Introduzione al Deep Learning

Giovanni Busonera
CRS4
17/05/2019

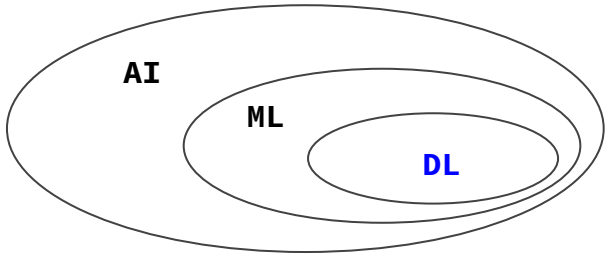
Deep Learning: Outline

---

1. *Definizione e contesto*
2. *Perceptron*
3. *Reti neurali*
4. *Introduzione a Keras*

Deep Learning:

Il Deep Learning (DL) fa parte del Machine Learning (ML) e dell'Intelligenza Artificiale (AI). Si basa su diversi livelli di rappresentazione corrispondenti a gerarchie di concetti in cui quelli di alto livello sono definiti sulla base di quelli di basso livello mediante relazioni non lineari.

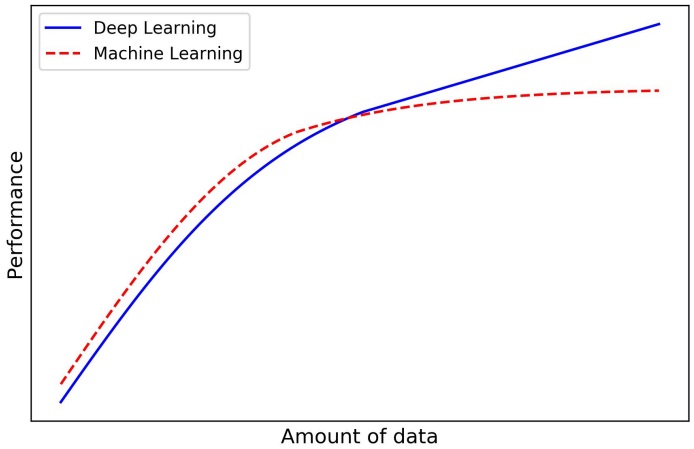


“Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.”

LeCun, Yann Bengio, Yoshua Hinton, Geoffrey - Deep learning - Nature - 2015/05/27/online

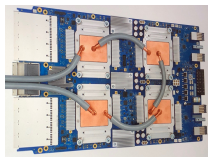
Deep Learning: alcune differenze con il ML

Prestazioni vs quantità di dati



Hardware

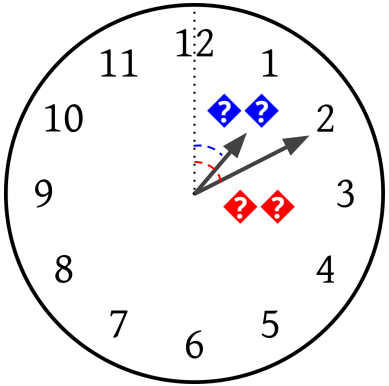
GPU



TPU

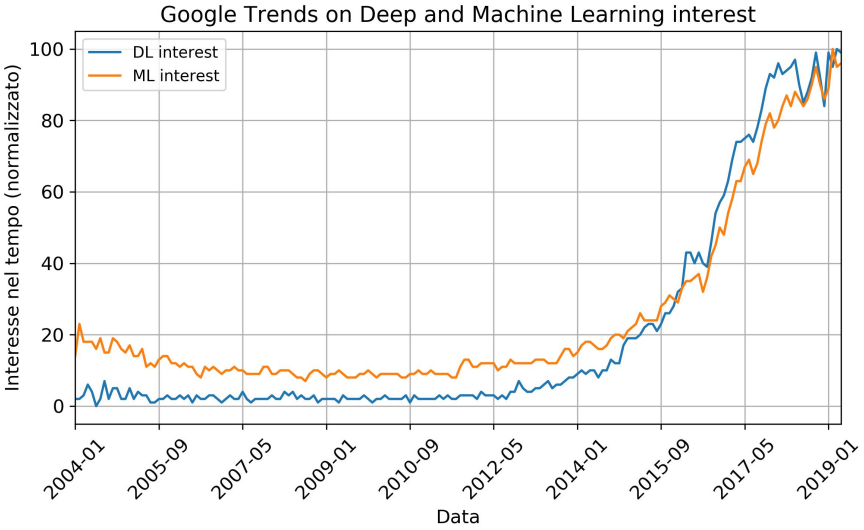
FPGA

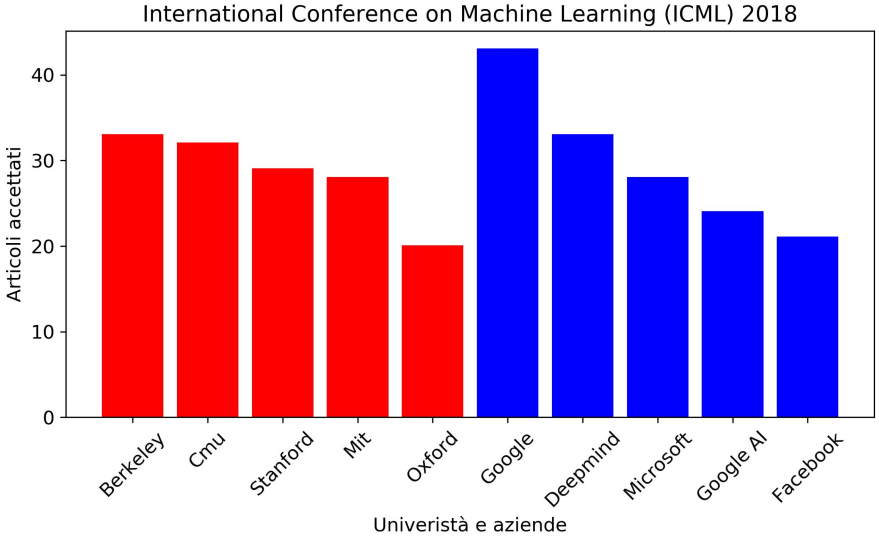
Feature extraction



	Input	Target
ML	α, β	1:10AM
DL	whole image	1:10AM

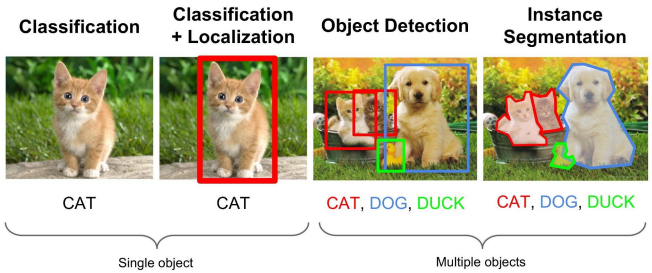
Deep Learning: Trend





Applicazioni

Visual recognition



Search engine

bing

Baidu 百度

Google

Speech recognition

Google Assistant

Apple Siri

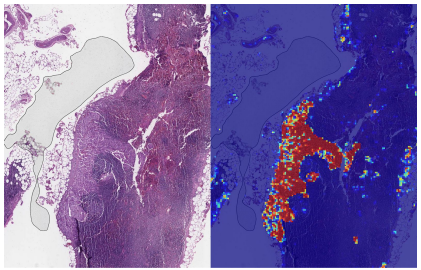
Amazon Alexa

Self driving car

WAYMO

TESLA

Healthcare



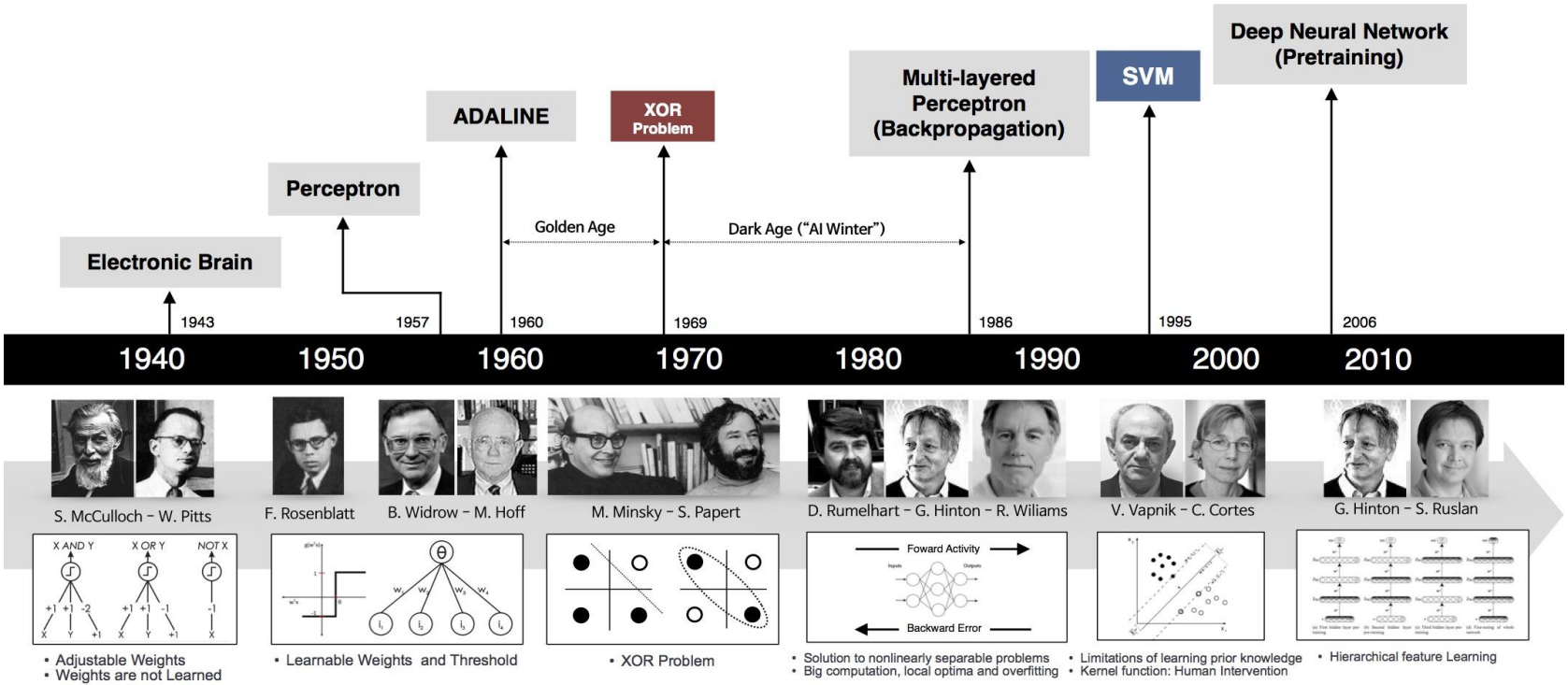
Google AI for cancer detection

Multilingual translation

Google Translate

DeepL

Cenni storici: dal Perceptron al Deep Learning

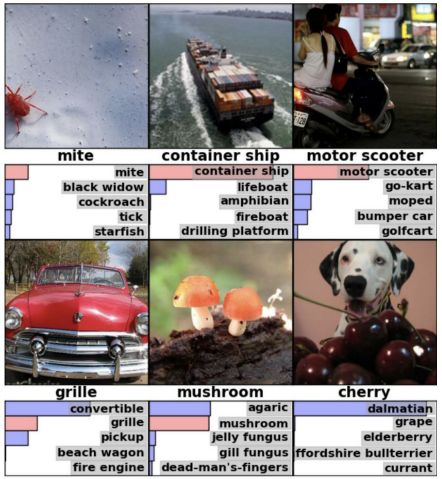


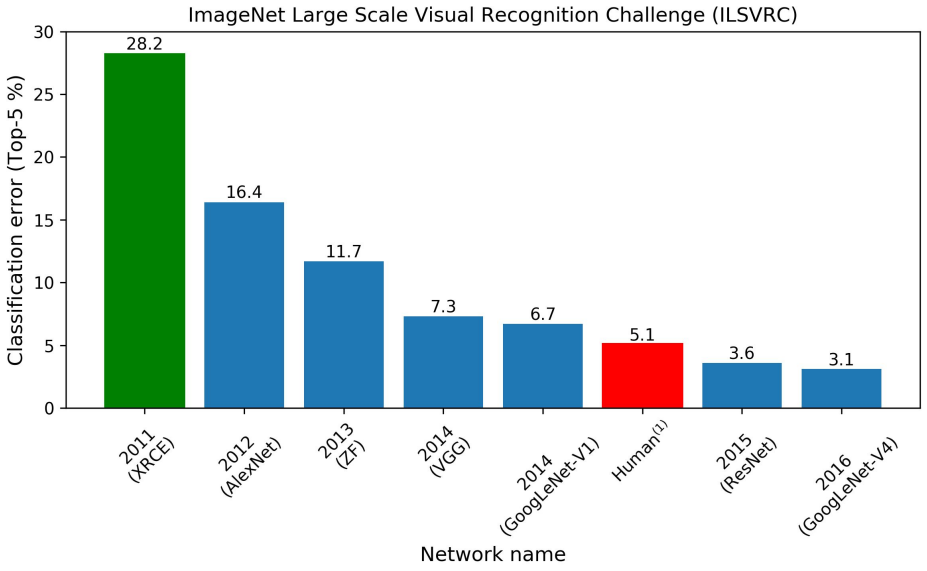
Nota: Immagine tratta da https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

ImageNet e ILSVRC

ImageNet Challenge

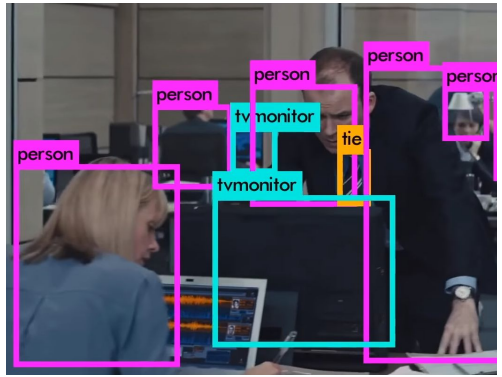
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.





(1) Russakovsky O. et Alter (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), 211–252

Real time object detection: Yolo Net v2

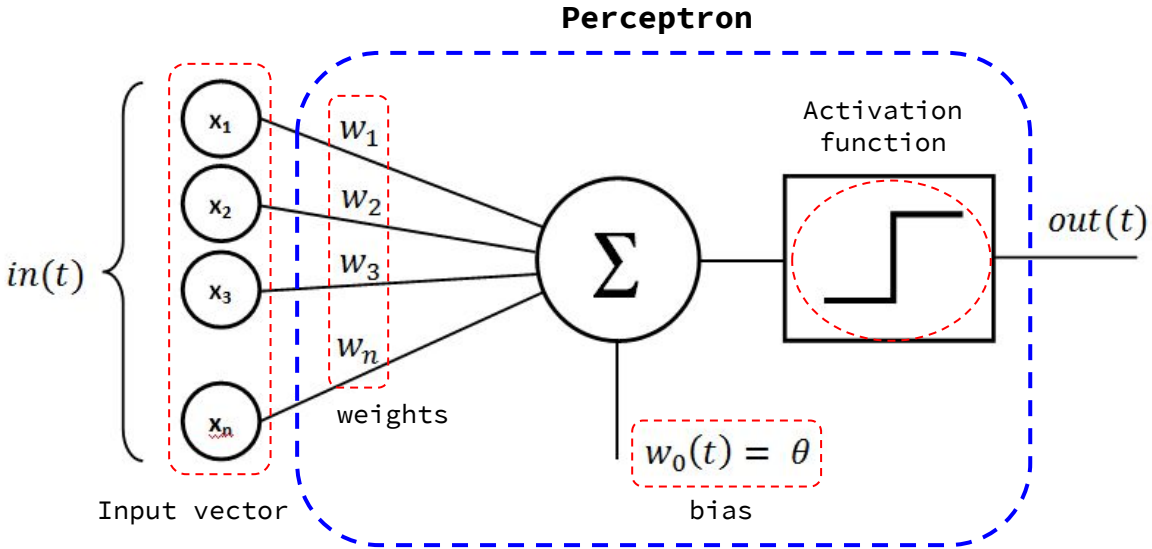


URL: https://pjreddie.com/darknet/yolov2/

Perceptron

Origini delle ANN: il Perceptron

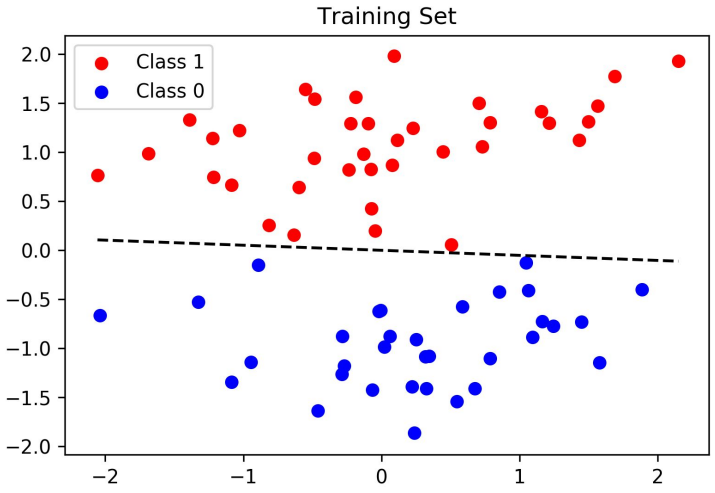
- È un classificatore lineare binario proposto da Frank Rosenblatt nel 1958 e ispirato dai precedenti lavori di Warren McCulloch e Walter Pitts.

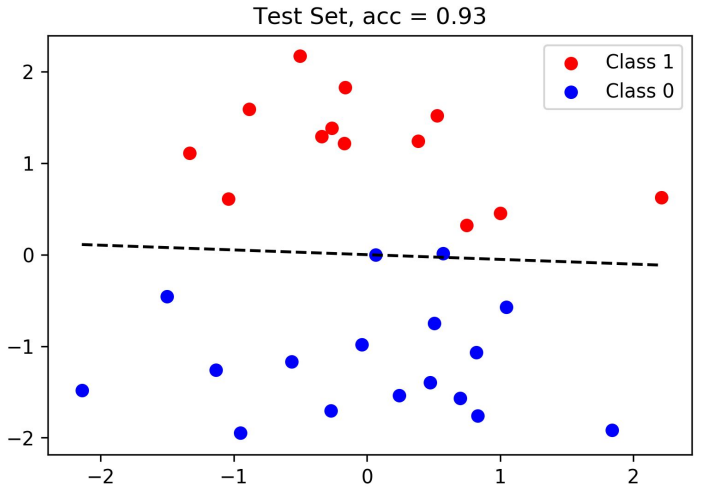


$$\begin{cases} 1, & \text{if } \sum_i x_i w_i + \theta > 0. \\ 0, & \text{if } \sum_i x_i w_i + \theta \leq 0. \end{cases}$$

Origini delle ANN: il Perceptron

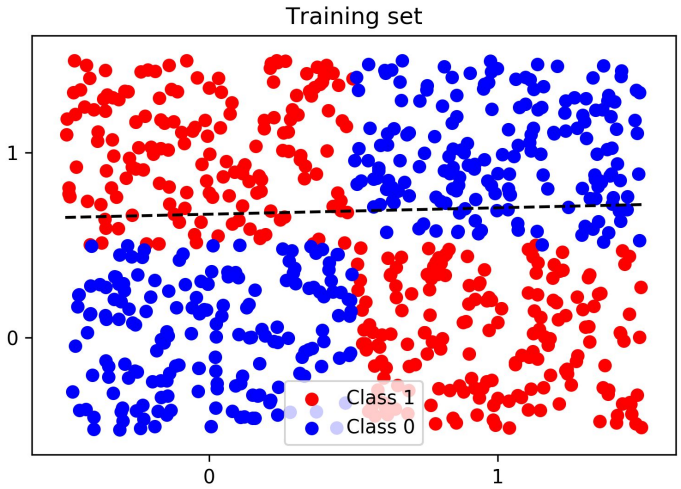
- L’algoritmo converge solo se l’insieme dei dati è linearmente separabile.

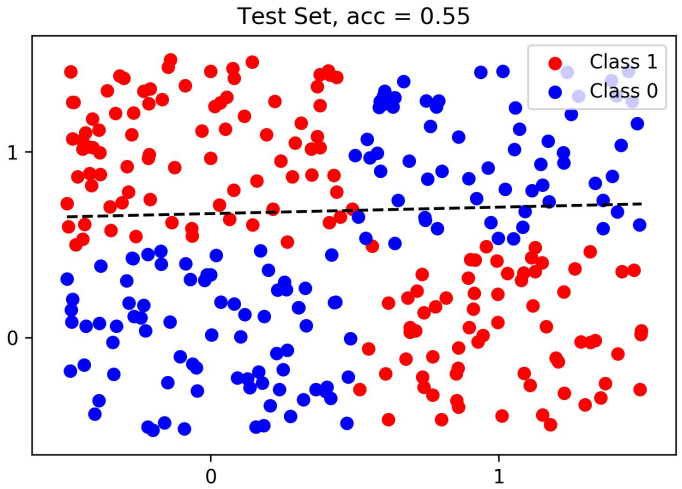




Origini delle ANN: il Perceptron e il problema dello XOR

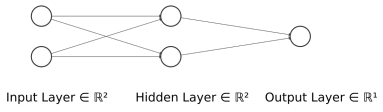
- Per dati non separabili linearmente l'algoritmo da risultati insoddisfacenti

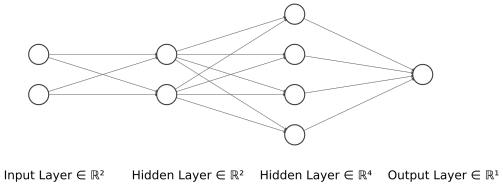


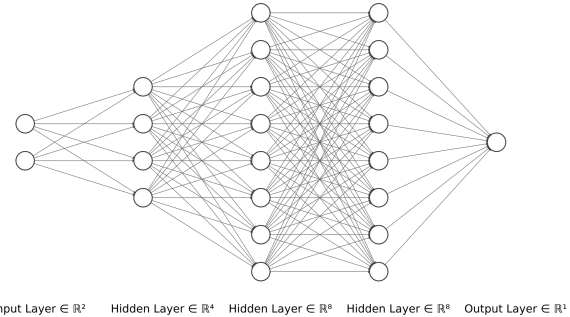


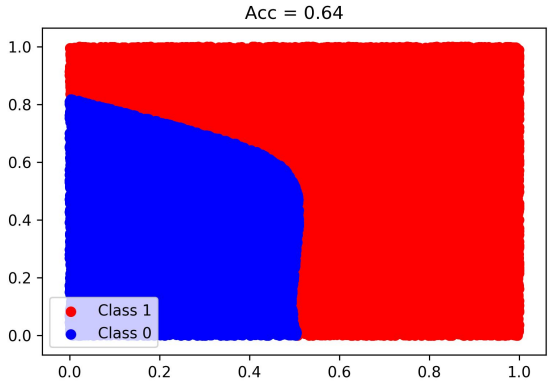
Origini delle ANN: La soluzione dello XOR → MLP

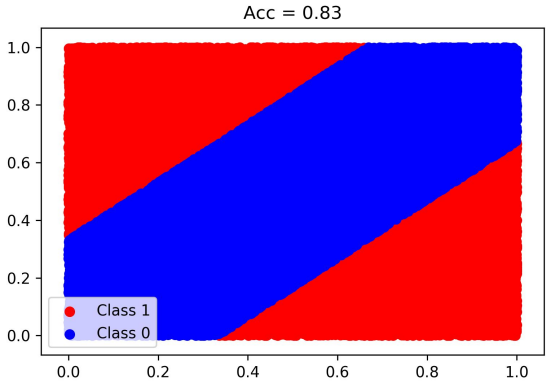
- Multilayer perceptron(MLP)

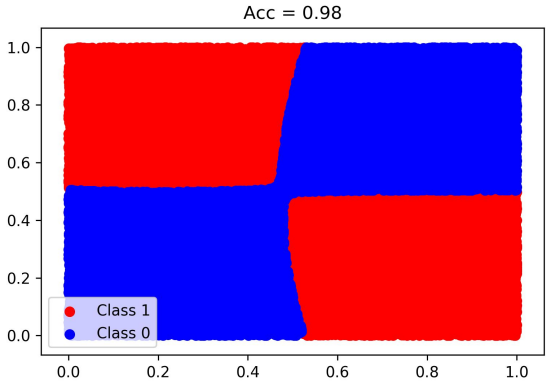












Origini delle ANN: L’apprendimento del MLP

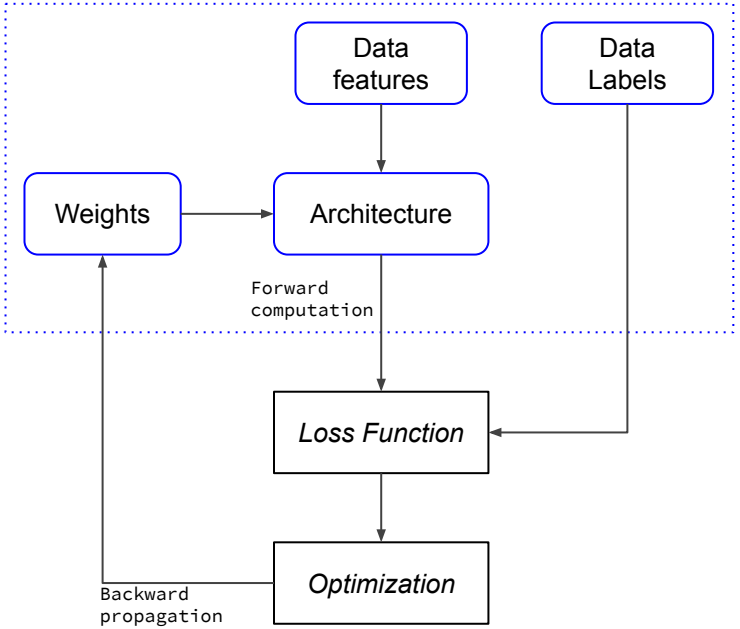
- Aumentano i parametri e serve un metodo per determinarli in modo automatico (Gradient based optimization e backpropagation)
- La funzione di attivazione deve avere delle caratteristiche adatte (derivabile) per l’applicazione degli algoritmi di training

Artificial Neural Networks

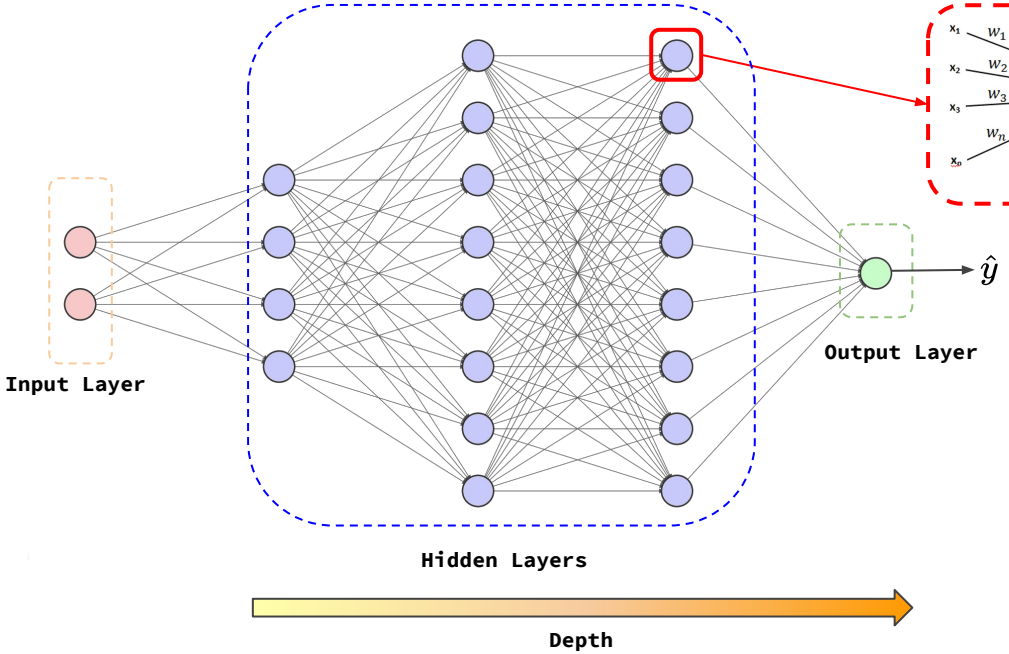
Anatomia di una rete neurale

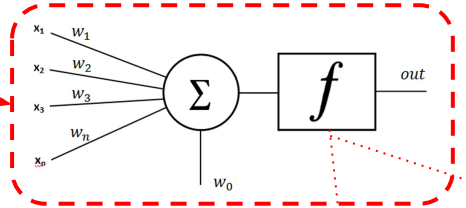
---

1. *Architettura* (forward computation)
2. *Loss Function*
3. *Gradient based optimization* (backward propagation)
4. *Misura della prestazione* (es: accuracy)

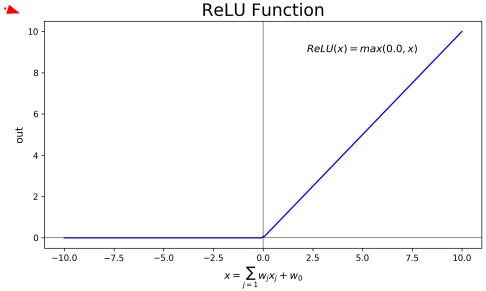


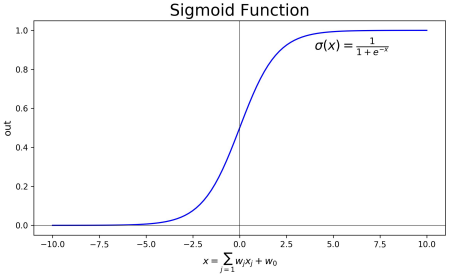
ANN: Architettura e Forward computation





$$f(\sum_{j=1} w_j x_j + w_0)$$





Loss function

- Misura la “distanza” tra la predizione della rete neurale e l’output di riferimento dei samples. Ad esempio la Mean Squared Error (MSE) è definita come:

$$L(w) \equiv \frac{1}{n} \sum_i \left\| y_i - \hat{y}_i \right\|^2$$

- Esistono varie tipologie di Loss function (MSE, Binary crossentropy, Categorical crossentropy, etc…)
- La scelta della Loss function dipende dall’applicazione

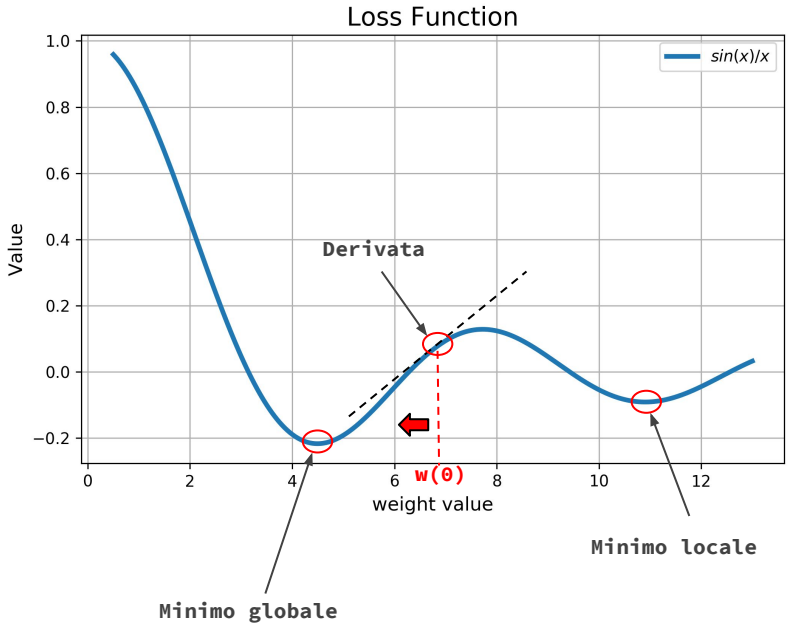
Optimization: Gradient Descent (GD)

- Si sceglie un punto a caso w(0)
- Si calcola la derivata
- Ci si muove in direzione opposta alla derivata

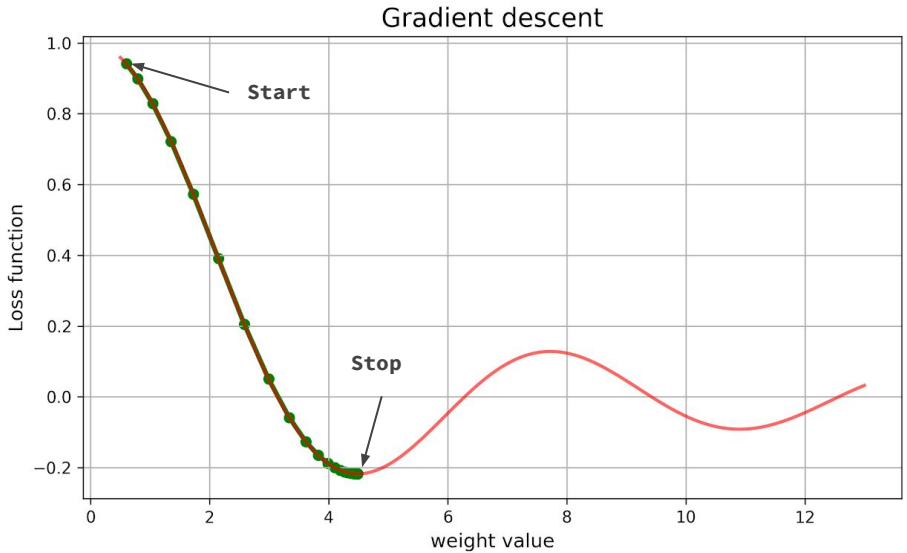
$$w(i+1) = w(i) - \eta \frac{dL}{dw}$$

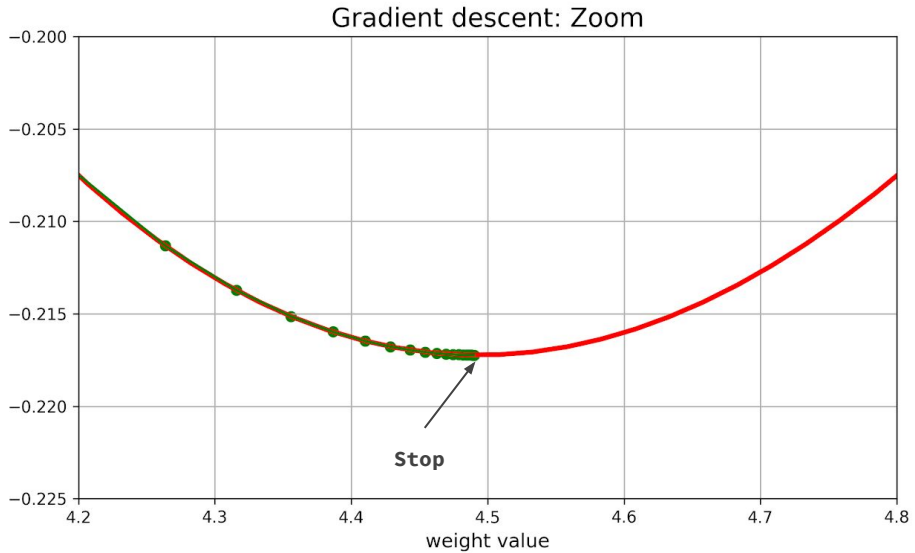
Learning rate

- Appena si raggiunge un minimo ci si ferma

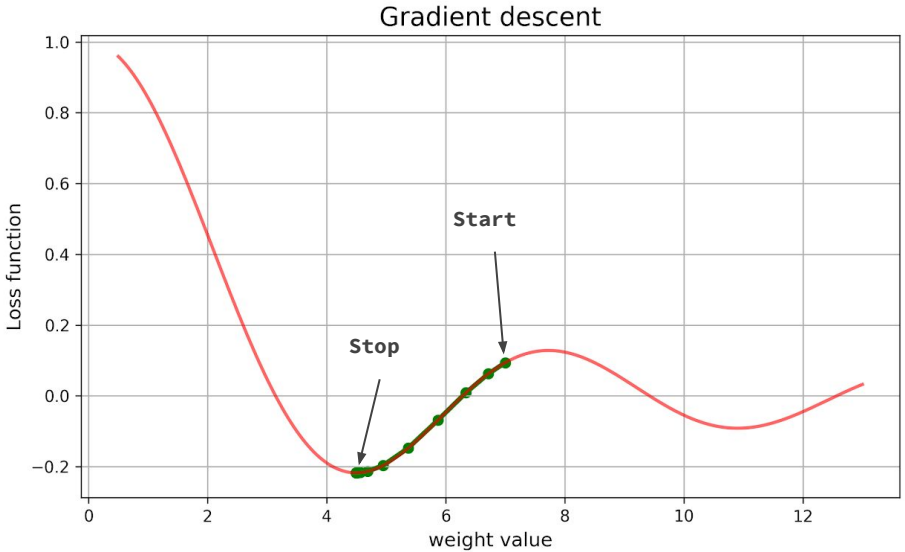


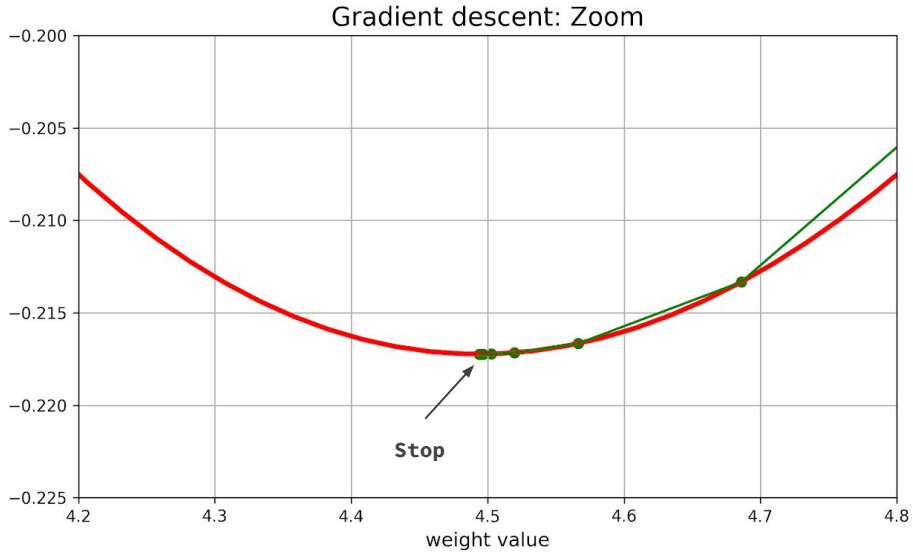
Optimization: Gradient descent



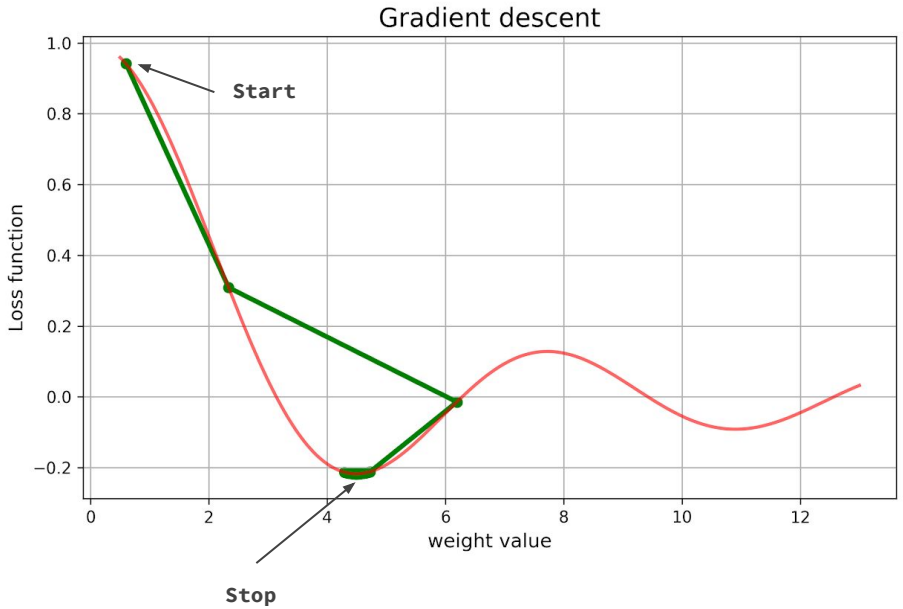


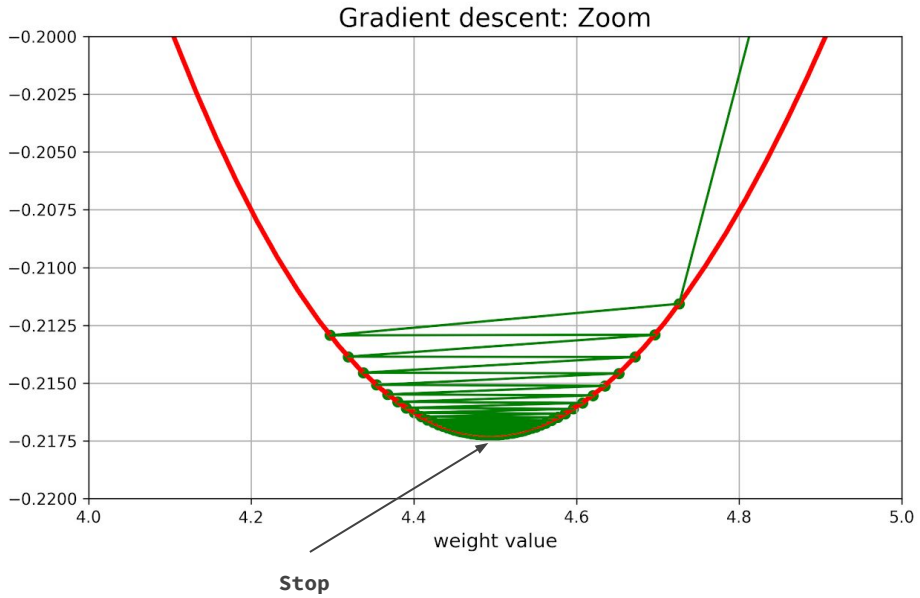
Optimization: Gradient descent



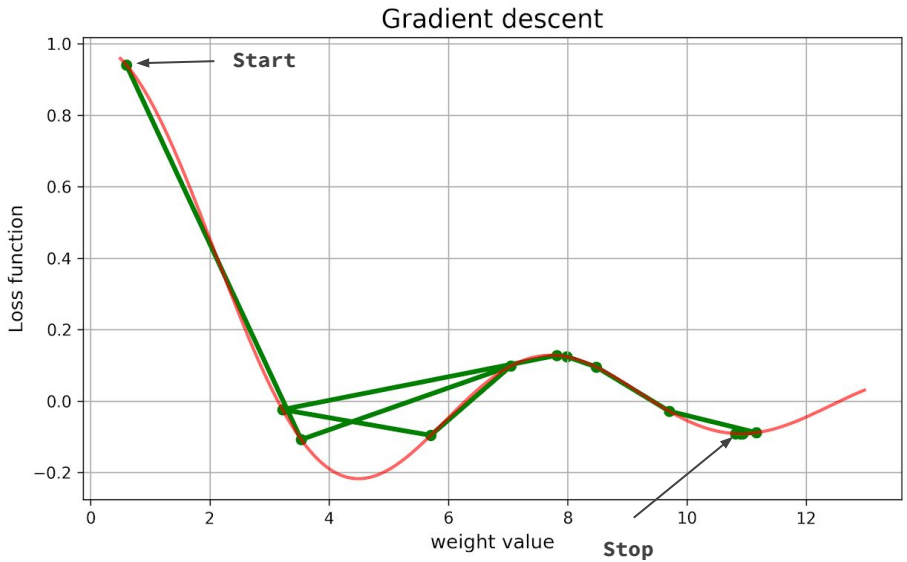


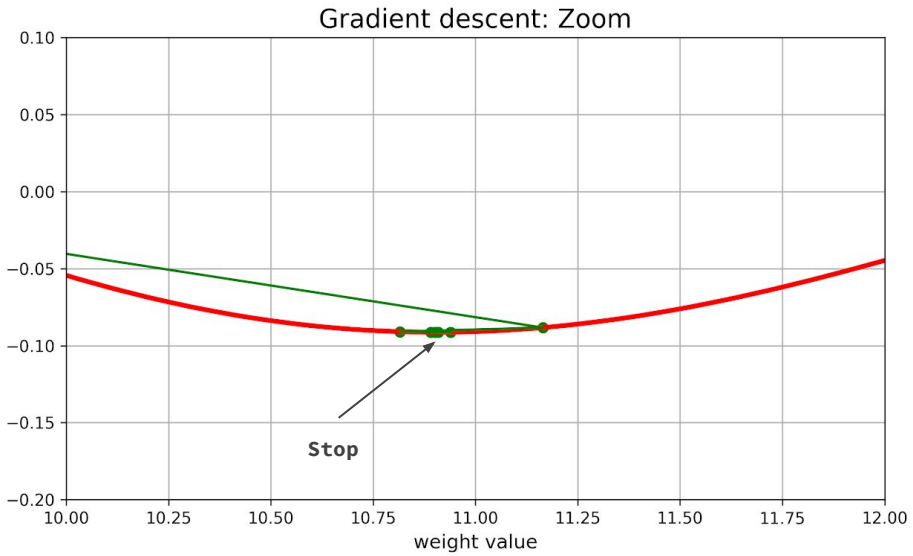
Optimization: Learning rate alto



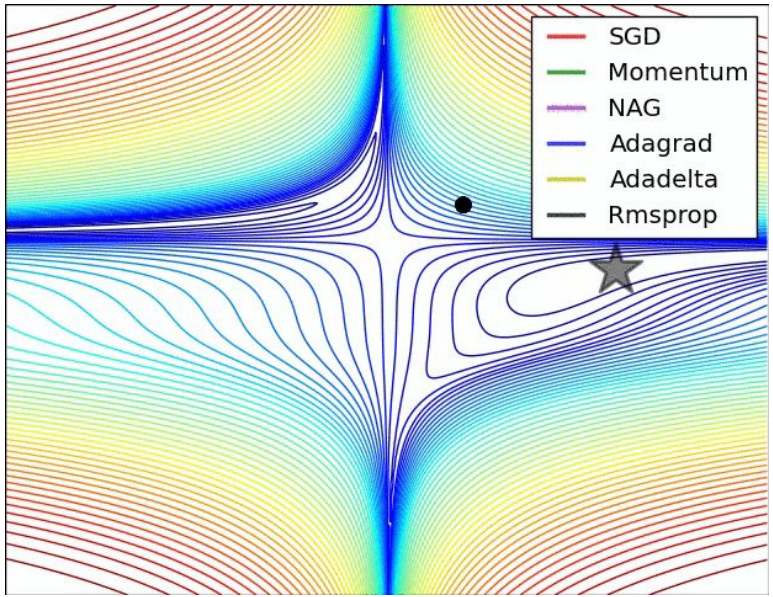


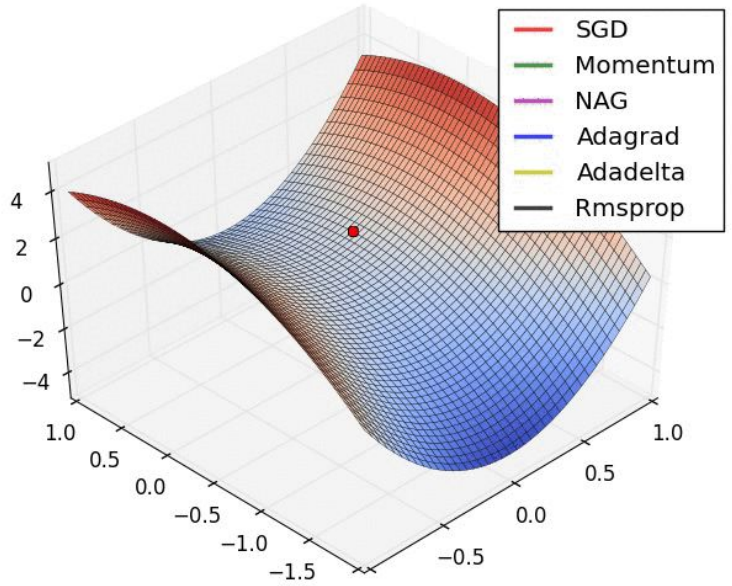
Optimization: Problema del minimo locale





Optimization: Algoritmi oltre l'SGD





Nota: Immagini tratte da http://cs231n.github.io/neural-networks-3/

Backpropagation

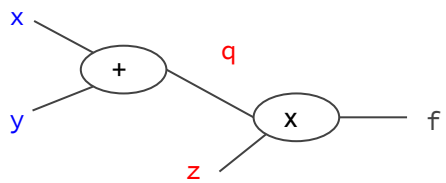
- La Loss function in generale dipende da più variabili (weights e bias)

$$w(i+1) = w(i) - \eta \frac{dL}{dw} \longrightarrow W(i+1) = W(i) - \eta \nabla L$$

$$W = (w_0, w_1, \ldots, w_N)$$
$$\nabla L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_N}\right)$$

- **Chain rule:** $\frac{df(g(x))}{dx} = \frac{df}{dg}\frac{dg}{dx}$

$$f(x,y,z) = (x+y)z$$
$$f = qz; \quad q = x+y$$



$$\frac{df}{dq} = z; \quad \frac{df}{dz} = q$$

$$\frac{df}{dx} = \frac{df}{dq}\frac{dq}{dx}; \quad \frac{df}{dy} = \frac{df}{dq}\frac{dq}{dy}$$

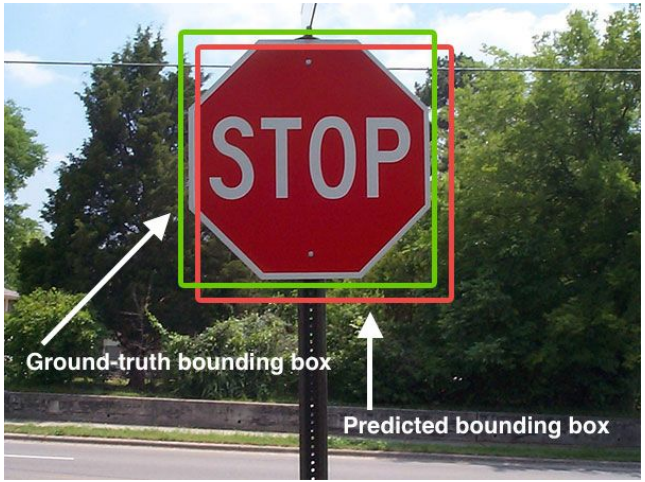
Derivate locali

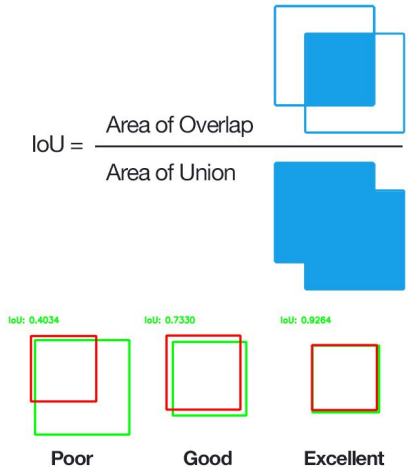
Derivate locali

Misura della prestazione: Esempi

Accuracy ⇨ $\frac{Predizioni\ corrette}{Totale\ predizioni} = \frac{TP+TN}{TP+TN+FP+FN}$

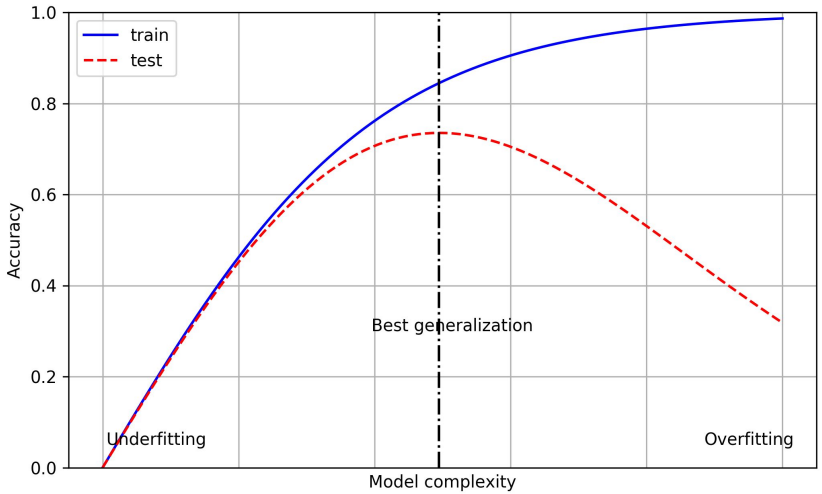
Jaccard Index (IOU) ⇨

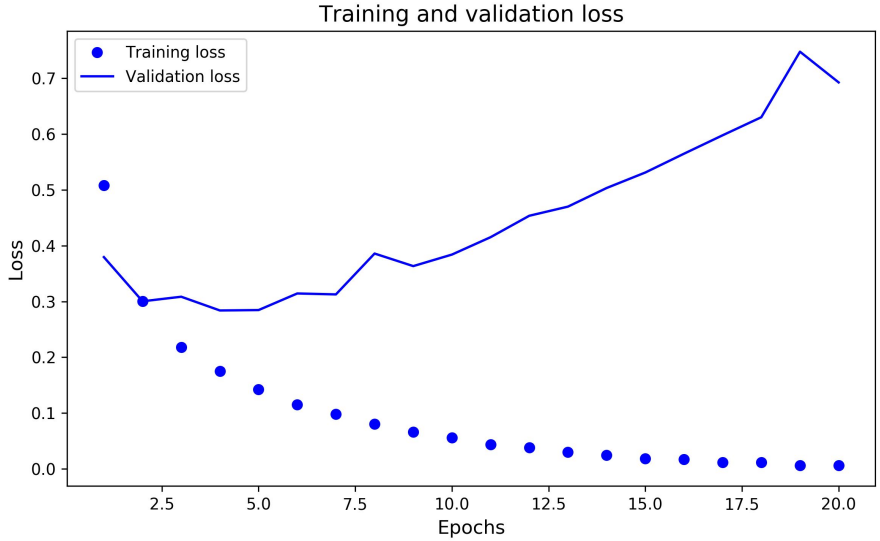




Overfitting: Validation set e test set

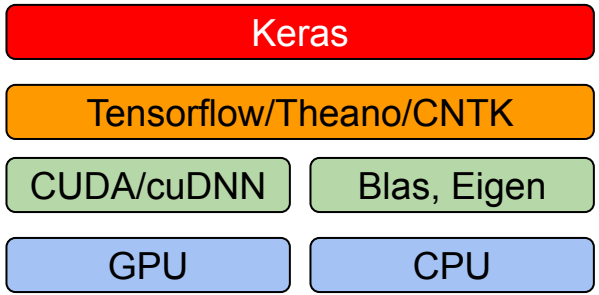
- Validation set e test set
- Tuning degli iper-parametri





Introduzione a Keras

Keras



- È una API Python di alto livello per poter sviluppare modelli con neural networks
- Può usare diversi tipi di Backend che si occupano dell'organizzazione dei calcoli
- Può sfruttare la potenza di calcolo offerta da diversi acceleratori (GPU, FPGA, TPU)

Keras: Tensore

- Nel contesto dei framework per DL/ML è un “contenitore” nel quale vengono strutturati i dati
- Può avere K dimensioni:
  - K=2: dati sotto forma di vettori → (samples, features)
  - K=3: timeseries → (samples, timesteps, features)
  - K=4: Immagini → (samples, channels, high, width)
  - K=5: Video → (samples, frames, channels, high, width
- I layer di una ANN hanno come I/O un tensore

Keras: MNIST Example

- Database di cifre scritte a mano
- 60000 train, 10000 test
- Ogni dato di input è un’immagine a 8 bit da 28x28 pixels
- L’uscita desiderata è espressa con un intero (0-9)

Keras: MNIST, codice completo

```
from keras.datasets import mnist
from keras import models
from keras import layers
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

network.fit(train_images, train_labels, epochs=5, batch_size=128)

test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

Caricamento dei dati e preprocessing

Definizione rete

Training

Testing

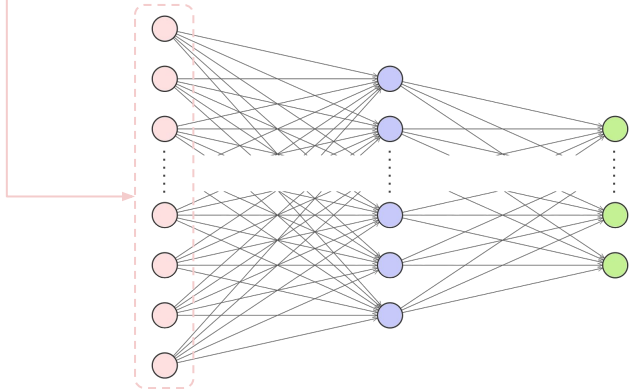
Keras: MNIST

Descrizione dell'architettura (Sequential model):

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	401920
dense_2 (Dense)	(None, 10)	5130

Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0



Keras: MNIST

Compilazione del modello e training (specifica loss, optimizer, metrics):

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Output fase di Training:

Epoch 1/5
60000/60000 [==============================] - 4s 75us/step - **loss: 0.2562 - acc: 0.9260**
Epoch 2/5
60000/60000 [==============================] - 5s 84us/step - loss: 0.1044 - acc: 0.9686
Epoch 3/5
60000/60000 [==============================] - 5s 76us/step - loss: 0.0695 - acc: 0.9789
Epoch 4/5
60000/60000 [==============================] - 5s 85us/step - loss: 0.0501 - acc: 0.9850
Epoch 5/5
60000/60000 [==============================] - 4s 73us/step - **loss: 0.0384 - acc: 0.9883**

Keras: MNIST

Valutazione del modello (test set):

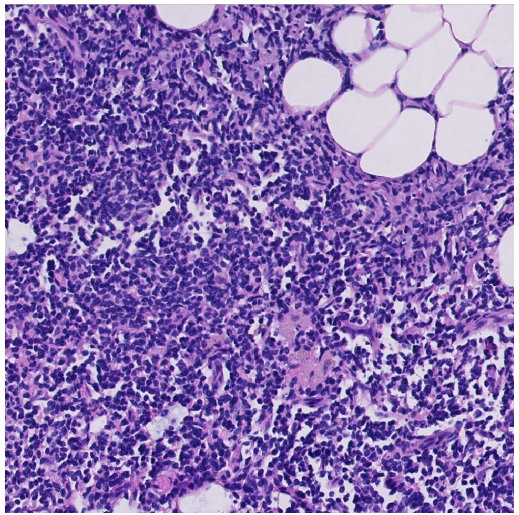
```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

10000/10000 [==============================] - 0s 47us/step

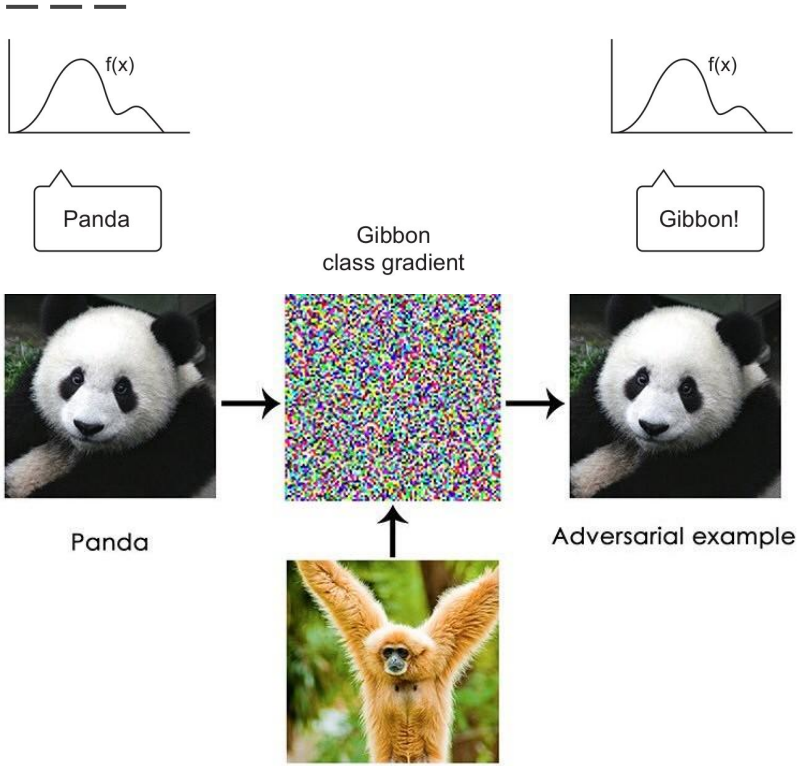
test_acc: 0.9774

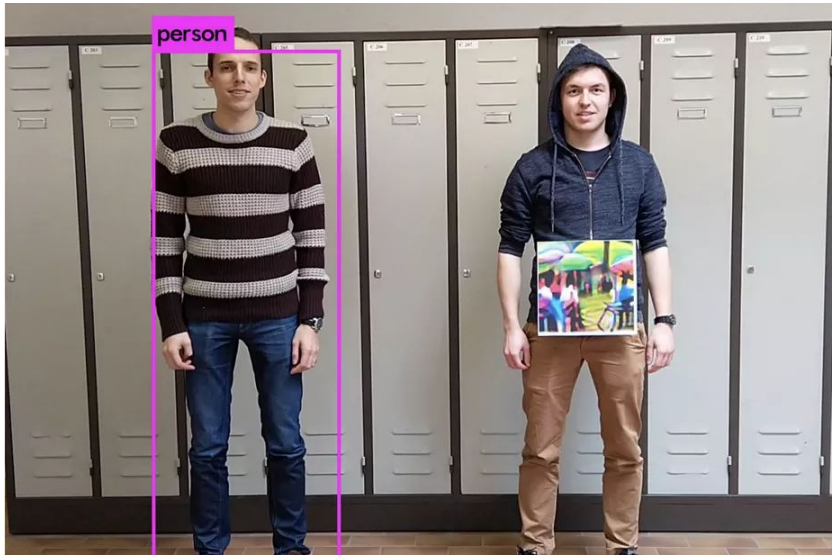
Limiti del Deep Learning (1)

- La ANN riconosce solo quello cha ha imparato



Limiti del Deep Learning (2): Adversarial images





Nota: Immagine tratta da *Francois Chollet, Deep Learning with Python*

Nota: Immagine tratta da https://arxiv.org/pdf/1904.08653.pdf

Domande