



LUMSA
UNIVERSITÀ
1939 **80**OTTANTESIMO 2019

Maurizio Naldi (IU0TPA)

Ordinario di Computer Science

Università LUMSA

L'intelligenza artificiale per i radioamatori

28 Settembre 2026

ARI - Sezione Roma



1. Introduzione all'intelligenza artificiale
2. Modelli classici e reti neurali
3. Strumenti codeless e code-based (Python)
4. Esempi di applicazione della IA in campo radioamatoriale

Istanza: un singolo esempio nel dataset

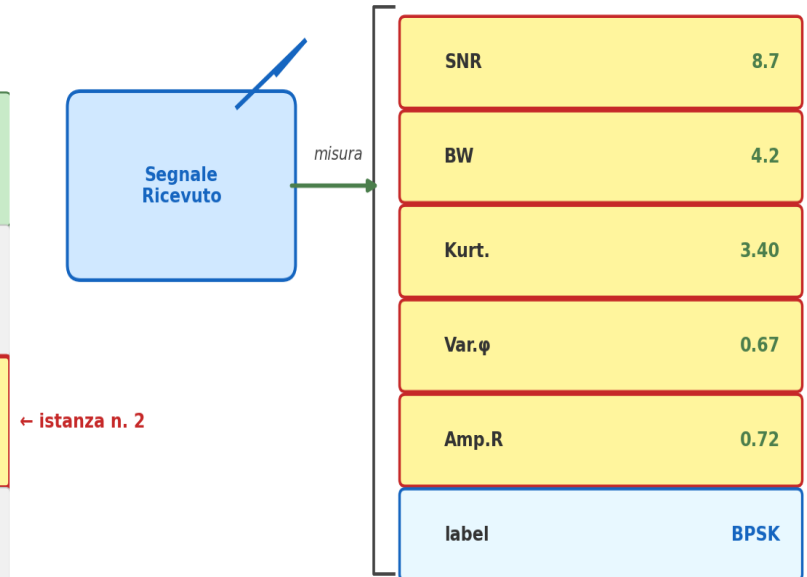
Un dataset è una tabella: ogni riga è un'istanza (un oggetto osservato), ogni colonna è una feature

In campo radio: SNR, larghezza di banda, kurtosis... diventano le componenti del vettore che identifica quella singola istanza

Un dataset: righe = istanze, colonne = feature

SNR (dB)	BW (kHz)	Kurtosis	Var. Fase	Amp. Ratio	Modulazione
12.3	15.0	1.80	0.12	0.95	AM
8.7	4.2	3.40	0.67	0.72	BPSK
15.1	25.3	1.20	0.08	0.98	FM
9.4	4.1	3.55	0.71	0.69	BPSK
6.2	50.1	1.05	0.03	0.99	FM

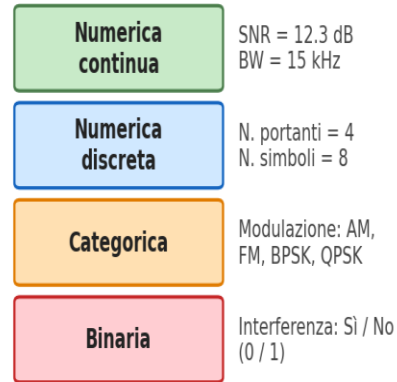
Un'istanza = vettore di feature $\vec{x}$



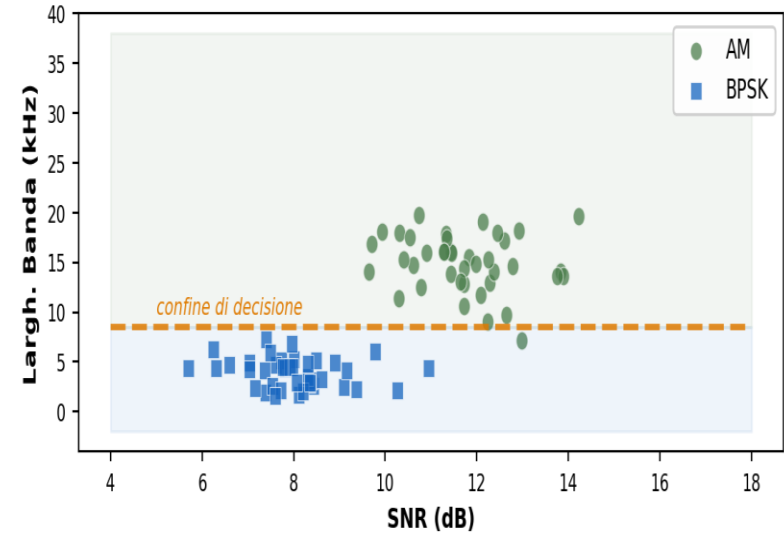
Ogni istanza è descritta da un insieme di misure (le feature) che ne catturano le proprietà

Feature: le caratteristiche che descrivono un'istanza

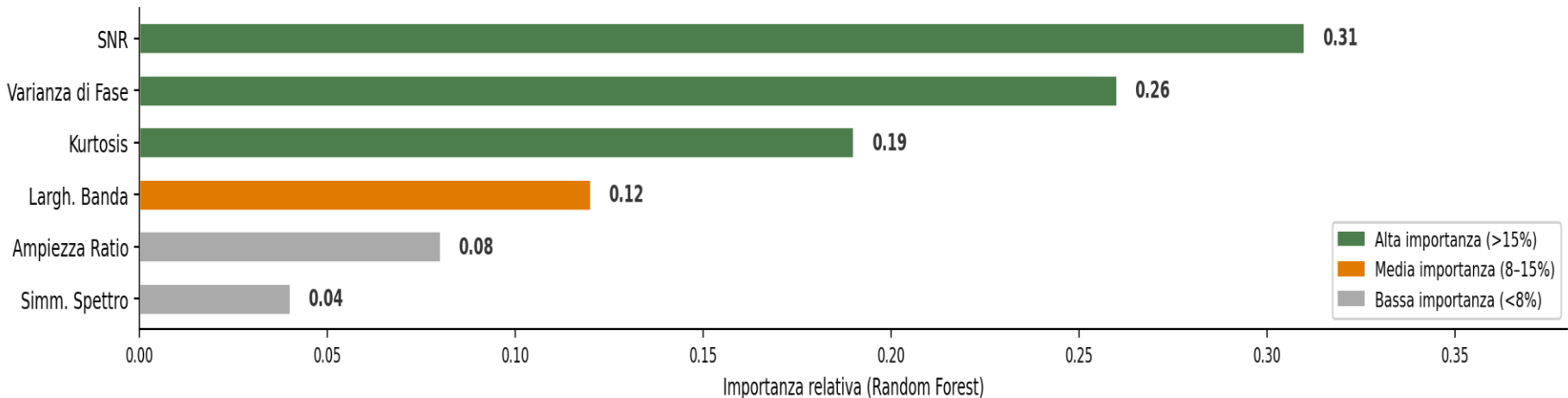
Tipi di feature



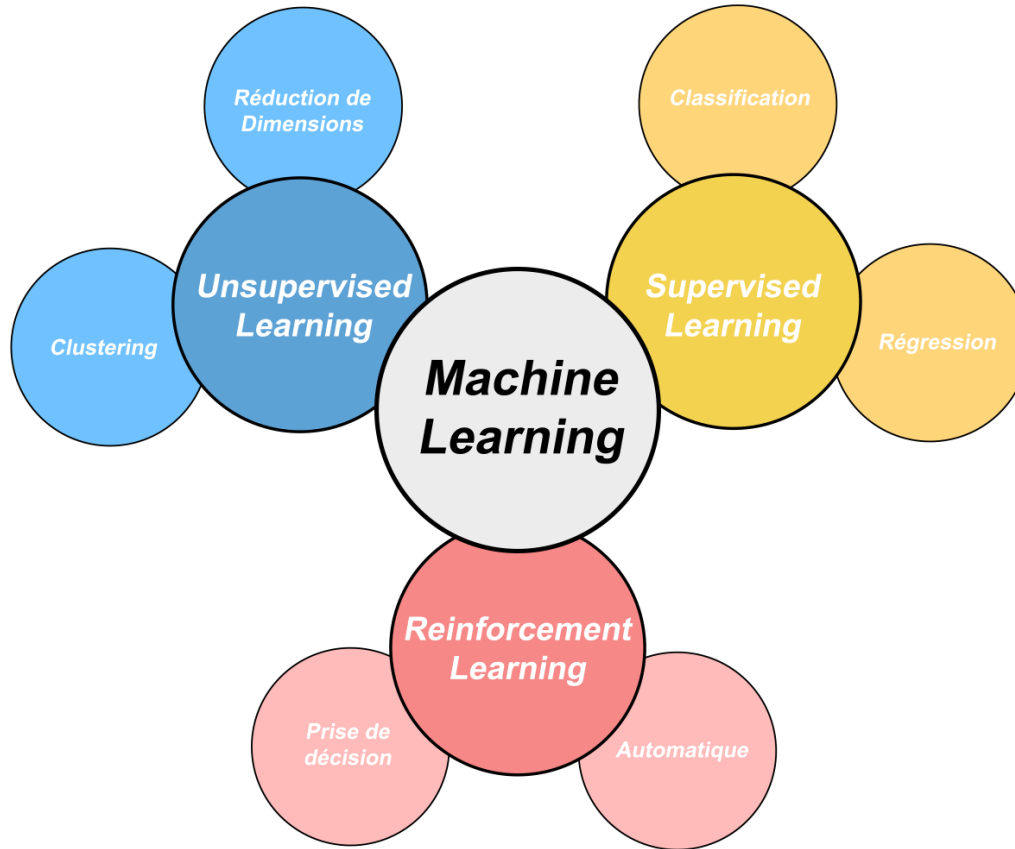
Potere discriminante di due feature



Importanza delle feature nella classificazione della modulazione

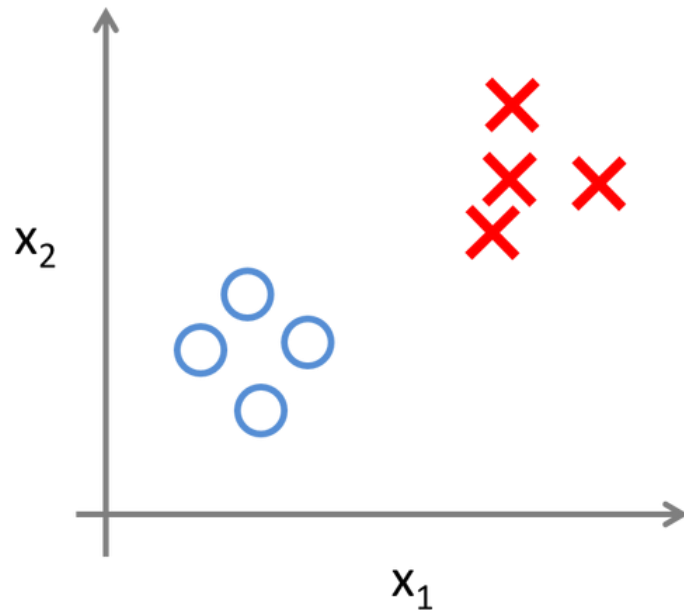


II Machine Learning

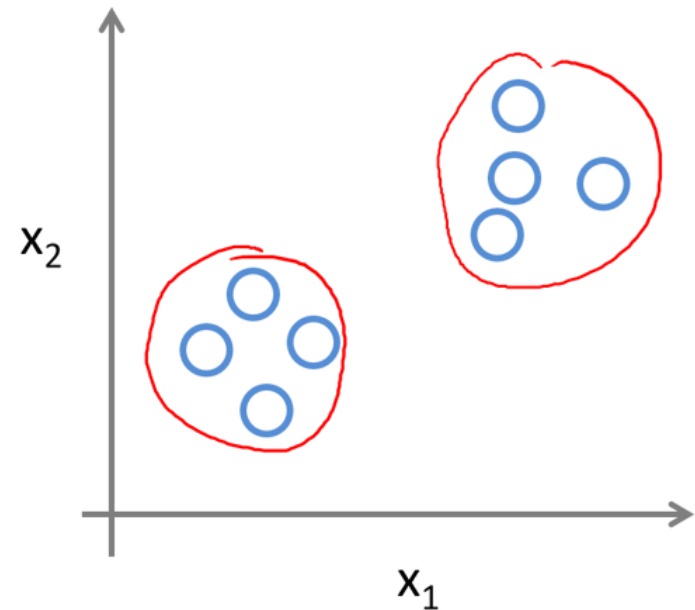


Apprendimento supervisionato vs non supervisionato

Supervised Learning



Unsupervised Learning



- Nell'apprendimento supervisionato disponiamo di esempi in cui conosciamo già la risposta (la "ground truth")
- Nell'apprendimento non supervisionato dobbiamo effettuare una suddivisione in classi basandoci solamente sulla vicinanza delle features (clustering)

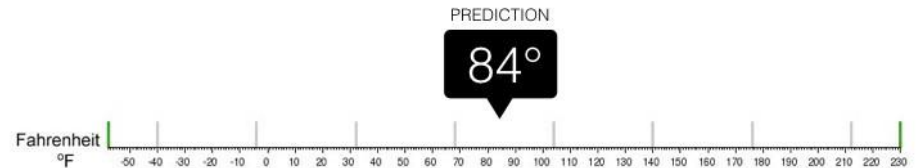
Compiti del machine learning supervisionato

- Le tecniche di machine learning svolgono essenzialmente due compiti
- Regressione/Predizione
- Classificazione



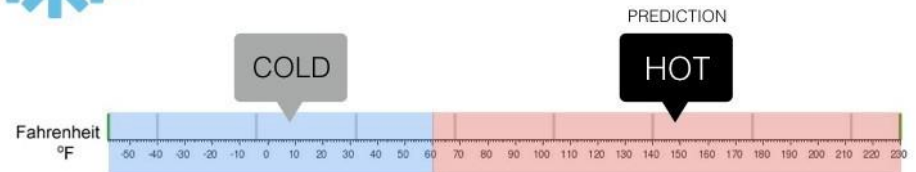
Regression

What is the temperature going to be tomorrow?

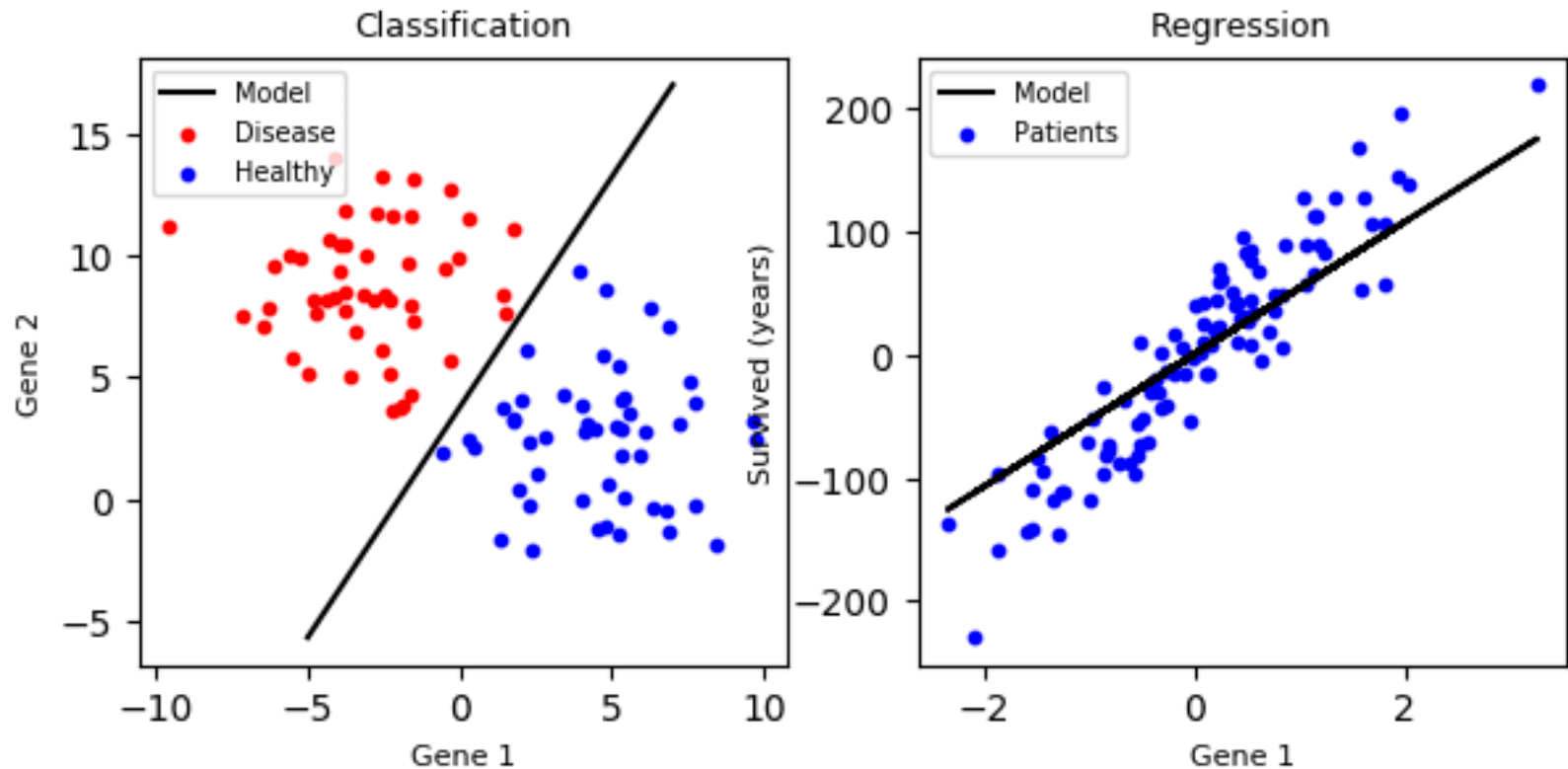


Classification

Will it be Cold or Hot tomorrow?

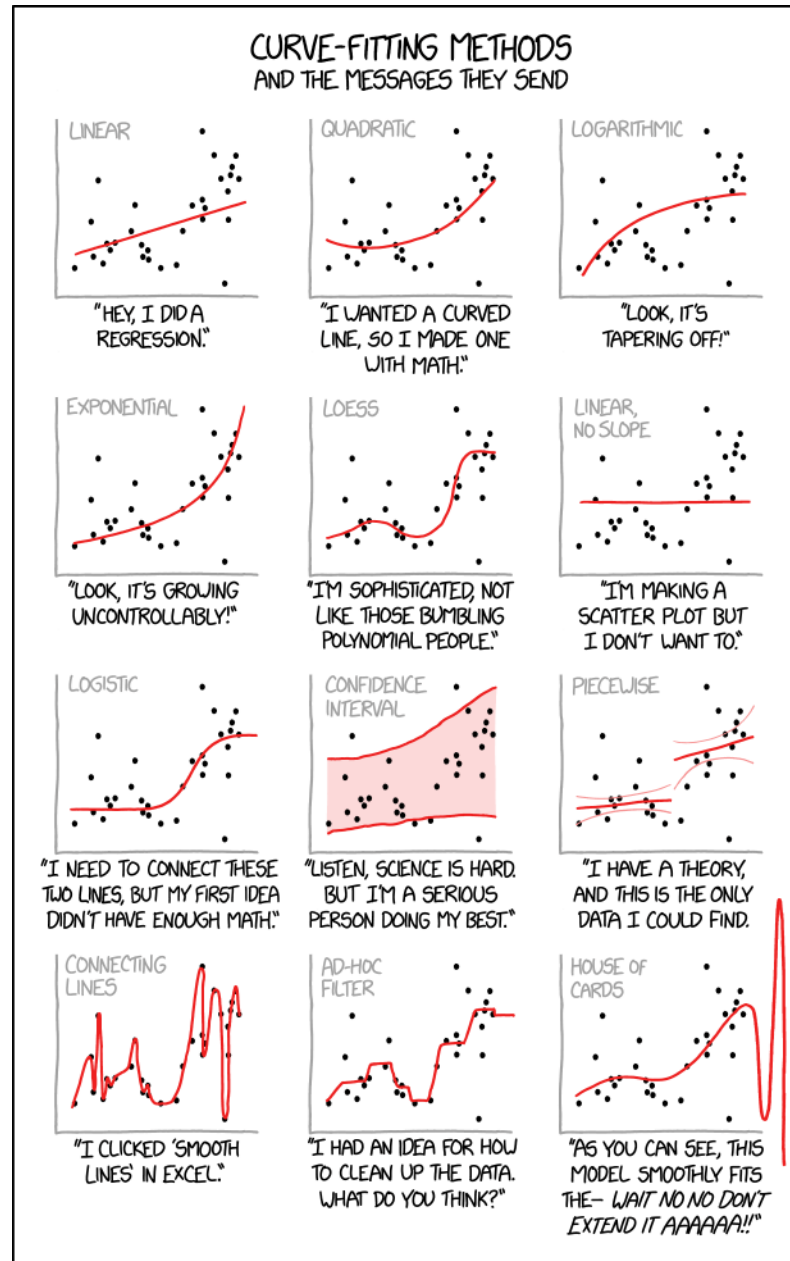


Classificazione vs regressione

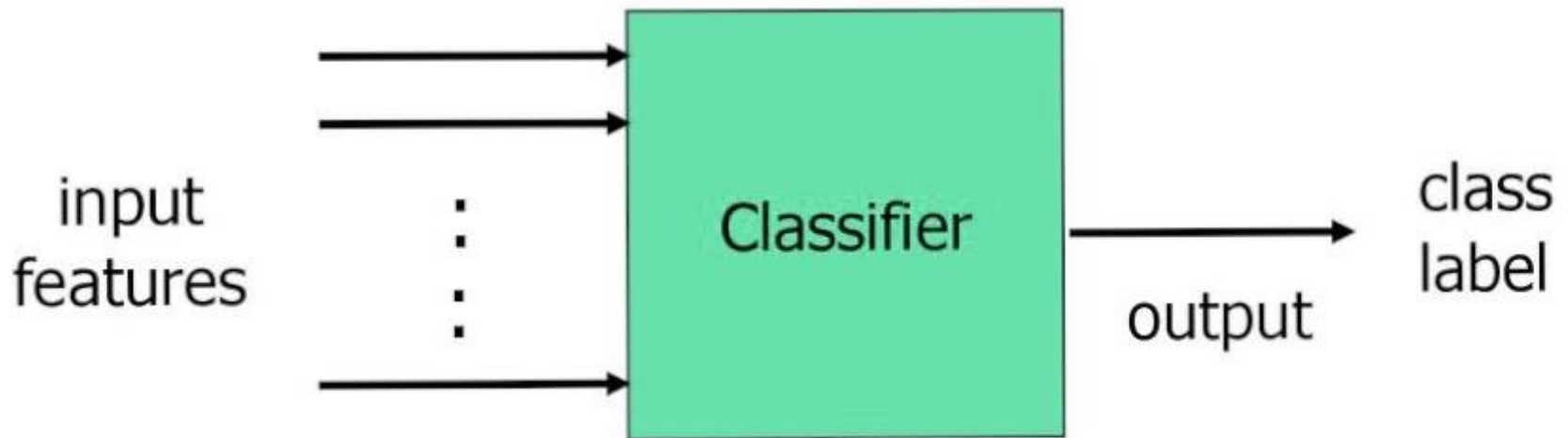


- La classificazione individua dei confini tra i valori associati a classi diverse
- La regressione individua una funzione matematica (per esempio una funzione lineare, corrispondente ad una retta), che individua il valore più vicino ai dati

Forme di regressione

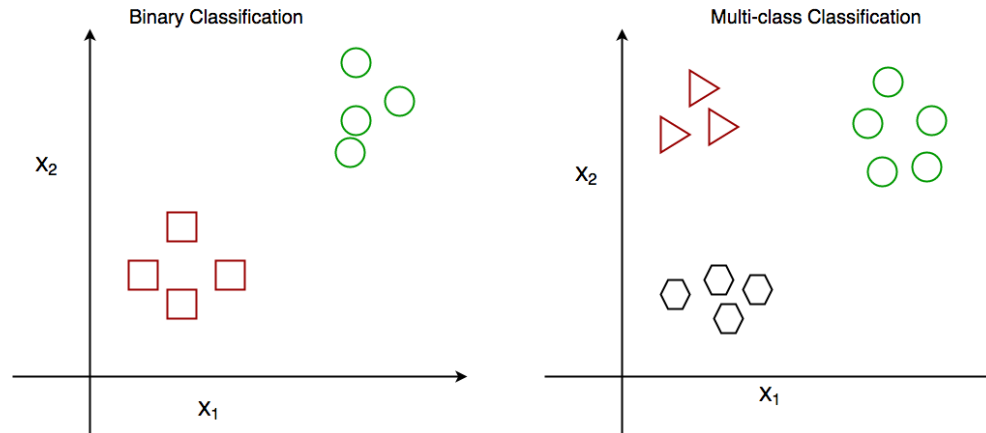


Il classificatore: ingresso e uscita



- Le features sono le caratteristiche dell'oggetto che si vuole classificare
- Il classificatore fornisce in uscita la classe a cui verosimilmente appartiene l'oggetto

La classificazione



- I dati vengono raggruppati secondo la vicinanza relativa in un piano cartesiano (eventualmente con molte dimensioni)
- La classificazione binaria prevede due soli gruppi (classi), quindi permette di rispondere a domande la cui risposta è del tipo SI/NO
- Nella classificazione multi-class abbiamo invece più gruppi

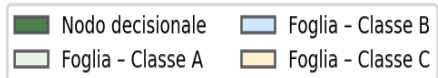
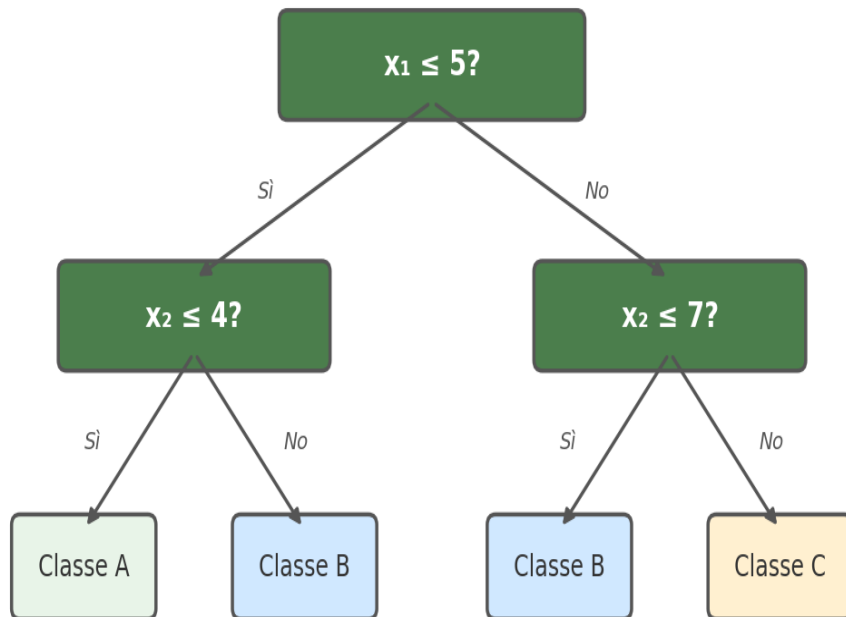
Misure di prestazioni

- Esistono diverse misure di classificazione, ognuna delle quali esamina un aspetto del comportamento del classificatore
- Una misura complessiva è la Accuracy, che è la percentuale di dati classificati correttamente
- In questo caso, la accuracy è $\frac{30+40}{100} = 70\%$
- Il suo complemento a 1 (100%) è il tasso di errore

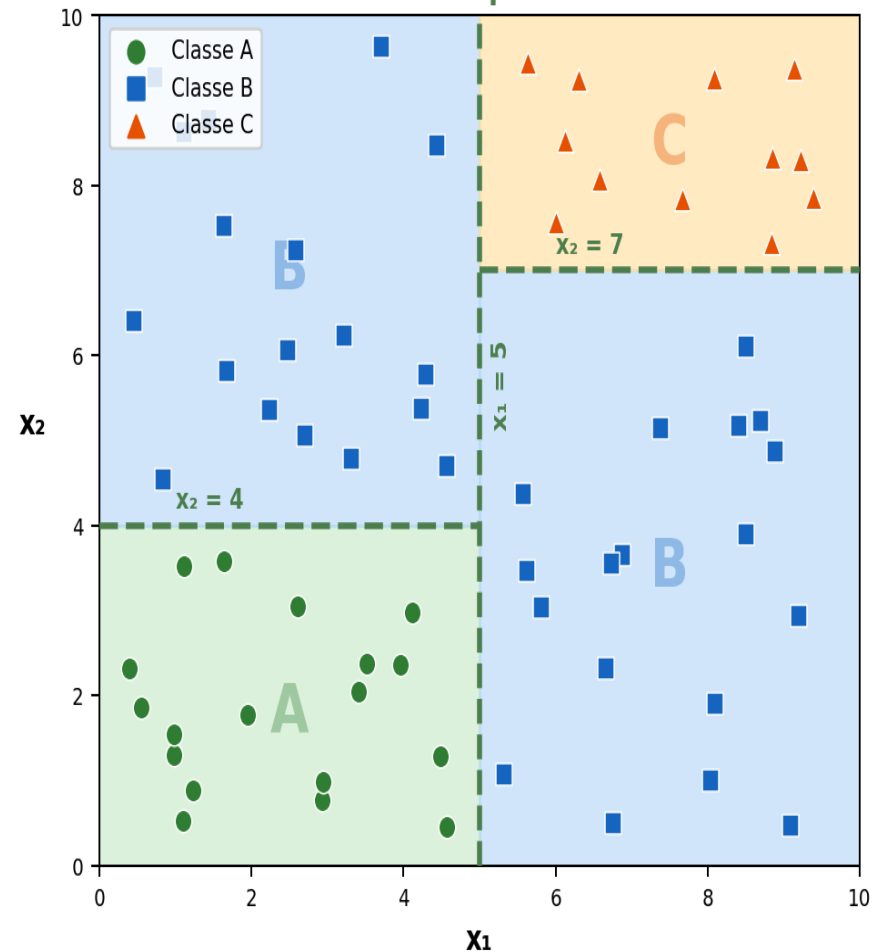
	<i>Predicted</i> ⊕	<i>Predicted</i> ⊖	
<i>Actual</i> ⊕	30	20	50
<i>Actual</i> ⊖	10	40	50
	40	60	100

Alberi di Decisione: struttura e partizione dello spazio

Albero di decisione



Suddivisione dello spazio delle feature



Alberi di Decisione: principi di funzionamento

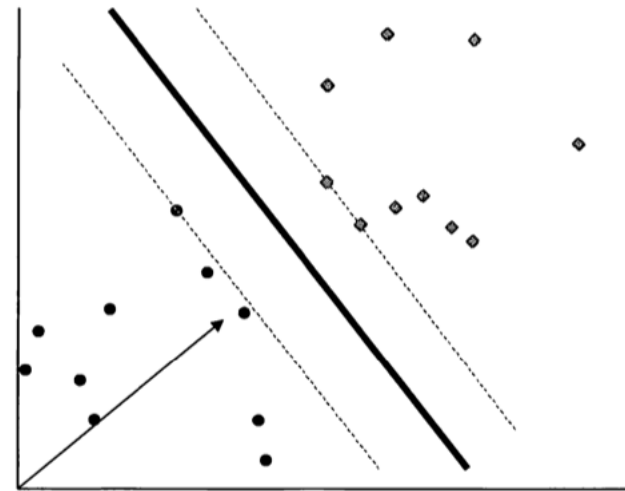
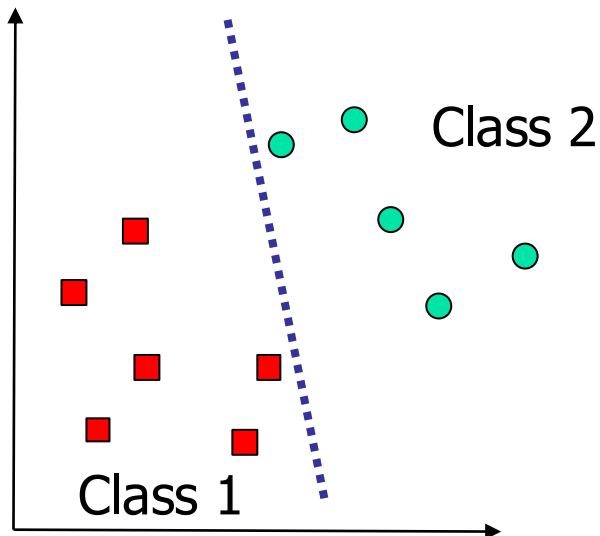
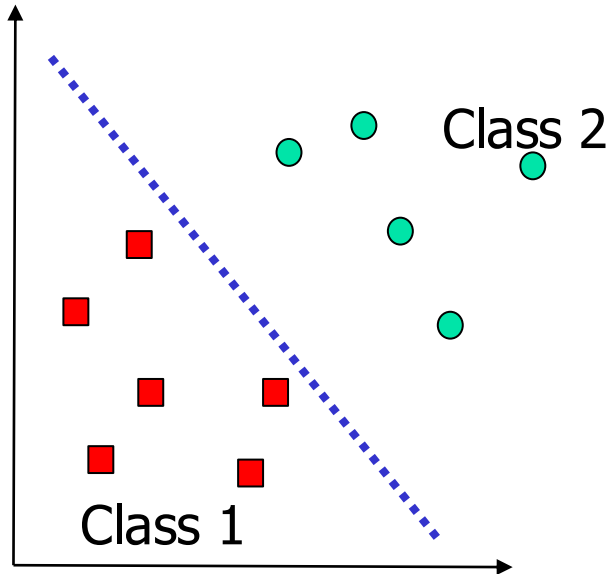
- Un albero di decisione è un modello che suddivide i dati in sottogruppi tramite una serie di domande binarie (nodi), fino ad assegnare ogni istanza a una classe (foglie)
- Ad ogni nodo si sceglie la feature che massimizza la riduzione di impurità (es. Indice di Gini o Entropia / Information Gain)
- Algoritmi principali: ID3, C4.5, CART — tutti costruiscono l'albero in modo greedy (top-down)
- Il modello risultante è interpretabile: il percorso radice-foglia descrive in modo leggibile la regola di classificazione

Alberi di Decisione: vantaggi, limiti e applicazioni

- **Vantaggi:** alta interpretabilità, non richiede normalizzazione dei dati, gestisce feature categoriche e numeriche, rapido in fase di predizione
- **Limiti:** tendenza all'overfitting su alberi profondi; sensibile a piccole variazioni nei dati (alta varianza); confini di decisione rettangolari
- Il pruning (potatura) riduce la complessità e migliora la generalizzazione limitando la profondità massima o il numero minimo di campioni per foglia
- **Estensioni ensemble:** Random Forest (bagging di alberi) e Gradient Boosted Trees (es. XGBoost) superano i limiti del singolo albero mantenendo parte dell'interpretabilità

Il modello SVM nello spazio delle feature

- La linea di separazione deve essere il più possibile distante dalle istanze delle due classi



Support Vector Machine (SVM): principi di funzionamento

- L'SVM cerca l'iperpiano di separazione che massimizza il margine tra le due classi; solo i punti più vicini al confine (support vectors) determinano la soluzione
- Il soft margin bilancia il margine massimo con la tolleranza agli errori di classificazione sul training set
- **Kernel trick:** proiettando i dati in uno spazio a dimensione superiore (tramite funzioni kernel come RBF, polinomiale, sigmoidale) si rende separabile linearmente un problema non lineare
- Il training si riduce a un problema di ottimizzazione quadratica convessa, con soluzione globale garantita

SVM: vantaggi, limiti e applicazioni

- **Vantaggi:** efficace in spazi ad alta dimensionalità (es. testo, spettri RF), robusto all'overfitting grazie alla massimizzazione del margine, funziona bene con pochi dati di training
- **Limiti:** scalabilità critica su dataset grandi (complessità $O(n^2 \div n^3)$), scelta del kernel e tuning di C e γ richiedono cross-validation; scarsa interpretabilità
- La versione SVR (Support Vector Regression) viene usata anche per task di predizione, es. stima del percorso di propagazione

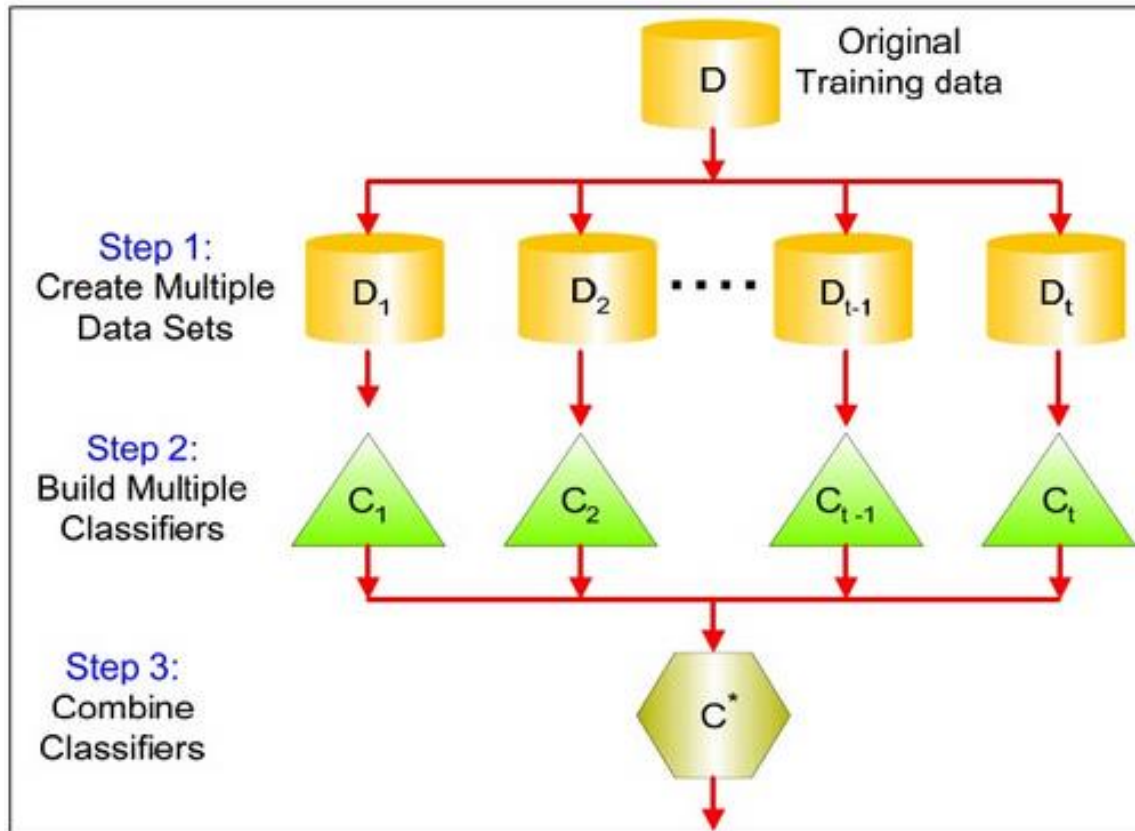
Naive Bayes: principi di funzionamento

- Naive Bayes applica il teorema di Bayes assumendo indipendenza condizionale tra le feature: $P(\text{classe} \mid \mathbf{x}) \propto P(\text{classe}) \cdot \prod P(x_i \mid \text{classe})$
- Il training consiste semplicemente nel stimare le probabilità a priori $P(\text{classe})$ e le verosimiglianze $P(x_i \mid \text{classe})$ dai dati di addestramento
- **Varianti principali:** Gaussian NB (feature continue con distribuzione normale), Multinomial NB (conteggi, es. testo), Bernoulli NB (feature binarie)
- Il Laplace smoothing evita il problema di probabilità zero per feature mai osservate nel training (es. parole rare nel classificatore di testo)

Naive Bayes: vantaggi, limiti e applicazioni

- **Vantaggi:** estrema semplicità e velocità di training anche su grandi dataset; funziona bene con dati limitati; robusto alle feature irrilevanti; fornisce probabilità calibrate
- **Limiti:** l'ipotesi di indipendenza è raramente vera nella pratica; le stime di probabilità possono essere imprecise; scarso con feature altamente correlate
- Gaussian NB è spesso usato come baseline veloce prima di modelli più complessi, grazie alla sua semplicità implementativa

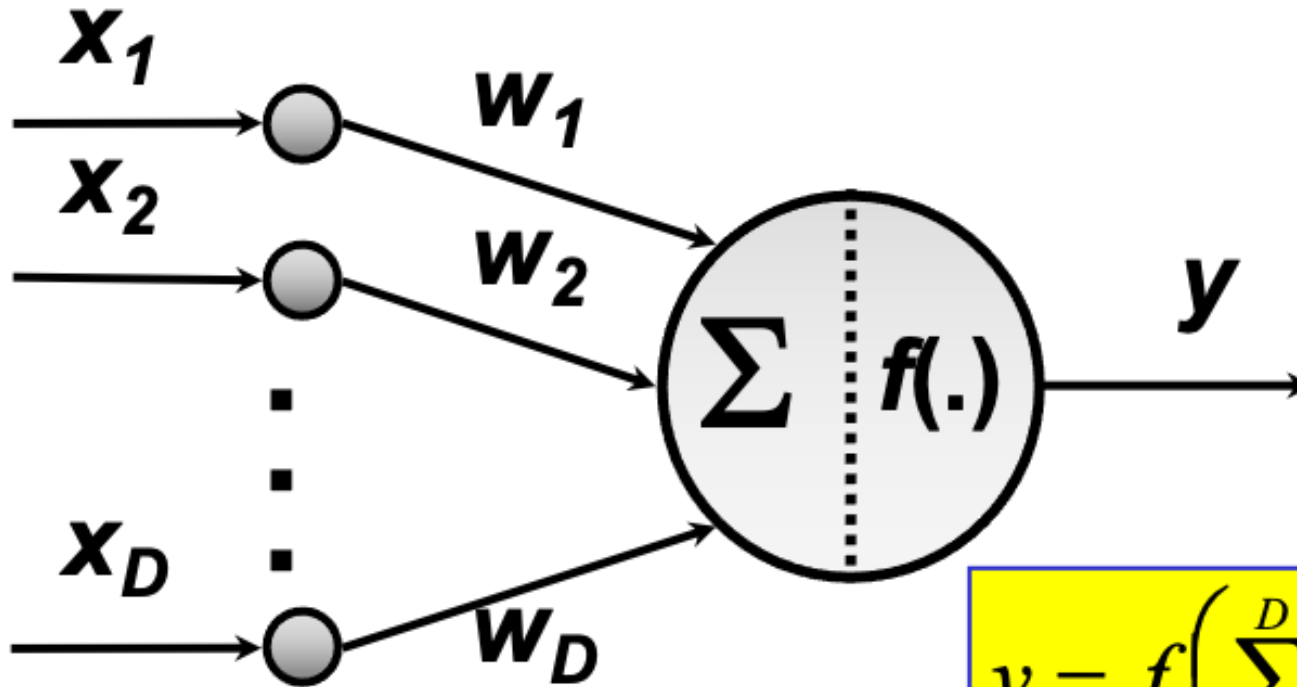
Ensemble learning



Ensemble: esempio illustrativo

Reality							
1							
2							
3							
4							
5							
Combine							

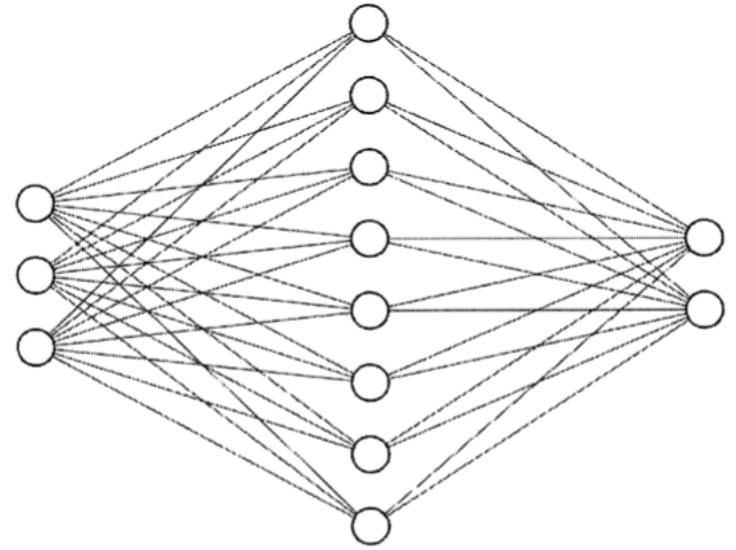
PROCESSORE ELEMENTARE



$$y = f\left(\sum_{i=1}^D w_i \cdot x_i\right)$$

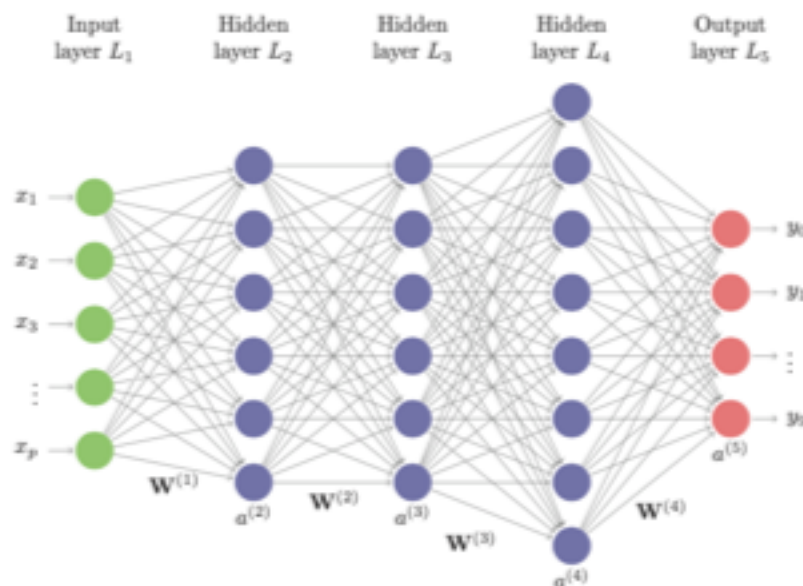
Reti neurali: struttura base (feedforward)

- Tre stadi
 - Input layer
 - Middle (hidden) layers
 - Output layer
- Il numero di pesi cresce con il numero di neuroni
- La complessità della fase di addestramento cresce con il numero di pesi

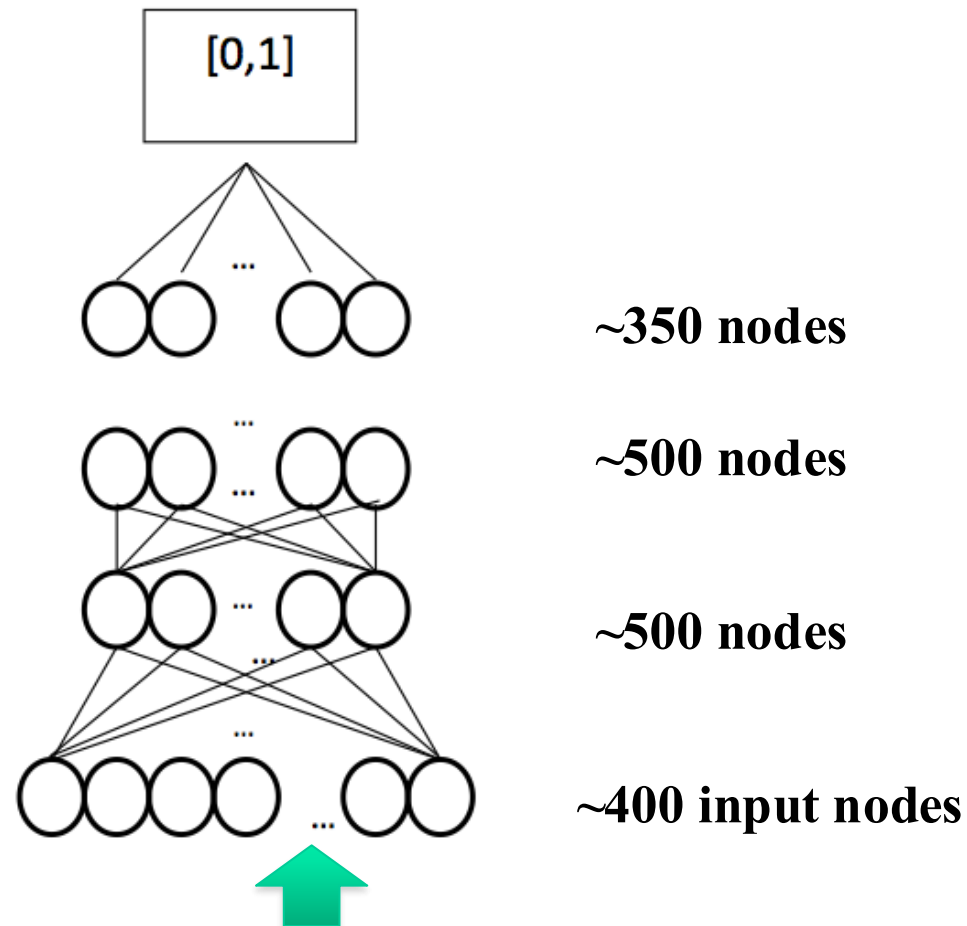


La rete neurale per il corpus MNIST

- La rete neurale utilizzata comprende 3 strati nascosti
- I tre strati nascosti sono costituiti rispettivamente da 1024, 1024 e 2048 nodi
- Lo strato d'ingresso ha 784 nodi (uno per ogni pixel)
- La rete richiede oltre 4 milioni di pesi da fissare
- Il tasso di errori è risultato pari allo 0.93%



An example of Deep Learning architecture



A Vector of ~400 Features (numbers between 0 and 1)

Training example

A dataset

Fields ***class***

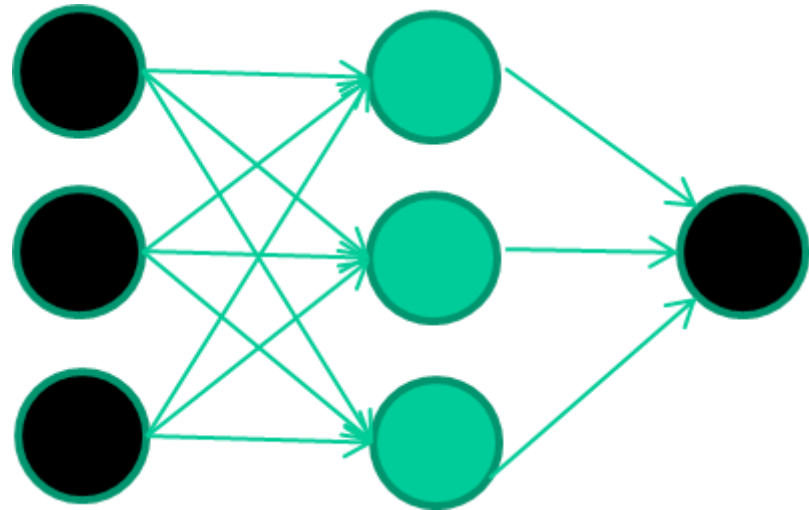
1.4 2.7 1.9 0

3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...



Training example

Training the neural network

Fields ***class***

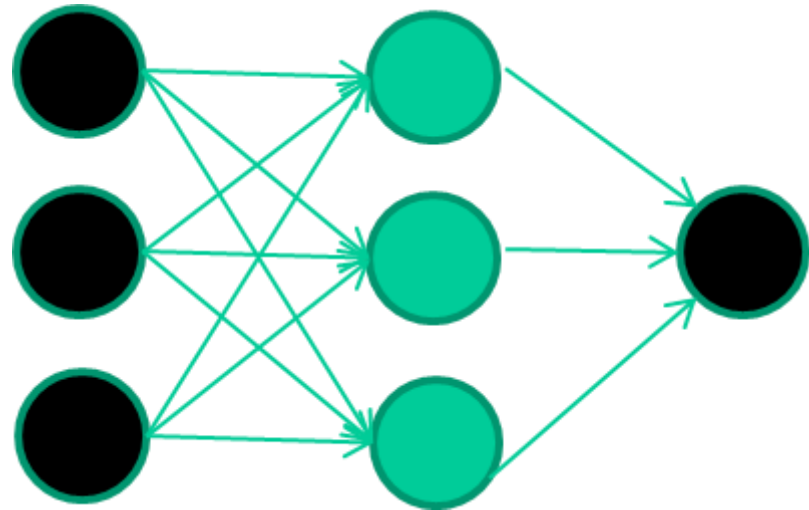
1.4 2.7 1.9 0

3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

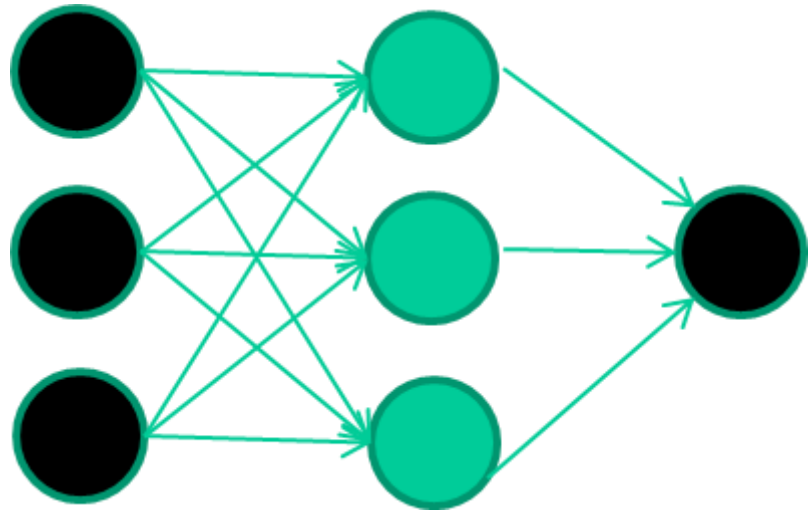
etc ...



Training data

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Initialise with random weights



Training example

Training data

Fields

1.4 2.7 1.9

3.8 3.4 3.2

6.4 2.8 1.7

4.1 0.1 0.2

etc ...

class

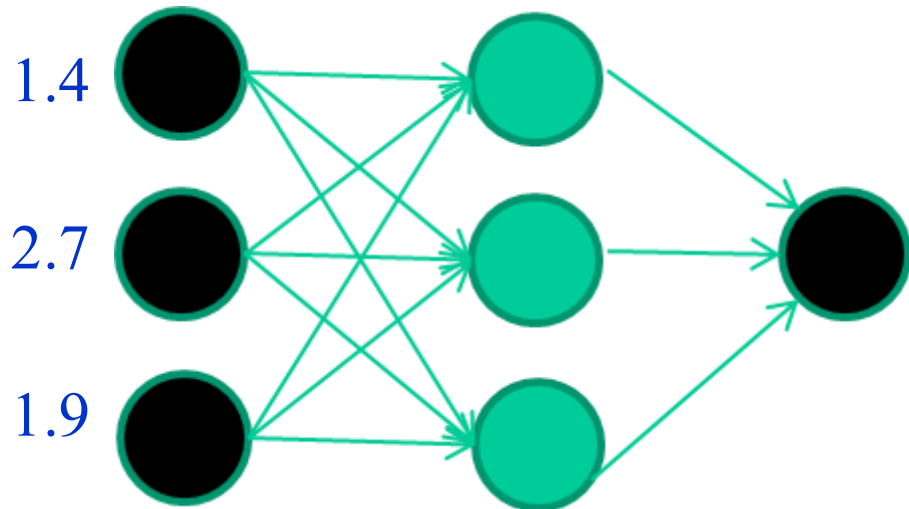
0

0

1

0

Present a training pattern



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

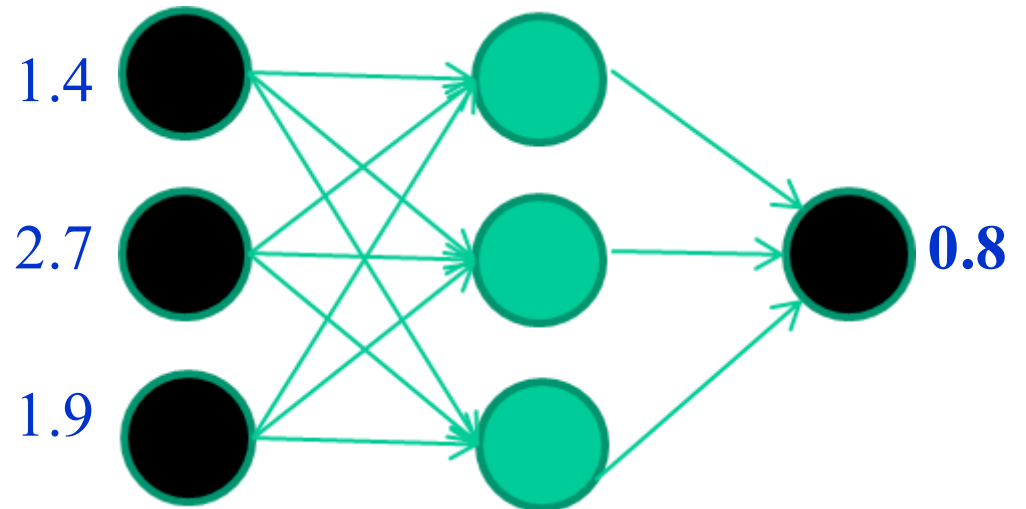
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Feed it through to get output



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

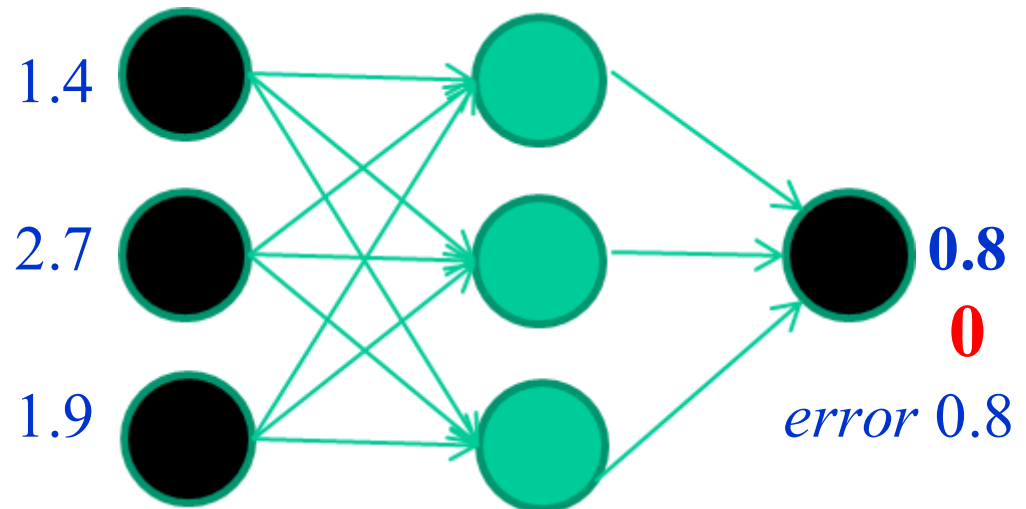
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Compare with target output



Training example

Training data

Fields *class*

1.4 2.7 1.9 0

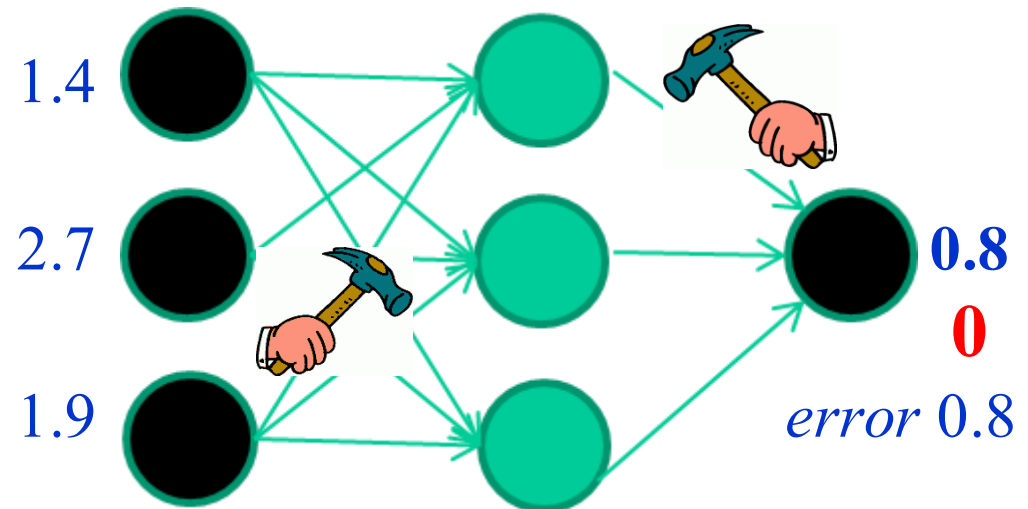
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Adjust weights based on error



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

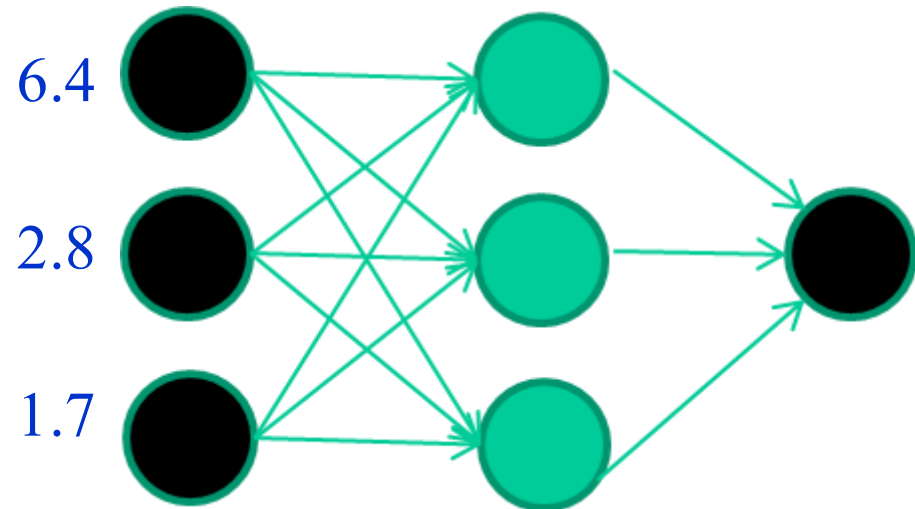
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Present a training pattern



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

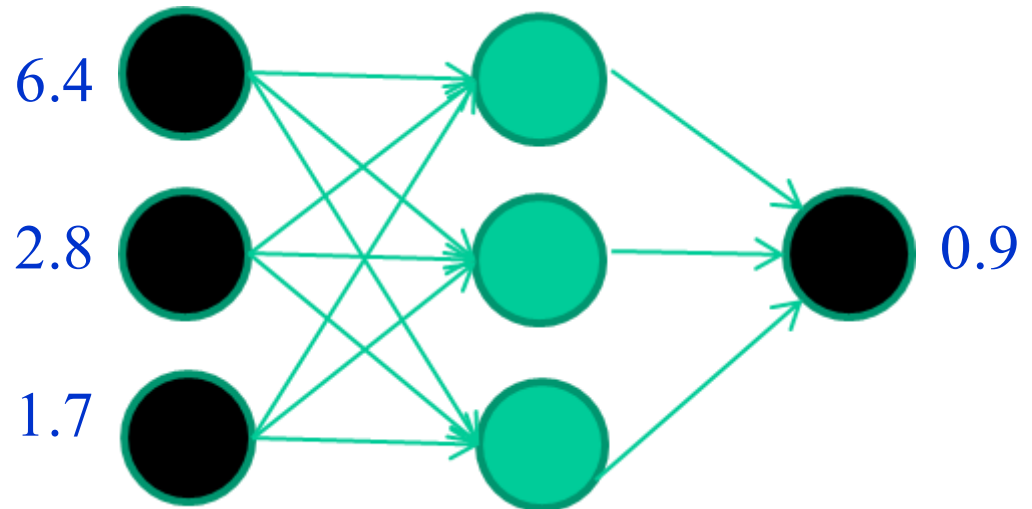
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Feed it through to get output



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

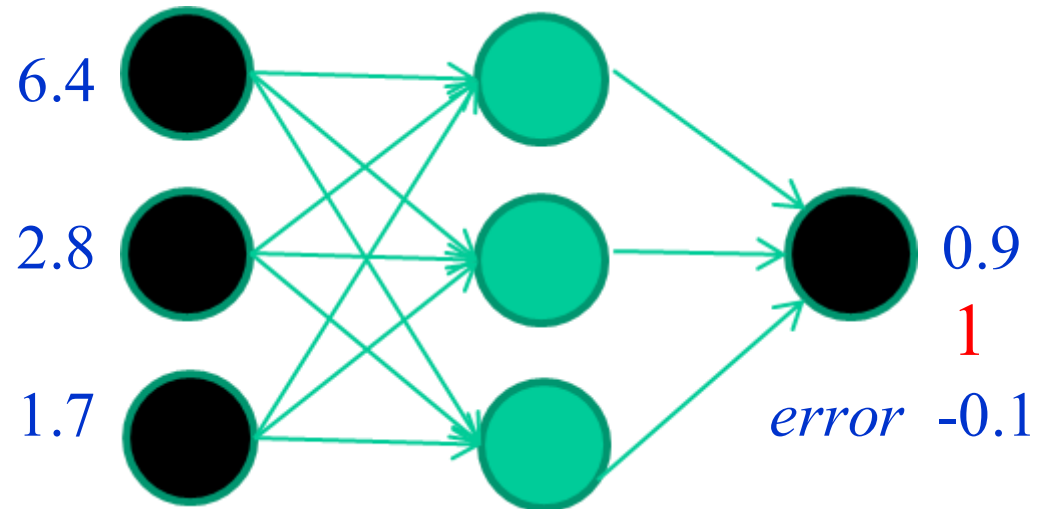
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Compare with target output



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

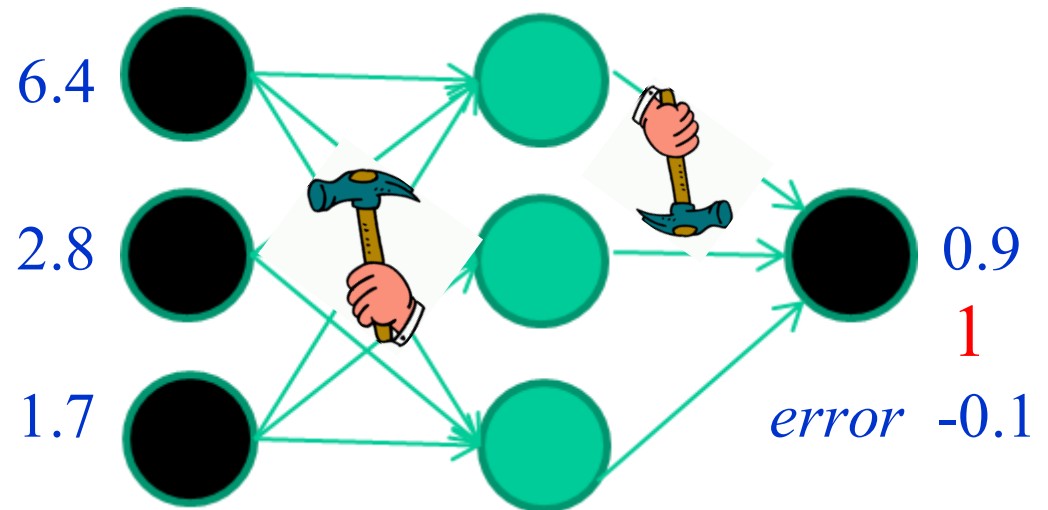
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Adjust weights based on error



Training example

Training data

Fields ***class***

1.4 2.7 1.9 0

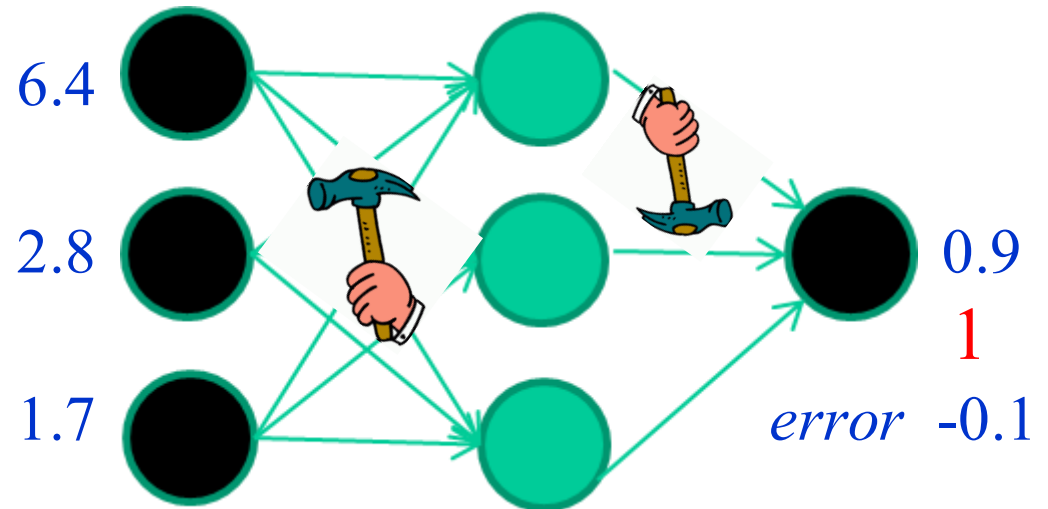
3.8 3.4 3.2 0

6.4 2.8 1.7 1

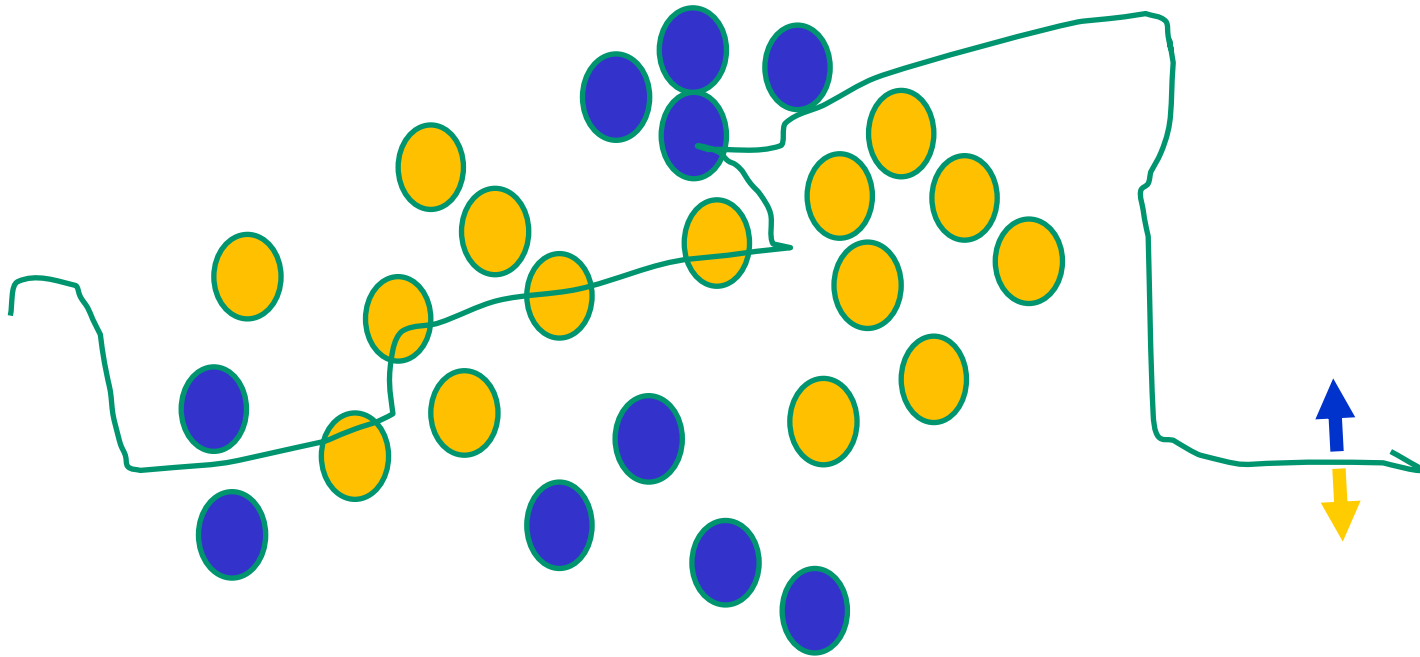
4.1 0.1 0.2 0

etc ...

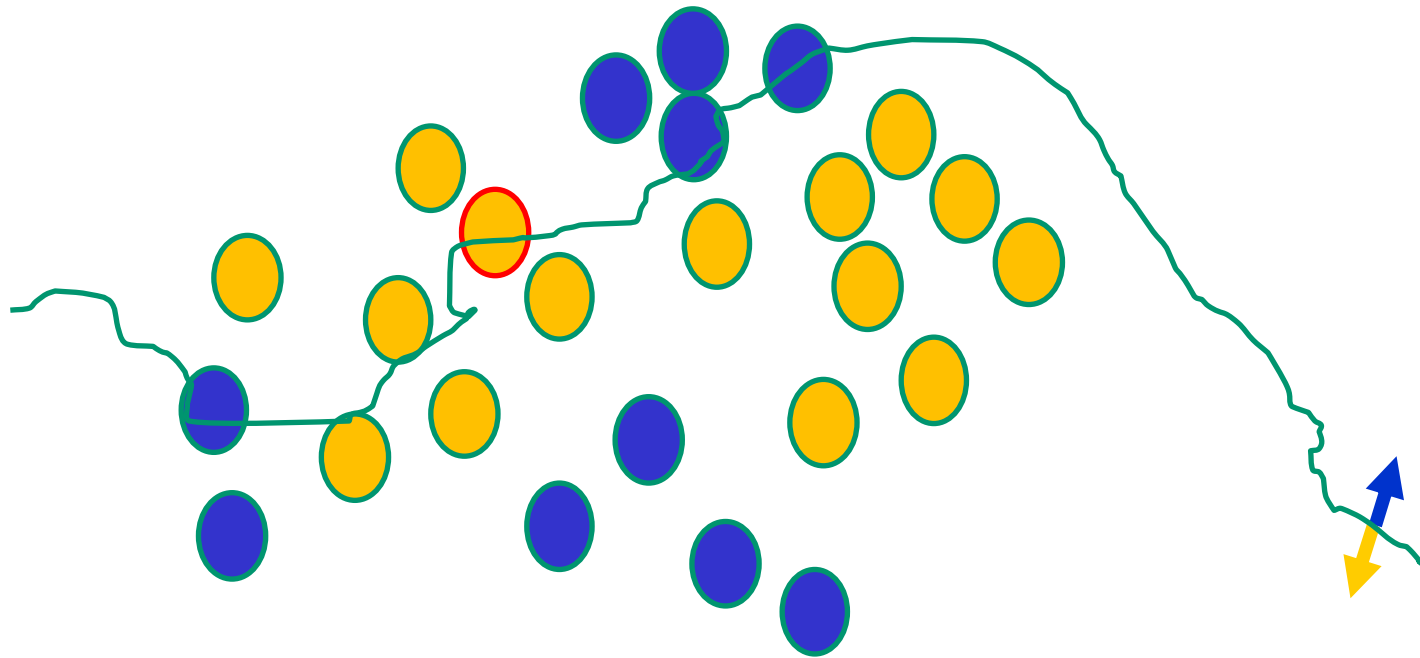
And so on



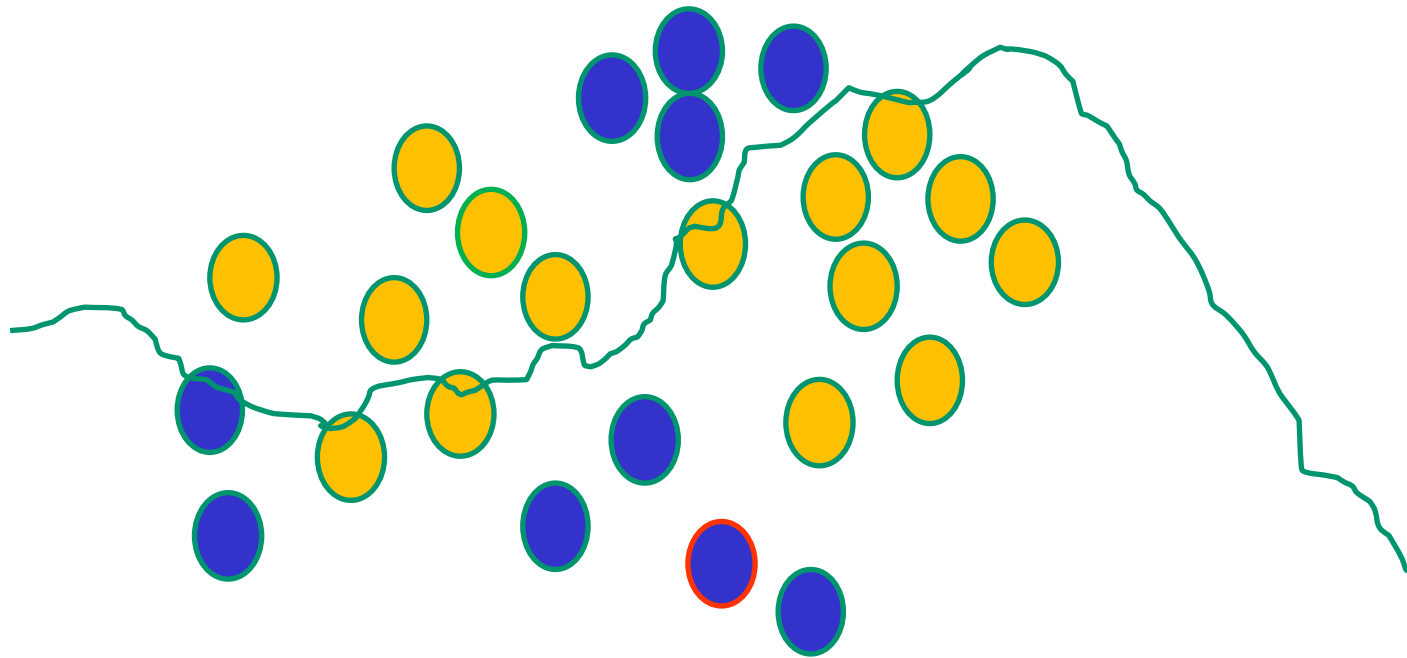
Initial random weights



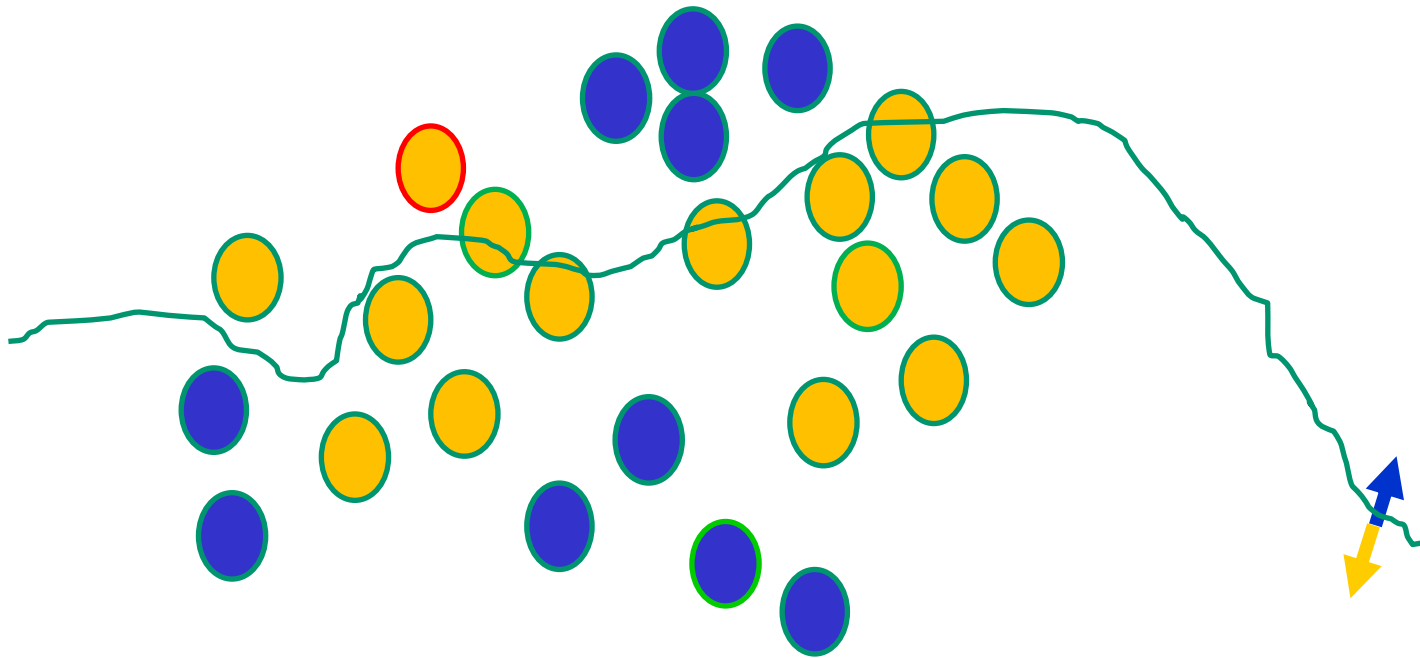
Present a training instance / adjust the weights



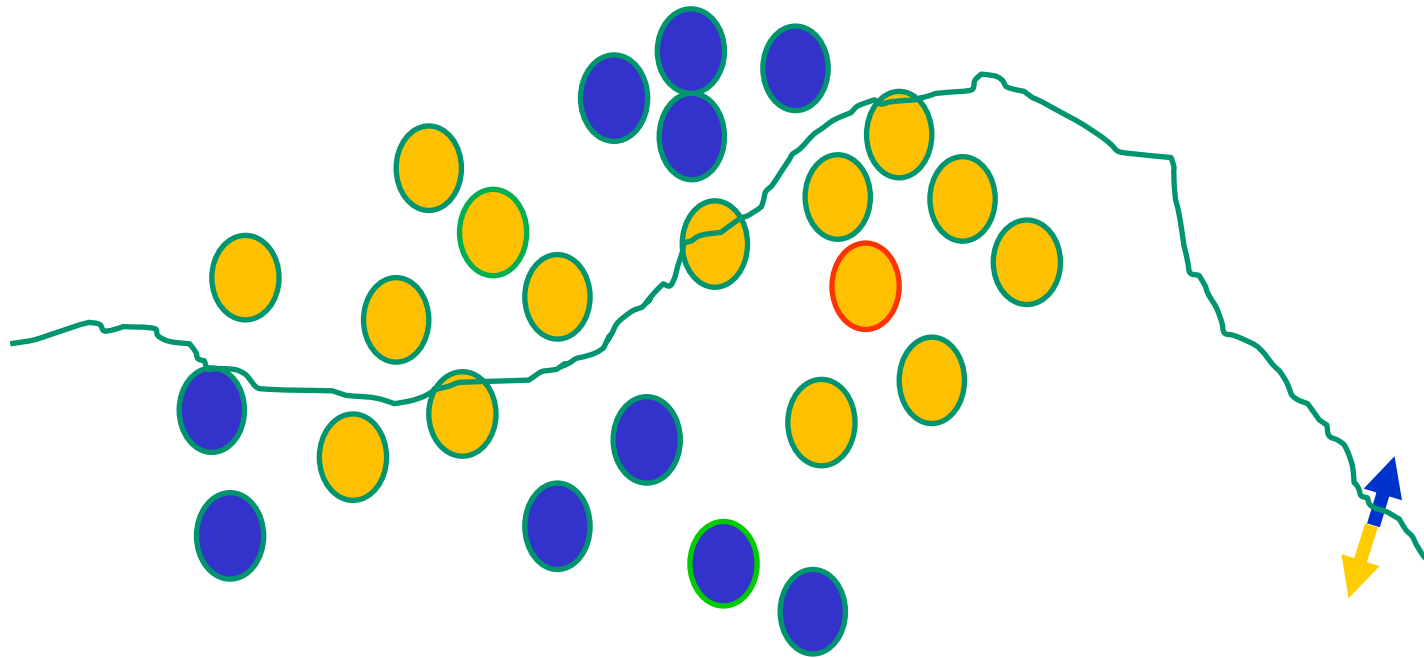
Present a training instance / adjust the weights



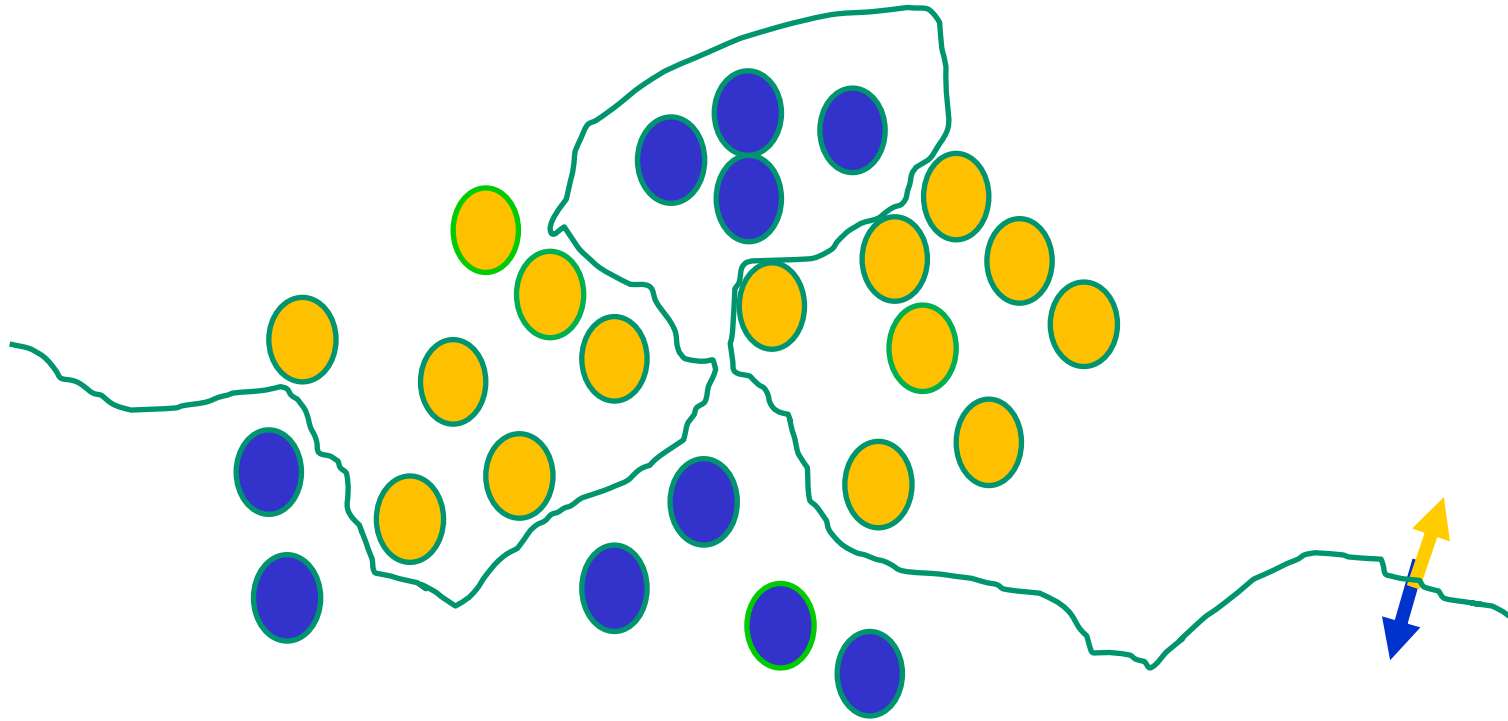
Present a training instance / adjust the weights



Present a training instance / adjust the weights



Eventually ing instance / adjust the weights



Deep Learning architectures

- Recurrent Neural Networks (RNN)
- Long Short Term Memory (LSTM)
- Auto-encoder
- Macchina di Boltzmann
- Deep Belief Network (DBN)
- Deep Convolutional Neural Network (DCNN)
- Generative Adversarial Network (GAN)
- Transformer

Large Language Model (LLM): cos'è e come funziona

- Un LLM è una rete neurale profonda addestrata su enormi corpus testuali per predire il token successivo; da questa predizione emergono capacità di ragionamento, traduzione, sintesi e generazione di testo coerente
- **Transformer:** introdotto da Vaswani et al. (2017): il meccanismo di self-attention permette a ogni token di relazionarsi con tutti gli altri nella sequenza, sostituendo le RNN e abilitando il parallelismo massivo su GPU
- **Token embedding:** ogni parola o sottoparola è un vettore denso (768–4096 dimensioni); la distanza geometrica tra vettori cattura relazioni semantiche (re – uomo + donna $\approx$ regina)
- **Pre-training e fine-tuning:** il pre-training non supervisionato apprende la struttura del linguaggio; il fine-tuning su esempi etichettati e il feedback umano allineano il modello agli obiettivi desiderati

Transformer architecture

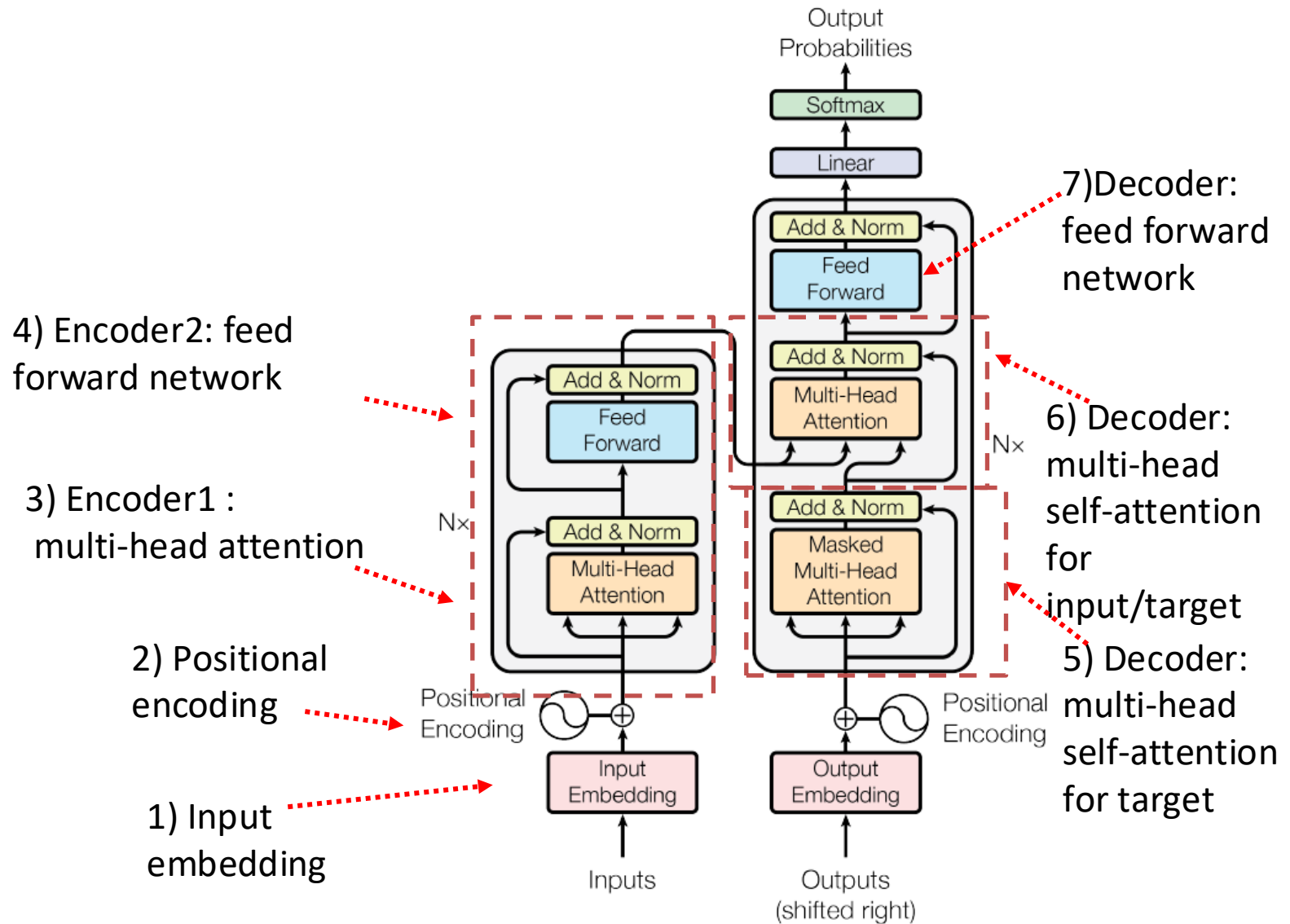


Figure 1: The Transformer - model architecture.

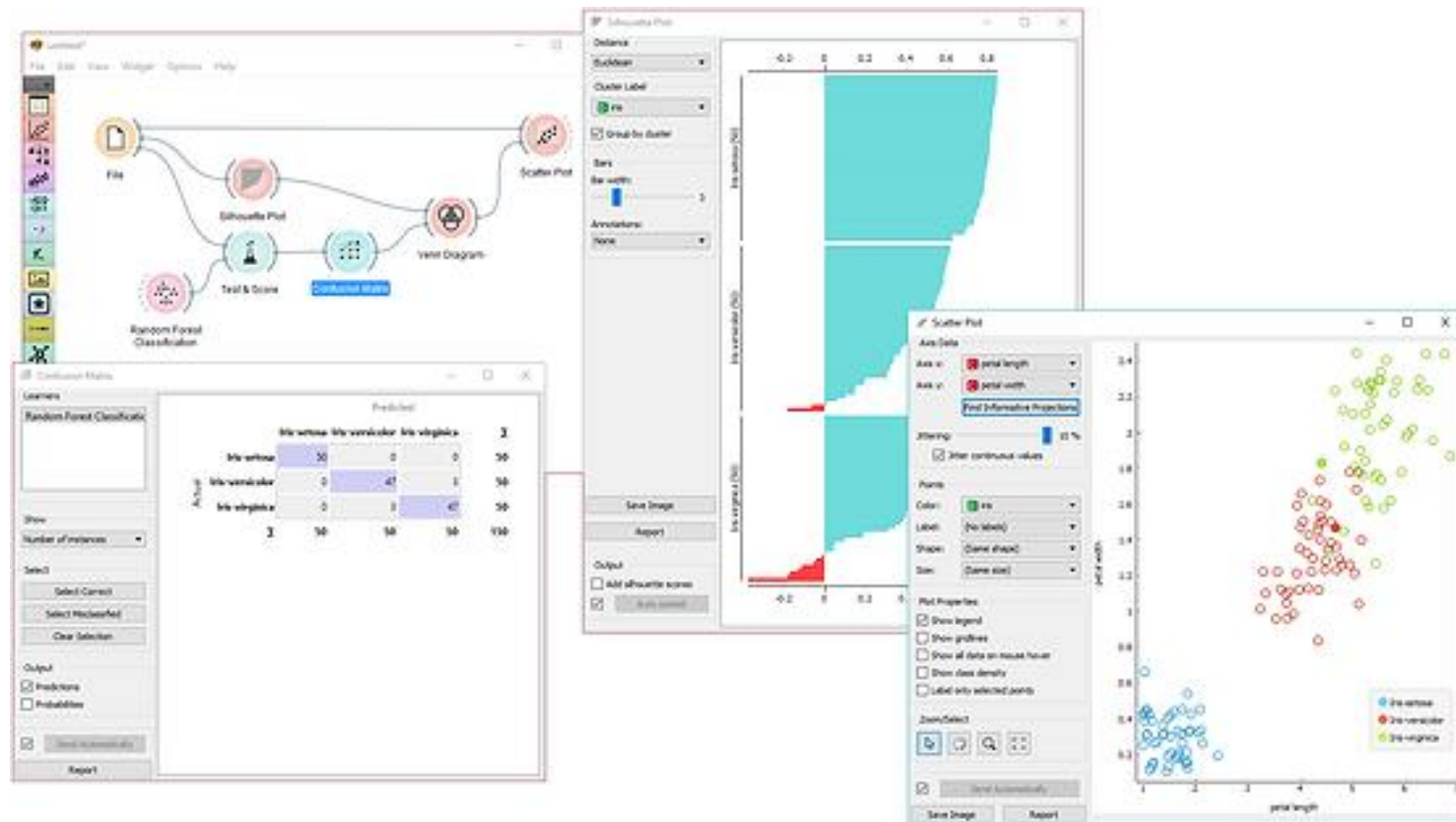
LLM: modelli principali e scale di parametri

- **GPT-3 / GPT-4 (OpenAI):** GPT-3 (175 miliardi di parametri, 2020) ha mostrato l'emergenza di capacità non previste; GPT-4 (2023) è multimodale e supera il 90° percentile degli esami professionali umani
- **LLaMA 3 (Meta AI):** LLaMA 3 (8B–405B parametri) è scaricabile e usabile localmente su hardware consumer senza inviare dati al cloud; punto di partenza per fine-tuning su domini specializzati
- **Claude / Gemini / Mistral / Phi:** Claude (Anthropic), Gemini (Google, multimodale nativo), Mistral/Mixtral (mixture-of-experts, open-weight), Phi-3 (Microsoft, compatto): ecosistema ricco e in rapida evoluzione
- **Scaling laws:** le prestazioni crescono in modo prevedibile con parametri, dati e compute (Kaplan et al., 2020); ogni ordine di grandezza sblocca nuove capacità non presenti nei modelli più piccoli

Orange: machine learning visuale senza scrivere codice

- Orange (<https://orangedatamining.com>) è un ambiente open-source di data mining e ML a programmazione visuale: il flusso di lavoro si costruisce collegando widget grafici con il mouse, senza scrivere codice
- **Paradigma a widget:** ogni widget è un blocco funzionale (dati, pre-processing, modello, valutazione, visualizzazione); le connessioni definiscono la pipeline; i risultati si aggiornano in tempo reale al variare dei parametri
- **Installazione e architettura:** scritto in Python, si installa con pip `install orange3` o tramite installer grafico per Windows/macOS/Linux; add-on opzionali estendono le funzionalità (testo, bioinformatica, spettroscopia, reti)
- **Algoritmi disponibili:** alberi di decisione, Random Forest, SVM, Naive Bayes, Rete neurale feedforward, regressione logistica e clustering sono tutti accessibili via widget grafici pronti all'uso, senza scrivere codice

Orange: ambiente grafico



Orange: workflow tipico e funzionalità chiave

- **Caricamento dati:** file CSV, Excel, URL o database; il widget File e il widget SQL caricano dataset di qualsiasi dimensione; il widget Data Table mostra i dati in forma tabellare con statistiche descrittive immediate
- **Pre-processing visuale:** widget Select Columns, Normalize, Impute (valori mancanti), Discretize, Feature Constructor; tutto configurabile con trascinamento e menù a tendina, senza scrivere codice
- **Valutazione e confronto modelli:** widget Test & Score esegue k-fold cross-validation su più modelli e confronta accuracy, AUC, F1, precision e recall; il widget Confusion Matrix mostra gli errori per classe
- **Visualizzazione interattiva:** scatter plot, box plot, heat map, t-SNE e MDS per la visualizzazione ad alta dimensionalità; selezionando punti nel grafico si filtrano automaticamente le righe del dataset

Python per il Machine Learning: perché è lo standard

- Python è il linguaggio dominante nel ML: sintassi leggibile, librerie mature e comunità scientifica globale ne rendono l'apprendimento accessibile anche senza un background informatico
- **NumPy / SciPy / Pandas:** arrays N-dimensionali veloci (NumPy), algoritmi scientifici (SciPy) e DataFrame per la gestione e pulizia dei dati (Pandas): la base comune di tutto l'ecosistema ML Python
- **Matplotlib / Seaborn:** grafici statici e interattivi essenziali per l'analisi esplorativa dei dati (EDA); Seaborn aggiunge visualizzazioni statistiche pronte all'uso sopra Matplotlib
- **Jupyter Notebook / Lab:** ambiente interattivo che unisce codice, output e documentazione in un unico documento riproducibile; standard nelle pubblicazioni scientifiche di ML e nei corsi universitari di data science

Python per il ML: librerie principali

- **scikit-learn:** libreria di riferimento per gli algoritmi classici: alberi di decisione, SVM, Naive Bayes, Random Forest, k-NN, regressione, clustering; API fit/predict uniforme e pipeline di pre-processing integrate
- **TensorFlow / Keras:** framework Google con Keras come API ad alto livello; costruisce e addestra reti neurali in poche righe; supporto nativo GPU/TPU e deployment in produzione (TensorFlow Serving, TFLite)
- **PyTorch:** libreria Meta AI con differenziazione automatica; preferita dalla ricerca accademica per flessibilità e debugging intuitivo; alla base di GPT, BERT, Whisper e molti modelli open-source
- **Ensemble boosting:** gradient boosting su alberi (XGBoost, LightGBM, CatBoost): vincitori storici di Kaggle su dati tabulari, spesso superiori alle reti neurali su dataset strutturati di piccole dimensioni

Python per la radio: elaborazione del segnale e SDR

- **GNU Radio:** ambiente di signal processing di tipo dataflow; i blocchi permettono di costruire ricevitori SDR, analizzatori di spettro e demodulatori in Python; interfaccia grafica senza scrivere codice (<https://www.gnuradio.org>)
- **PySDR / SoapySDR:** PySDR è una libreria Python open source creata da Marc Lichtman (ricercatore e docente universitario negli USA) con un duplice scopo: essere al tempo stesso un manuale didattico interattivo sulla SDR e il DSP, e una libreria pratica per lavorare con segnali radio in Python; SoapySDR è un'API unificata che permette di controllare qualsiasi ricevitore o trasmettitore SDR con lo stesso codice Python, indipendentemente dal produttore e dal modello. (<https://pysdr.org>
<https://github.com/pothosware/SoapySDR/wiki>)
- **commpy:** CommPy è una libreria Python open source per le comunicazioni digitali — fornisce implementazioni di algoritmi di teoria delle comunicazioni che non sono presenti in NumPy/SciPy (<https://commpy.readthedocs.io/en/latest/>)
- **scipy.signal:** filtri, FFT, spettrogrammi, resampling dentro SciPy (<https://docs.scipy.org/doc/scipy/reference/signal.html>)

Python per la radio: Esempi di applicazione ML in campo radioamatoriale

- **Automatic Modulation Recognition (AMR):** dataset RadioML, sviluppato originariamente dal gruppo di Timothy O'Shea e successivamente distribuito da DeepSig (La versione RadioML 2018.01A contiene circa 2,56 milioni di esempi I/Q appartenenti a 24 tipi di modulazione)
- **Propagazione e DX analytics:** Stimare MUF, path loss e condizioni ionosferiche in tempo reale. Il gruppo HamSCI (Ham Radio Science Citizen Investigation) ha pubblicato modelli LSTM per la previsione degli indici ionosferici usando dati WSPR come proxy (<https://github.com/HamSCI/HamSCI/tree/master>) (<https://github.com/Mertchan/IonoBench>)
- **Noise reduction:** DeepFilterNet (<https://github.com/Rikorose/DeepFilterNet>)
- **Progettazione circuitale:** Flux (flux.ai)

LLM per la generazione di codice: come funziona

- i moderni LLM (GPT-4, Claude, Gemini, CodeLlama) sono addestrati su miliardi di token di codice sorgente (GitHub, Stack Overflow, documentazione); la predizione token-per-token si applica al codice come al testo naturale
- **Prompt → codice → spiegazione:** si descrive in linguaggio naturale cosa si vuole; il modello genera codice funzionante, lo spiega e lo corregge su richiesta; il dialogo avviene senza dover conoscere la sintassi
- **Modelli specializzati nel codice:** GPT-4o e Claude Sonnet eccellono su Python/ML; CodeLlama 70B e DeepSeek-Coder sono open-weight, eseguibili in locale via Ollama; GitHub Copilot si integra nell'editor (VS Code, JetBrains)
- **Accuratezza e limiti:** errori sintattici rari, errori logici possibili su algoritmi complessi; il codice generato va sempre revisionato e testato; su task noti l'accuratezza supera il 90% al primo tentativo

LLM per il codice: workflow pratico in Python e ML

- **Generazione da zero:** "Scrivi un classificatore SVM con scikit-learn su un CSV con colonne SNR, BW, kurtosis, label; stampa accuracy e matrice di confusione."; il LLM produce codice completo e commentato in pochi secondi
- **Debug guidato:** si incolla il traceback nel prompt: "ottengo questo errore: [traceback]; come lo risolvo?"; il LLM individua la causa radice e propone la correzione; efficace anche per bug sottili (shape mismatch, dtype errato)
- **Refactoring e adattamento:** "Trasforma questo codice NumPy in PyTorch" oppure "Aggiungi normalizzazione MinMax prima del fit"; il modello modifica il codice esistente senza riscriverlo da zero

LLM per il codice radioamatoriale: esempi pratici

- **Script SDR con PySDR:** "Scrivi uno script Python con PySDR che acquisisce 2 secondi di campioni I/Q da un RTL-SDR a 144.8 MHz, calcola lo spettrogramma e lo salva come PNG."; in pochi secondi si ottiene codice funzionante
- **Classificatore di modulazione:** "Genera un classificatore scikit-learn che prende SNR, varianza di fase e kurtosis e predice AM, FM o SSB; usa Random Forest e stampa feature importance."; il LLM produce pipeline completa con splitting train/test
- **Analisi log QSO in ADIF:** "Leggi un file ADIF, estrai banda, orario UTC e paese, e disegna un grafico a barre dei QSO per banda con matplotlib."; il LLM conosce il formato ADIF e genera parser e grafico in un unico blocco
- **Conversione e traduzione di codice:** traduce da GNU Radio Companion in Python puro, converte script Octave/MATLAB di DSP in NumPy/SciPy e adatta esempi tra linguaggi diversi

Esempio di analisi QSO

Prompt: Read the attached ADIF file.
Extract time, Band, Mode, and Country
information. Plot separate bar plots of
QSOs per time range, band, mode, and
country by using matplotlib.

