da una rappresentazione a pixel?
- A differenza del hand-designed, il **representation learning** introduce un sotto-task nel processo di ML che mira a riconoscere le feature più rilevanti in modo automatico.

Representation learning (2)

- Identificare le features in modo automatico garantisce vantaggi:
  - L'approccio hand-designed è lungo e richiede risorse
  - Si può facilmente adattare l'addestramento a nuovi tasks

Autoencoders

- Gli **autoencoders** sono composti da un *encoder* e un *decoder*. Il primo converte l'input in una rappresentazione compatta, cioè con dimensionalità ridotta,, il decoder mira a ricostruire l'input originale da tale rappresentazione.
- L'addestramento degli autoencoders crea uno *spazio* che mira a rappresentare solo le features salienti necessarie per identificare una certa istanza, tralasciando informazioni non utili.
- Nel corso vedremo diversi tipi di autoencoders. Le *Generative Adversarial Network (GAN)* impiegano tali tecnologie.

Factors of variation

- Le features identificate con un approccio hand-designed, oppure riconosciute in modo automatico durante il learning, devono saper distinguere i *fattori di variazione*.
- Sono spesso considerati degli elementi astratti (non misurabili) che influenzano il modo in cui le istanze vengono viste dagli approcci di ML. Se identificati ci permettono di capire meglio la grande variabilità di istanze in certi domini.
- Ad esempio, età, sesso, un certo accento possono influenzare le parole pronunciate da una certa persona in un task di speech-recognition. Osservando un automobile, la posizione, il colore, l'angolo di incidenza dei raggi solari sono altri tipici fattori per l'analisi di una immagine.
- Se riusciamo a riconoscerli e ignorarli durante il processamento saremmo in grado di semplificare molti task di ML.

Esercizio

- Prova a identificare un ulteriore task adatto ad un approccio di ML, ed elenca qualche fattore di variazione.

Deep Learning

- Il Deep learning segue un approccio di *representation learning*, dove certe rappresentazioni sono espresse mediante altre più semplici, es., l'immagine di un uomo viene composta da angoli e contorni, che a sua volta sono rappresentati con piccoli segmenti.
- Un esempio di una architettura Deep, il *multilayer perception:*

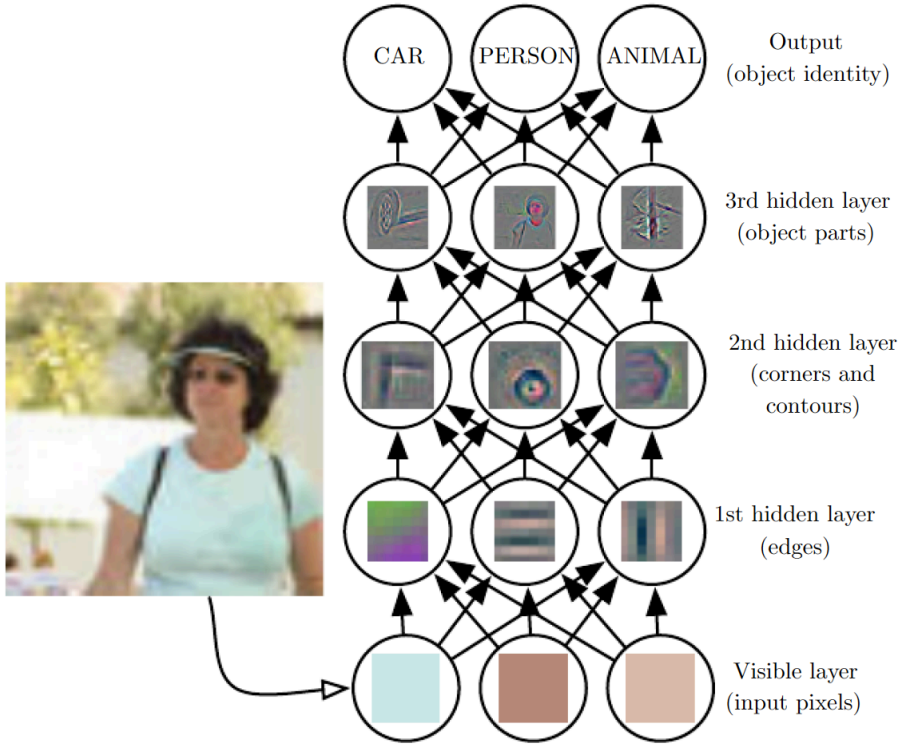
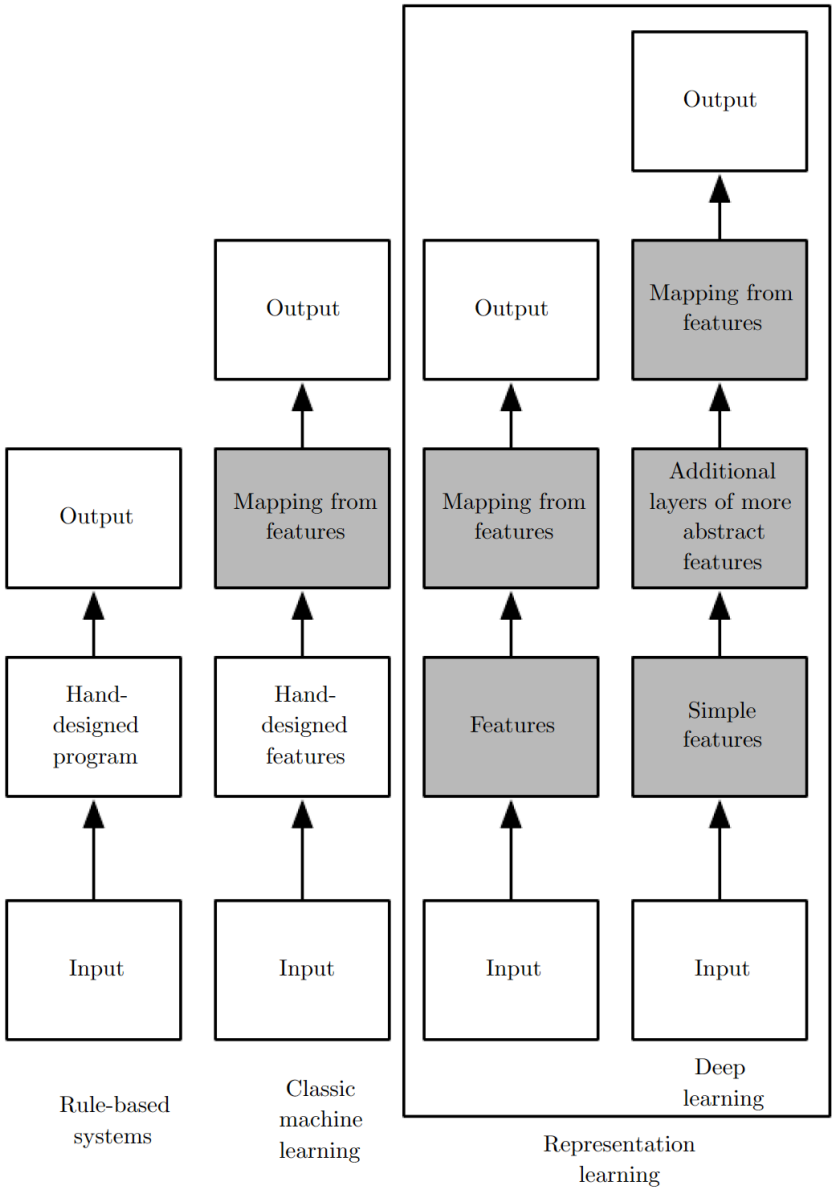


Immagine tratta da Zeiler and Fergus (2014).

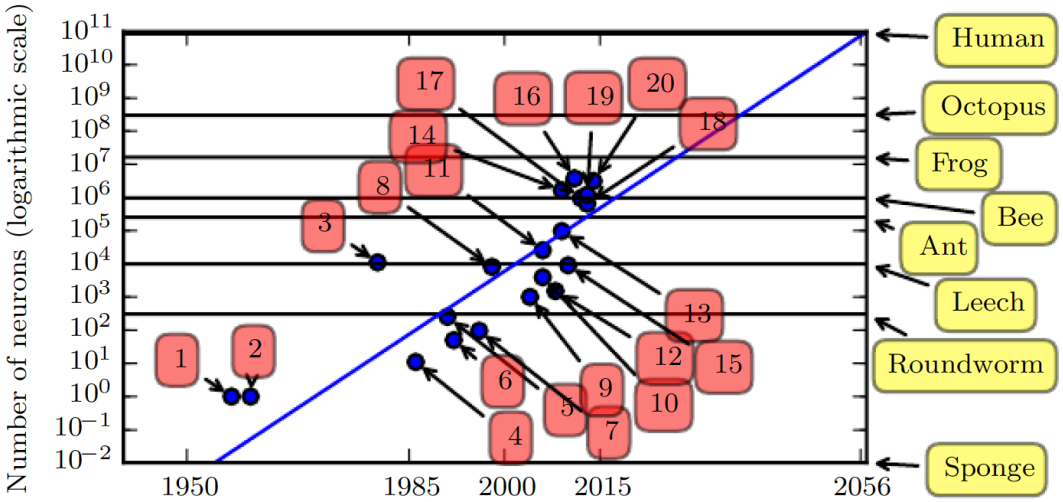
Schema semplificato approcci di IA



- I box scuri includono fasi esplicite di apprendimento.

Evoluzione delle architetture

- Ogni 2,4 anni il numero di neuroni nascosti è all'incirca raddoppiato.



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)

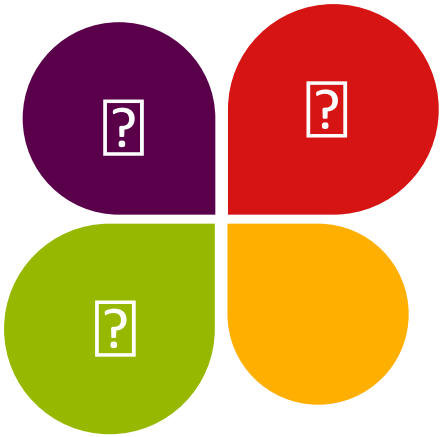
I.A. & Big Data: il contesto attuale

Large amount of Human-generated content

Posizione GPS, “mi piace”, precedenti acquisiti, immagini sui social.

Infrastructures

Cloud computing, GPU-enabled infrastructure



AI-enabled frameworks

Recommender systems, Speech and Text processing, Video and Image analysis

Oligopoly on data

Poche imprese possiedono grandi quantità di dati sull’utente.

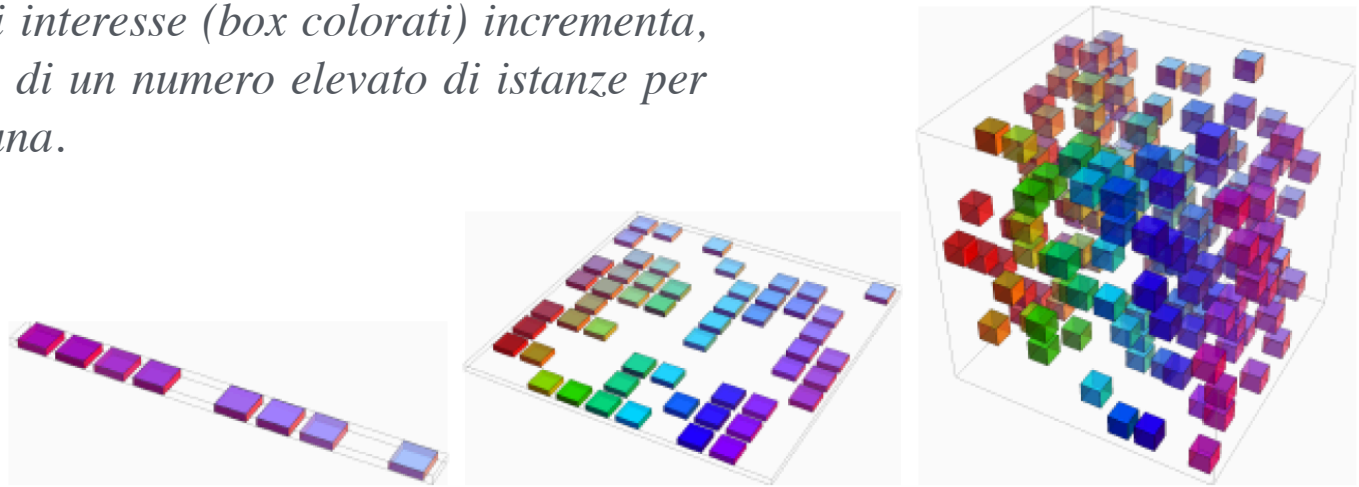
EU GDPR
personal information as economic asset

AI-based interpretation of content
by natural language processing, personality traits, object recognition, etc.

Curse of dimensionality

- Nei corsi di I.A. e M.L. sono stati descritti numerosi algoritmi che si adattano facilmente i vari task. Ma solo pochi riescono ad affrontare problemi centrali come riconoscere il parlato od oggetti arbitrari.
- Tali task causano il cosiddetto *curse of dimensionality*: il numero di potenziali configurazioni di variabili di ingresso cresce in modo esponenziali col numero di variabili considerate.
  - Ne segue: istanze di training << # potenziali configurazioni

Incrementando il numero di dimensioni (da 1d a 3d) il numero di regioni di interesse (box colorati) incrementa, e abbiamo necessità di un numero elevato di istanze per caratterizzarne ognuna.



Local constancy & Smoothness regularization

- Imponiamo che la distribuzione di probabilità a priori, che influenza i parametri ma anche la funzione che stiamo cercando di stimare, sia **smoothness prior** o **local constancy prior**.
- Sotto questa assunzione, se l'output di una funzione è OK per una certa istanza *x*, allora l'output è buono anche per istanze vicine ad *x*:

$$f^*(\boldsymbol{x}) \approx f^*(\boldsymbol{x} + \epsilon)$$

- Esempio: algoritmo di *k*-nearest neighbors.
- Per dimensionalità elevate, una funzione smooth potrebbe cambiare (in modo smooth) in modo diverso a seconda della dimensione. Occorrono molte istanze di training per caratterizzarle.
- Il deep learning introduce dipendenze tra le regioni di interesse (cioè nelle distribuzioni dei dati) per ridurre il numero di istanze necessarie.

La libreria D2L

- Durante il corso faremo uso di una libreria Python di supporto chiamata **D2L.ai**
- La libreria d2l mette a disposizione alcune funzionalità per rendere il codice più interpretabile e compatto.
- Vediamone alcuni esempi di impiego:
  - 01-d2l_3.2.ipynb
  - 02-d2l_regressione_3.3.ipynb