Cheatsheet di Machine Learning Classico

Corso di Machine Learning per la Fisica - AA 2024/25

Contents

1 Machine Learning dal punto di vista statistico

Consideriamo un **training set**:

$$T = \{(x_1, y_1), \ldots, (x_n, y_n)\} \tag{1}$$

dove ogni coppia (x_i, y_i) è estratta indipendentemente (i.i.d.) da una distribuzione di probabilità $p(x, y)$ fissata ma ignota.

1.1 Apprendimento supervisionato

Obiettivo: usare T per costruire una funzione f tale che, per un nuovo input x, produca una predizione $\hat{y}$ del vero output y:

$$\hat{y} = f(x) \tag{2}$$

1.2 Distribuzioni di probabilità

La relazione tra input ed output si descrive con:

$$p(x, y) = p(y|x)p(x) \tag{3}$$

2 Funzione di costo (Loss)

Per valutare la bontà di una funzione predittiva si introduce una **funzione di costo** $L(y, f(x))$, che misura la discrepanza tra predizione e realtà.

2.1 Esempi di funzioni di loss

- **0-1 loss**
- **Errore assoluto medio (MAE)**
- **Errore quadratico medio (MSE)**

2.2 Rischio statistico

$$R(f) = \mathbb{E}[L(y, f(x))] = \int L(y, f(x))dp(x, y) \tag{4}$$

3 Rischio empirico e apprendimento come approssimazione

Poiché $p(x, y)$ è ignota, introduciamo il **rischio empirico**:

$$\hat{R}(f) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{5}$$

3.1 Errori principali

- **Errore di approssimazione**
- **Errore di stima**

4 Teoria bayesiana della decisione

Definiamo classi ω_j, decisioni α_i, probabilità a priori $P(\omega_j)$, funzione di costo $\lambda(\alpha_i|\omega_j)$.

4.1 Teorema di Bayes

$$P(\omega_j|x) = \frac{P(x|\omega_j)P(\omega_j)}{p(x)} \tag{6}$$

4.2 Classificatore bayesiano

$$\alpha(x) = \arg\max_j P(\omega_j|x) \tag{7}$$

5 Likelihood Ratio Test per 2 classi

Se errori hanno costi diversi, si confrontano i rischi condizionali ottenendo la regola del **Likelihood Ratio Test**.

$$\frac{P(x|\omega_1)}{P(x|\omega_2)} > \gamma \quad \Rightarrow \quad x \in \omega_1 \tag{8}$$

6 Stima delle PDF

6.1 Approccio parametrico

Si assume una forma funzionale (es. gaussiana) e si stimano i parametri tramite Maximum Likelihood.

6.2 Approccio non parametrico

Si stimano le PDF dai dati senza ipotesi forti: istogrammi, metodo dei kernel, k-NN.

7 Naive Bayes

Assume indipendenza tra feature:

$$P(x|\omega_i) = \prod_k P(x_k|\omega_i) \tag{9}$$

Vantaggi: semplice, intuitivo. Svantaggi: ignora correlazioni.

8 K-Nearest Neighbors (k-NN)

8.1 Definizione

Classificatore non parametrico che assegna a x la classe più frequente tra i k vicini.

8.2 Formula

$$p(\omega_i|x) \approx \frac{k_i}{k} \tag{10}$$

8.3 Esempio

Con $k = 3$, se due vicini sono di classe A e uno di classe B, allora x è assegnato alla classe A.

9 Scikit-learn

9.1 Naive Bayes in Python

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
accuracy = model.score(X_test, y_test)
```

9.2 k-NN in Python

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

10 Modelli lineari e Perceptron

La funzione discriminante lineare:

$$g(x) = w_0 + \sum_{i=1}^{D} w_i x_i \tag{11}$$

Decisione: ω_1 se $g(x) > 0$, altrimenti ω_2.

10.1 Interpretazione geometrica

L'iperpiano $g(x) = 0$ separa lo spazio in due regioni. w è ortogonale al piano.

11 Algoritmi non metrici: Alberi di decisione

Classificatori come k-NN si basano su nozioni metriche di distanza. Esistono però casi in cui non c'è una nozione naturale di distanza tra osservazioni (es. attributi categoriali come colore, forma, ecc.).

Decision Tree: modello non parametrico, semplice, interpretabile, basato su partizioni ricorsive del dataset.

12 Alberi di decisioni binarie

Un **binary decision tree** è un grafo orientato senza cicli, in cui:

- Nodo radice (root)
- Nodi interni con test binari
- Foglie che corrispondono alle classi finali

Esempio: classificare un frutto tramite attributi come colore, forma, sapore, grandezza.

13 Costruzione di un albero

1. Partire dal training set
2. Suddividere ricorsivamente il campione con test su feature
3. Fermarsi quando le foglie sono pure o viene raggiunto un criterio di stop

13.1 Misure di purezza

- Entropia: $H = -\sum_j p(\omega_j) \log_2 p(\omega_j)$
- Indice di Gini: $G = 1 - \sum_j p^2(\omega_j)$

14 Criteri di stop

Si possono usare:

- Numero minimo di eventi per foglia

- Profondità massima
- Miglioramento minimo nella purezza

Questi criteri funzionano da **regolarizzatori** per evitare overfitting.

15 K-fold Cross Validation

Tecnica di resampling per stimare prestazioni e ottimizzare iperparametri. Il dataset è suddiviso in K parti: si addestra su $K-1$ e si valida sulla rimanente. Il punteggio finale è la media sui K test.

16 Decision tree e superfici di separazione

Geometricamente un decision tree suddivide lo spazio delle feature in iperrettangoli. Con sufficiente profondità, può approssimare superfici complesse, ma tende a overfittare.

17 Pruning

Per ridurre l'overfitting si applica la **potatura**, eliminando nodi che non migliorano la performance complessiva. Due algoritmi diffusi:

- **Expected error pruning**
- **Cost complexity pruning**

18 Ensemble di classificatori

Un singolo albero è debole, ma combinandone molti si possono ottenere classificatori potenti. Concetti chiave: **Bias-Variance tradeoff**. Gli ensemble riducono la varianza.

19 Bagging e Random Forests

Bagging (Bootstrap Aggregation):

1. Generare M campioni bootstrap dal dataset
2. Addestrare M alberi

3. Predizione finale: media o voto di maggioranza

Random Forest: bagging applicato ad alberi di decisione, con scelta casuale delle feature.

20 Boosting

Idea: costruire sequenzialmente alberi, dando più peso agli esempi mal classificati in precedenza.

20.1 AdaBoost

Algoritmo:

1. Inizializzare pesi uguali per tutti i dati
2. Iterativamente addestrare alberi su campioni pesati
3. Aumentare peso dei dati mal classificati
4. Predizione finale: voto pesato

20.2 Gradient Boosting

Concetto: aggiornare il modello aggiungendo nuovi alberi che fittano i **residui** del modello precedente. Equivalente a una discesa del gradiente sulla funzione di loss.

21 xGBoost e varianti

xGBoost: implementazione ottimizzata di gradient boosting, con regolarizzazione, shrinkage, parallelizzazione e supporto GPU. Alternative: LightGBM, CatBoost.

22 Scikit-learn: esempi pratici

22.1 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
```

22.2 Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

22.3 AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier

adb = AdaBoostClassifier(n_estimators=200)
adb.fit(X_train, y_train)
y_pred = adb.predict(X_test)
```

23 Visualizzazione dei dati e riduzione dimensionale

La fase di esplorazione e visualizzazione dei dati è cruciale in qualsiasi progetto di Machine Learning (ML). Saltare questo passo significa rischiare di utilizzare strategie sbagliate e non ottenere un modello soddisfacente.

Esempio: dataset differenti possono avere statistiche di riepilogo (*summary statistics*) identiche — come medie, varianze, correlazioni, parametri di fit lineare — ma presentare comportamenti molto diversi se analizzati visivamente. Solo l'analisi visuale permette di individuare artefatti, rumore, outliers, e scegliere il modello appropriato.

23.1 Obiettivi della riduzione dimensionale

- Visualizzare dati multidimensionali.
- Ottenere rappresentazioni più efficaci dei dati (*data compression*).
- Rimuovere ridondanze e mantenere le informazioni più rilevanti.
- Ridurre il costo computazionale dei modelli.

23.2 Problemi in alta dimensione: Curse of Dimensionality

La geometria in spazi ad alta dimensione presenta proprietà contro-intuitive. Ad esempio, punti uniformemente distribuiti in un ipercubo $C = [-l/2, l/2]^D$ tendono a concentrarsi vicino ai bordi.

Consideriamo un'ipersfera S di raggio $l/2$ centrata nell'origine e contenuta in C. La probabilità che un punto x estratto casualmente da C sia contenuto in S è approssimata dal rapporto dei volumi:

$$p(x \in S) \propto \frac{\left(\frac{l}{2}\right)^D}{l^D} = \frac{1}{2^D}. \tag{12}$$

Per $D \to \infty$, $p \to 0$ con velocità esponenziale: i punti si addensano negli angoli dell'ipercubo.

23.3 Crowding Problem

Quando si riduce la dimensione cercando di preservare le distanze relative tra punti, se si scende sotto la dimensionalità intrinseca dei dati si rischia il

sovraffollamento. Esempio: il *swiss roll*, distribuito in 3D, può essere rappresentato in 2D senza problemi. Ma se 3 punti equidistanti in 2D vengono rappresentati in 1D, le distanze relative collassano.

Soluzioni:

- Rilasciare i vincoli di similitudine.
- Usare algoritmi come **t-SNE**, **UMAP**.

Curse of Dimensionality e Machine Learning. Nonostante la *curse of dimensionality*, il machine learning funziona anche con dati ad altissima dimensione per due ragioni principali: (i) lo spazio latente che descrive il fenomeno ha in genere dimensione molto inferiore rispetto allo spazio delle feature misurate; (ii) i dati risultano localmente mediati (*smoothed*), per cui variazioni nelle feature producono solo piccoli cambiamenti nel target. Questo è analogo a quanto avviene in fisica statistica, dove sistemi con molti gradi di libertà possono essere descritti da pochi parametri d'ordine (es. variabili termodinamiche per un gas o la magnetizzazione nel modello di Ising).

24 Algoritmi di riduzione dimensionale

Due categorie principali:

1. **Feature Elimination**: eliminazione di feature poco significative.
   - Vantaggi: semplicità, mantenuta interpretabilità.
   - Svantaggi: perdita di informazione.
2. **Feature Extraction**: costruzione di nuove feature come combinazioni lineari/non-lineari di quelle originali.
   - Vantaggi: preservano parte significativa dell'informazione.
   - Svantaggi: minore trasparenza, perdita di significato fisico.

24.1 Proiezioni casuali

L'approccio più semplice: proiezioni casuali. Spesso però le strutture interessanti possono andare perse. Per questo si preferiscono metodi come PCA, LDA, Isomap, t-SNE, UMAP.

25 PCA - Principal Component Analysis

25.1 Definizione

La PCA è un metodo lineare che trova le direzioni di massima varianza nei dati:

1. Trasformazione ortogonale dei dati.
2. Ordinamento delle direzioni per varianza.
3. Selezione delle prime $d < D$ direzioni come rappresentative, scartando le altre come rumore.

PCA: spiegazione intuitiva. La *Principal Component Analysis* si basa su quattro idee chiave: (i) le feature rilevanti sono quelle che spiegano la maggiore varianza dei dati; (ii) la matrice di covarianza Σ misura varianze e correlazioni tra le feature; (iii) gli autovettori di Σ individuano le direzioni principali di dispersione dei dati; (iv) i corrispondenti autovalori quantificano l'importanza relativa di tali direzioni. L'idea è quindi di proiettare i dati sugli autovettori associati agli autovalori maggiori, mantenendo solo le d direzioni che spiegano una quota significativa della varianza totale.

25.2 Formalismo matematico

Dati N punti $\{x_i\}$ con vettore $x_i \in \mathbb{R}^D$, e matrice dei dati $X \in \mathbb{R}^{N \times D}$:

$$\Sigma(X) = \frac{1}{N-1} X^T X \tag{13}$$

dove Σ è la matrice di covarianza. Si applica la Singular Value Decomposition:

$$X = USV^T \quad \Rightarrow \quad \Sigma(X) = V \Lambda V^T \tag{14}$$

con Λ matrice diagonale degli autovalori $\lambda_i = s_i^2/(N-1)$. Le colonne di V sono le direzioni principali.

25.3 Proiezione

I dati proiettati nello spazio delle prime d componenti principali sono:

$$\tilde{X} = X\tilde{V}. \tag{15}$$

26 LDA - Linear Discriminant Analysis di Fisher

La PCA non è ottimale per classificazione. La LDA estende l'idea massimizzando la separazione tra classi.

26.1 Criterio di Fisher

Si cerca un vettore w che massimizza:

$$J(w) = \frac{|w^T(m_1 - m_2)|^2}{w^T S_w w} \tag{16}$$

dove m_i è la media della classe i, e S_w è la somma delle matrici di covarianza intra-classe. La soluzione ottimale è:

$$w = S_w^{-1}(m_1 - m_2). \tag{17}$$

27 Metodi non lineari

27.1 Isomap

Mantiene le distanze geodetiche tra punti costruendo un grafo k-NN e applicando MDS per ottenere un embedding ridotto. Funzione di loss:

$$Stress(x_1, \ldots, x_N) = \sum_{i \neq j} (d_{ij} - \|x_i - x_j\|)^2. \tag{18}$$

27.2 t-SNE

Metodo non lineare che preserva le strutture locali. Minimizza la divergenza di Kullback-Leibler tra distribuzioni nello spazio originale e ridotto:

$$L(y) = D_{KL}(p||q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{19}$$

27.3 UMAP

Simile a t-SNE, ma più veloce e migliore nel preservare la struttura globale. Costruisce un *fuzzy simplicial complex* e ottimizza una funzione di cross-entropia tra grafi originali e ridotti.

28 Autoencoder (ANN)

Un autoencoder è una rete neurale non supervisionata che apprende una funzione identità compressa.

$$z = f_e(x) = \phi(W_1 x + b_1) \tag{20}$$
$$x' = f_d(z) = \phi(W_2 z + b_2) \tag{21}$$

con obiettivo $x' \approx x$.

29 Esempi pratici

29.1 Scikit-learn

- PCA: `from sklearn.decomposition import PCA`
- Isomap: `from sklearn.manifold import Isomap`
- t-SNE: `from sklearn.manifold import TSNE`

29.2 UMAP

```
import umap.umap_ as umap
reducer = umap.UMAP(n_neighbors=15, min_dist=0.1)
embedding = reducer.fit_transform(X)
```

30 Artificial Neural Networks (ANN)

Le **Artificial Neural Networks** (ANN) rappresentano l'approccio più popolare al Machine Learning e costituiscono la base delle moderne tecniche di Deep Learning (DL).

Un'ANN è un modello matematico capace di approssimare con precisione arbitraria una funzione generica:

$$f : \mathbb{R}^n \to \mathbb{R}^m, \quad y = f(x)$$

con implementazione tramite neuroni artificiali connessi da pesi.

Caratteristiche principali

- Architettura: gruppo interconnesso di unità di calcolo elementari (neuroni).
- Approccio computazionale: connessionista (azioni eseguite collettivamente e in parallelo).
- Apprendimento: la rete si adatta durante l'addestramento cambiando i pesi sulla base dei dati.
- Supporta risposte non lineari e apprendimento gerarchico delle rappresentazioni.

—

31 Neurone artificiale

Il **neurone artificiale** (McCulloch-Pitts 1943, Rosenblatt 1962) si basa sul modello:

$$z = w_0 + \sum_{i=1}^{n} w_i x_i = w_0 + w^T x, \qquad \hat{y} = a(z)$$

dove $a(z)$ è la funzione di attivazione.

Note pratiche

- Con una sola unità a funzione *step*, un neurone può separare classi linearmente separabili.

- L'estensione a più layer con attivazioni non lineari permette di approssimare ipersuperfici complesse.
- Esempio: un MLP con ReLU può risolvere il problema dell'XOR, impossibile per un singolo Perceptron.

—

32 Universal Approximation Theorem

Enunciato: per ogni funzione Lebesgue p-integrabile $f : \mathbb{R}^n \to \mathbb{R}^m$ e $\varepsilon > 0$, esiste un'ANN con attivazioni ReLU e larghezza limitata tale che:

$$\int_{\mathbb{R}^n} \|f(x) - F(x)\|_p \, dx < \varepsilon$$

Non è possibile approssimare con reti più strette di $d_m = \max\{n + 1, m\}$.

Spiegazione intuitiva

- Un neurone con sigmoide: $\sigma(z) = \frac{1}{1+e^{-z}}$ può approssimare funzioni a gradino.
- Giocando con pesi e bias si possono costruire funzioni a *step* in qualunque punto.
- Combinando più neuroni si ottengono approssimazioni a rettangoli $\to$ qualsiasi funzione continua può essere approssimata.
- In input multidimensionale, si ottiene una generalizzazione simile alle stime non parametriche.

—

33 ANN e reti neurali biologiche

- Analogia suggestiva ma non identica: neuroni biologici sono molto più complessi.
- Nei modelli biologici i dendriti eseguono calcoli non lineari avanzati.
- Le sinapsi sono dinamiche e non semplici pesi.

- Le reti biologiche sono reti spiking (inviano segnali solo al superamento di una soglia).

—

34 Multilayer Perceptron (MLP)

Un **MLP** è una feed-forward ANN con grafico aciclico diretto:

$$z_i = w_{0i}^{(1)} + \sum_{j=1}^{m} x_j w_{ji}^{(1)}, \qquad \hat{y}_i = a^{(2)}\left(w_{0i}^{(2)} + \sum_{j=1}^{d_1} a^{(1)}(z_j) w_{ji}^{(2)}\right)$$

Architettura

- Layers: input, hidden (1..K), output.
- Connessioni solo verso layer successivi.
- Densi: tutti i neuroni di un layer connessi al successivo.
- Parametri fondamentali: pesi w, topologia (#layer, #neuroni), funzioni di attivazione.

—

35 Funzioni di attivazione

Per gli hidden layer

- **Sigmoide**: $a(z) = \frac{1}{1+e^{-z}}$, evita valori estremi ma soffre di vanishing gradient. preferibilmente da non usare con reti con molti layers:
- **Tanh**: $a(z) = \tanh(z)$, centrata in zero, utile in RNN.
- **ReLU**: $a(z) = \max(0, z)$, la più usata, veloce e stabile.
- Varianti: LeakyReLU, ELU, Maxout. esempio: maxout ha tutti pregi di LeakyReLU/ELU e le generalizza ... a spese di due volte il numero degli iper-parametri della rete ...

Per l'output layer

- **Sigmoide**: classificazione binaria con singolo neurone di output o multilabel (più classi mutualmente inclusive.
- **Softmax**: multi-classe esclusiva, output $\in [0,1]^n$ come probabilità.
- **Identità**: regressione.

—

36 Addestramento di una ANN

Addestrare una rete significa ottimizzare i pesi minimizzando una funzione di loss.

Esempio: regressione supervisionata

$$L(w,T) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - f_w(x^{(i)})\right)^2$$

- Inizializzazione pesi: piccoli valori random (es. $\mathcal{N}(0,\sigma)$ o $\mathcal{U}[-\epsilon,\epsilon]$).
- Ottimizzazione: stochastic gradient descent (SGD) con backpropagation.
- Iperparametri: #layer, learning rate, batch size, ottimizzati via grid/random search, AutoML, ecc. ottimizzati a mano in maniera euristica

Aggiornamento pesi via SGD

$$w^{(k+1)} = w^{(k)} - \eta \nabla_w L(T|w)$$

dove η è il learning rate. L'update avviene su mini-batch T_i di dimensione Nb.

—

37 Loss Functions

Per regressione

- **MSE**: $\frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$.

- **MAE**: $\frac{1}{N}\sum_i |y_i - \hat{y}_i|$.
- **Huber**: robusta agli outliers, combina MSE e MAE.

Per classificazione

- **Binary Cross-Entropy**:

$$L = -\frac{1}{N}\sum_i \left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right]$$

- **Cross-Entropy Loss**: estensione multi-classe con softmax.

—

38 Curve di apprendimento

Durante il training:

- All'inizio: loss alta.
- Col progredire delle epoche: loss decresce fino a plateau.
- Le curve (training vs validation) mostrano overfitting o underfitting.

—

39 PyTorch

PyTorch è una libreria ottimizzata per DL:

- Operazioni tra tensori su CPU e GPU.
- API per layers, loss, ottimizzatori, modelli.
- Supporta backpropagation automatica (`torch.autograd`).

`https://pytorch.org/tutorials/`

Esempio MLP in PyTorch

```
import torch
import torch.nn as nn
import torch.optim as optim

class SimpleMLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(SimpleMLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.out = nn.Sigmoid()  # es. classificazione binaria

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return self.out(x)

model = SimpleMLP(input_dim=10, hidden_dim=32, output_dim=1)
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Training loop

```
for epoch in range(epochs):
    model.train()
    for xb, yb in train_loader:
        optimizer.zero_grad()
        outputs = model(xb)
        loss = criterion(outputs, yb)
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
        for xb, yb in val_loader:
            val_outputs = model(xb)
            # calcolo della validation loss
```

40 Inizializzazione dei pesi

- Pesi iniziali rompono la simmetria.
- Bias inizializzati a 0, pesi distribuiti random $\sim \mathcal{N}(0, \sigma)$ o uniformi.
- Esempi: **Glorot uniform**, **He initialization**.

Inizializzazione dei pesi. L'inizializzazione dei pesi deve rompere la simmetria tra neuroni hidden connessi agli stessi input, altrimenti verrebbero aggiornati in modo identico. La regola standard prevede biases fissati (tipicamente a zero) e pesi inizializzati casualmente attorno a zero (ad es. $N(0, \sigma)$ o $U[-\epsilon, \epsilon]$). Valori troppo grandi di σ riducono le ridondanze ma possono causare instabilità numeriche (es. *exploding gradients*). Le euristiche più comuni includono: $N(0, \frac{1}{m})$, $U[-\frac{1}{m}, \frac{1}{m}]$ per layer densi, e la Glorot uniform $U\left[-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right]$.

41 Introduzione

Questi appunti sono una trascrizione fedele e commentata delle slide del corso di *Metodi di Intelligenza Artificiale per la Fisica* (Applicazioni fisiche al Machine Learning). Ogni sezione corrisponde a una slide, con spiegazioni dettagliate delle formule e del contesto.

42 Back-propagation: Concetti Generali

Il training di una rete neurale artificiale (NN) avviene in due fasi che si ripetono a ogni iterazione:

42.1 Forward phase

- I pesi sono fissati.
- Il vettore di input viene propagato layer per layer fino ai neuroni di output.
- Questo segnale propagato viene detto **function signal**.

42.2 Backward phase

- Si calcola l'errore Δ confrontando l'output con il target.
- L'errore viene propagato indietro, ancora layer per layer.
- Questo segnale viene detto **error signal**.

Ogni neurone (sia hidden che di output) riceve e confronta i segnali di funzione e di errore.

La **back-propagation** consiste in una semplificazione del calcolo del gradiente ottenuta applicando ricorsivamente la **chain rule** (regola di derivazione di funzioni composte). Questo viene implementato come un grafo computazionale su cui si applica la differenziazione automatica.

42.3 Nota pratica

Per aggiornare i pesi di tutti i layer della rete è necessario calcolare il gradiente di funzioni complesse e non convesse rispetto a ciascun peso. La

back-propagation rende questo procedimento efficiente.

Infatti, con la backprop, il calcolo del gradiente della Loss risulta veloce quanto calcolare la Loss stessa:

$$t(\nabla_w L) \simeq t(L) \tag{22}$$

43 Back-propagation: Neuroni di Output

Consideriamo il j-mo neurone di output, e indichiamo con n l'n-mo evento del dataset.

43.1 Definizioni fondamentali

$$\xi_j(n) = \sum_{i=0}^{m} w_{ji}(n)\, y_i(n) \tag{23}$$

$$y_j(n) = \varphi(\xi_j(n)) \tag{24}$$

$$e_j(n) = Y_{\text{true}}(n) - y_j(n) \tag{25}$$

- $\xi_j(n)$: **induced local field**, ossia il campo locale indotto nel neurone j.
- $y_j(n)$: output del neurone, ottenuto applicando la funzione di attivazione φ.
- $e_j(n)$: scarto/errore, differenza tra output vero e output predetto.

La funzione di costo istantanea (MSE) è:

$$E_j(n) = \frac{1}{2} e_j^2(n) \tag{26}$$

Per semplificazione, in queste slide viene considerato **bias nullo**.

43.2 Aggiornamento dei pesi

Applichiamo la chain rule per calcolare la derivata:

$$\frac{\partial E_j(n)}{\partial w_{ji}(n)} = \frac{\partial E_j(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \xi_j(n)} \frac{\partial \xi_j(n)}{\partial w_{ji}(n)} \tag{27}$$

$$= -e_j(n)\, \varphi_j'(\xi_j(n))\, y_i(n) \tag{28}$$

Da cui segue l'aggiornamento del peso:

$$\Delta w_{ji}(n) = \eta\, \delta_j(n)\, y_i(n) \tag{29}$$

dove abbiamo definito il **gradiente locale**:

$$\delta_j(n) = e_j(n)\, \varphi_j'(\xi_j(n)) \tag{30}$$

44 Back-propagation: Neuroni Hidden

La situazione si complica nei neuroni hidden, perché non abbiamo accesso diretto a Y_{true}. Quindi non possiamo calcolare direttamente $e_j(n)$.

44.1 Strategia

Il problema si risolve calcolando $e_j(n)$ in modo ricorsivo usando i segnali di errore dei neuroni successivi a cui il neurone hidden è connesso.

44.2 Esempio

Consideriamo un neurone hidden j, connesso a un neurone di output k:

$$e_k(n) = Y_{\text{true}}(n) - y_k(n) \tag{31}$$

44.3 Gradiente locale per neuroni hidden

Ridefiniamo quindi il gradiente locale del neurone hidden j:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial \xi_j(n)} \tag{32}$$

$$= -\frac{\partial E(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial \xi_j(n)} \tag{33}$$

$$= \left(\sum_{k \in C} \delta_k(n)\, w_{kj}(n) \right) \varphi_j'(\xi_j(n)) \tag{34}$$

dove C è l'insieme dei neuroni collegati a j (successivi nel forward pass).

45 Conclusioni

- La back-propagation permette un calcolo efficiente dei gradienti in reti neurali multilayer.
- Il principio fondamentale è l'applicazione della **chain rule**.
- Nei neuroni di output il gradiente locale è calcolabile direttamente tramite Y_{true}.
- Nei neuroni hidden, invece, il gradiente viene calcolato ricorsivamente grazie ai neuroni successivi.

46 Introduzione

Questi appunti sono una trascrizione fedele e commentata delle slide del corso di *Metodi di Intelligenza Artificiale per la Fisica* (Applicazioni fisiche al Machine Learning). Ogni sezione corrisponde a una slide, con spiegazioni dettagliate delle formule e del contesto. Particolare enfasi viene data alle **Physics Informed Neural Networks (PINN)** e, più in generale, al concetto di **Physics Informed Machine Learning (PIML)**.

47 Transfer Learning

Il **transfer learning** consiste nell'utilizzare conoscenze e funzionalità apprese in un problema per affrontarne un altro simile.

47.1 Motivazione

- Riduce la necessità di disporre di enormi quantità di dati supervisionati.
- Sfrutta rappresentazioni di basso livello già apprese da altri modelli.

47.2 Strategie comuni

1. **Apprendimento non supervisionato**: ad esempio autoencoder o self-supervised learning.
2. **Data augmentation**: creazione di dati sintetici etichettati facilmente.
3. **Trasferimento di rappresentazioni**: riuso dei layer inferiori di un modello per un nuovo compito.

47.3 Transfer learning in DL

Tipico workflow:

1. Prendere i layer di un modello pre-addestrato.
2. Congelarne i parametri.
3. Aggiungere nuovi layer addestrabili.
4. Allenare i nuovi layer sul nuovo dataset.

47.4 Varianti

- **Feature extraction**: uso di una CNN pre-addestrata come estrattore di feature + modello shallow ML.
- **Fine-tuning**: ri-addestramento parziale dei layer feature extractor.
- **Domain adaptation**: adattamento a domini differenti (es. riconoscimento facciale da dataset etichettato a non etichettato).

48 Physics Informed Machine Learning (PIML)

48.1 Definizione

Il PIML rappresenta l'integrazione tra conoscenza fisica e modelli ML.

- Combina modelli data-driven con vincoli fisici noti.
- Migliora accuratezza e robustezza dei modelli.

48.2 Esempio intuitivo: il pendolo

- Tradizionale ML: Autoencoder per comprimere e ricostruire il moto.
- PI-ML: apprendimento anche della dinamica, cioè delle equazioni differenziali che descrivono il sistema.

Esempio di equazione del pendolo smorzato:

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta) + \delta\dot{\theta} \tag{35}$$

49 Fasi di costruzione di un modello ML

1. **Decisione del task**: definizione di input/output e obiettivi.
   In PIML possiamo incorporare direttamente concetti fisici (Hamiltoniana, lagrangiana, forze, ecc.).
2. **Dati**: raccolta e preprocessing.
   Possibile data augmentation fisicamente motivata (rotazioni, traslazioni, scelta di coordinate appropriate).
3. **Architettura neurale**: scelta del modello in base alle conoscenze fisiche (CNN, GNN, AE, HNN, ecc.).

4. **Loss function**: definizione di una funzione obiettivo che include vincoli fisici.

5. **Algoritmo di ottimizzazione**: allenamento bilanciando ricostruzione dati e rispetto delle leggi fisiche.

50 Architetture Neurali Informate dalla Fisica

50.1 Principi generali

- Interpretabilità e generalizzabilità.
- Parsimonia e semplicità ("tutto deve essere il più semplice possibile, ma non più semplice di questo" - A. Einstein).
- Simmetrie e leggi di conservazione: vincoli strutturali nei modelli neurali.

50.2 Esempi

- **ResNet**: ogni blocco residuale approssima un integratore di Eulero.
- **UNet**: bias multiscala utile in segmentazione e denoising.
- **Graph Neural Network**: rappresentazione naturale di sistemi molecolari o dinamici.
- **Reti Lagrangiane**: garantiscono conservazione dell'energia.
- **TBNN**: reti che rispettano simmetrie di Galilei e vincoli tensoriali.

51 Physics Informed Neural Networks (PINN)

Le PINN incorporano le equazioni fisiche nella funzione di costo.

51.1 Loss fisica

$$L = L_{\text{dati}} + L_{\text{fisica}} \tag{36}$$

dove L_{fisica} rappresenta l'errore nel soddisfare le equazioni (es. Navier-Stokes).

51.2 Esempio di PINN per fluidodinamica

- Input: coordinate spazio-temporali.
- Output: campi di flusso.
- Loss: penalizza la violazione delle equazioni di Navier-Stokes anche su punti non etichettati.

52 Hamiltonian Neural Networks (HNN)

52.1 Motivazione

Gli HNN sfruttano la struttura simplettica dei sistemi dinamici. Permettono di preservare quantità conservate come l'energia.

52.2 Equazioni di Hamilton

$$\dot{q}_i = \frac{\partial H}{\partial p_i} \tag{37}$$

$$\dot{p}_i = -\frac{\partial H}{\partial q_i} \tag{38}$$

52.3 Loss function

La loss dell'HNN assicura che le equazioni di Hamilton siano soddisfatte:

$$\min_{\theta} \left(\left\| \frac{dq}{dt} - \frac{\partial H_\theta}{\partial p} \right\|^2 + \left\| \frac{dp}{dt} + \frac{\partial H_\theta}{\partial q} \right\|^2 \right) \tag{39}$$

52.4 Vantaggi

- Conservazione accurata dell'energia.
- Migliore generalizzazione rispetto a reti standard.

53 Conclusioni

- PIML e PINN rappresentano un approccio fondamentale per integrare conoscenza fisica nei modelli ML.

- Architetture e loss possono essere progettate per rispettare simmetrie, invarianti e leggi di conservazione.
- Applicazioni in fisica delle particelle, fluidodinamica, chimica computazionale, cosmologia, e altro.

54 Introduzione alle Reti Neurali Ricorrenti (RNN)

Molti problemi di Deep Learning hanno come caratteristica principale l'esistenza di **correlazioni temporali** tra le feature dell'input. Alcuni esempi pratici includono:

- Analisi di testo scritto (language modeling, sentiment analysis).
- Riconoscimento di musica o suoni complessi.
- Dinamica di oggetti soggetti a forze (ad esempio reazioni tra particelle elementari).
- Analisi di serie temporali: dati finanziari, domanda/offerta di merci, segnali medici o di sensori, segnali da interferometri gravitazionali.

Le **Recurrent Neural Networks (RNN)** sono un'architettura di deep neural network (DNN) ottimizzata per processare sequenze: durante l'elaborazione non considerano solo l'input corrente (come accade in una CNN), ma tengono traccia anche di una parte degli input precedenti.

54.1 Esempio intuitivo

Quando analizziamo una sequenza di parole, il significato di una parola dipende spesso dalle precedenti. Un modello efficace deve quindi combinare l'informazione corrente con quella passata.

55 Modelli per Sequenze e Linguaggio

Un tipico task consiste nel predire la parola successiva in una sequenza di parole.

55.1 Distribuzione reale del linguaggio

La probabilità congiunta di una sequenza di parole $(w_0, w_1, \ldots, w_n)$ si scrive come:

$$P(\text{text}) = P(w_0, w_1, w_2, \ldots, w_n) = P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_0, w_1) \cdots P(w_n|\ldots) \tag{40}$$

Il compito di un modello di linguaggio è approssimare questa distribuzione.

55.2 Uso di finestre fisse

CNN e modelli basati su filtri locali possono catturare correlazioni tra elementi vicini. Tuttavia, non riescono a catturare dipendenze a lungo termine: per esempio, per predire correttamente la parola *italiano* in una frase come "Io sono cresciuto in Italia ... Io parlo fluentemente italiano", servono informazioni lontane nella sequenza.

55.3 Bag of Words (BoW)

Il modello più semplice è il **Bag of Words**, in cui ogni parola è trattata come indipendente dalle altre:

$$P(\text{text}) \sim P(w_0) \cdot P(w_1) \cdot \ldots \cdot P(w_n) \tag{41}$$

Tuttavia, questo approccio ignora grammatica e ordine delle parole, portando a risultati insoddisfacenti.

55.4 Markov Models e RNN

Per migliorare, si possono usare **modelli di Markov**, in cui la probabilità di una parola dipende solo da quella precedente. Un'ulteriore generalizzazione è rappresentata dalle RNN, che approssimano meglio la distribuzione del linguaggio tenendo traccia del contesto.

56 Word Embedding

Il testo deve essere trasformato in vettori per poter essere processato da una rete neurale.

56.1 One-hot encoding

Ogni parola è rappresentata da un vettore binario con un solo 1. Questo metodo è semplice ma inefficiente: la dimensione del vettore cresce con la dimensione del vocabolario e non cattura relazioni semantiche.

56.2 Word2Vec

Proposto da Mikolov (Google, 2013), il Word2Vec è una tecnica di embedding che associa a ogni parola un vettore di numeri reali. Le parole con significati simili vengono mappate in vettori vicini nello spazio. Il modello si allena predicendo la probabilità che una parola target compaia in un dato contesto.

57 Architetture RNN

Le RNN sono progettate per:

- Ricevere input di lunghezza variabile.
- Mantenere traccia delle dipendenze tra elementi distanti.
- Conservare informazioni sull'ordine degli elementi.
- Usare parametri condivisi, permettendo di trasferire le correlazioni lungo tutta la sequenza.

57.1 Topologie di RNN

- Input singolo $\rightarrow$ sequenza (es. image captioning).
- Sequenza $\rightarrow$ output singolo (es. sentiment analysis, time series forecasting).
- Sequenza $\rightarrow$ sequenza (es. traduzione automatica, generazione di musica).

58 Funzionamento di una RNN

58.1 La cella ricorrente

Ad ogni passo temporale t, la cella riceve un input x_t e lo stato nascosto precedente h_{t-1}, producendo un nuovo stato h_t:

$$h_t = f_w(h_{t-1}, x_t) \tag{42}$$

$$\hat{y}t = g(Whyh_t) \tag{43}$$

La stessa funzione f_w con gli stessi pesi viene usata in ogni passo.

58.2 RNN come grafo computazionale

Una RNN può essere vista come una serie di copie della stessa rete, ognuna che passa informazioni alla successiva.

58.3 Vanilla RNN

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{44}$$
$$\hat{y}t = g(Whyh_t) \tag{45}$$

dove g è in genere una softmax (per classificazione).

59 Problemi delle RNN e Soluzioni

59.1 BPTT: Backpropagation Through Time

Per allenare le RNN si usa l'unrolling nel tempo e la backpropagation. Questo è computazionalmente costoso e sensibile ai problemi di **vanishing** ed **exploding gradients**.

59.2 Soluzioni parziali

- Gradient clipping per limitare l'esplosione dei gradienti.
- Inizializzazione dei pesi come matrici identità.
- Uso di funzioni di attivazione limitate come sigmoid e tanh.

59.3 Soluzioni efficaci: Gated RNN

Le **LSTM (Long Short-Term Memory)** e le **GRU (Gated Recurrent Unit)** introducono meccanismi di gating per controllare il flusso di informazione.

60 LSTM: Long Short-Term Memory

60.1 Struttura

La LSTM introduce uno **cell state** C_t, che funge da memoria a lungo termine, e un hidden state h_t, memoria a breve termine.

Ogni cella LSTM include tre gate principali:

- Forget gate f_t: decide quali informazioni eliminare dal cell state.
- Input gate i_t: decide quali nuove informazioni aggiungere.
- Output gate o_t: decide cosa emettere come output.

60.2 Aggiornamenti

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{46}$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{47}$$
$$\tilde{C}t = \tanh(W_C[ht - 1, x_t] + b_C) \tag{48}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{49}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{50}$$
$$h_t = o_t \odot \tanh(C_t) \tag{51}$$

60.3 Vantaggi

- Gestione efficace delle dipendenze a lungo termine.
- Miglior conservazione del gradiente durante il backpropagation.

61 GRU: Gated Recurrent Unit

La GRU semplifica la LSTM combinando cell state e hidden state. Usa due gate (reset e update) invece di tre, riducendo la complessità computazionale mantenendo prestazioni simili.

62 Esempio pratico in PyTorch

62.1 Classificatore LSTM

Esempio di implementazione in PyTorch:

```
import torch
import torch.nn as nn

class LSTMClassifier(nn.Module):
    def _init_(self, input_dim, hidden_dim, output_dim, n_layers=1):
        super(LSTMClassifier, self)._init_()
        self.lstm = nn.LSTM(input_dim, hidden_dim, n_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        out, (h, c) = self.lstm(x)
        out = self.fc(out[:, -1, :])  # usiamo l'ultimo hidden state
```

```
        return out

# esempio di uso
model = LSTMClassifier(input_dim=50, hidden_dim=100, output_dim=2)
x = torch.randn(32, 10, 50)  # batch_size=32, seq_len=10, input_dim=50
y = model(x)
print(y.shape)  # -> torch.Size([32, 2])
```

63 Convolutional Neural Networks (CNN)

63.1 Introduzione

Le **Convolutional Neural Networks** (CNN) sono una classe di reti neurali particolarmente adatte per dati strutturati in griglie, come immagini, segnali o video.

Rispetto a un MLP (*Multilayer Perceptron*), una CNN sfrutta due proprietà fondamentali:

- **Local connectivity**: ogni neurone è connesso solo a un sottoinsieme locale dell'input.
- **Weight sharing**: lo stesso filtro (kernel) viene applicato in più posizioni, riducendo drasticamente il numero di parametri.

Reti neurali convoluzionali (CNN). Le CNN sono progettate per il riconoscimento di immagini, operando direttamente sui pixel organizzati in griglie. I loro bias induttivi derivano da proprietà tipiche degli input visivi: (i) *località* delle feature (bastano pochi pixel contigui per descrivere una sub-feature); (ii) *invarianza/equivarianza rispetto a traslazioni*; (iii) *autosimilarità*, cioè lo stesso filtro può riconoscere sub-feature identiche in punti diversi; (iv) *composizionalità*, per cui feature complesse si ottengono combinando sub-feature più semplici.

—

64 Motivazione delle CNN

64.1 Limiti degli MLP

Un MLP con layer fully connected applicato a immagini soffre di:

- Numero di parametri troppo elevato: una immagine 256×256 con 3 canali ha già oltre $200k$ feature in input.
- Mancanza di **invarianza spaziale**: un oggetto spostato nell'immagine produce feature completamente diverse.

64.2 Vantaggi delle CNN

- Catturano **strutture locali** (bordi, texture, pattern).

- Generalizzano meglio sfruttando la **convoluzione** e il riuso dei pesi.
- Introducono naturalmente **invarianza traslazionale**.

Convolutional Feature Extraction Layer. Questi layer identificano feature simili in diverse posizioni dell'immagine, basandosi su tre concetti chiave: (i) *local receptive field* (kernel o filtri convoluzionali), (ii) *shared weights* per i kernel, (iii) *pooling layers*. I neuroni di input (uno per ogni pixel) non sono fully connected al layer nascosto, ma connessi solo a una piccola regione dell'immagine (campo ricettivo locale). Tale campo scorre sull'intera immagine e a ciascuno shift corrisponde un neurone nel layer nascosto.
Nelle CNN tutti i neuroni di un hidden layer condividono gli stessi pesi: ciò consente *invarianza traslazionale* e *auto-similarità*, poiché lo stesso kernel identifica la medesima caratteristica in punti diversi dell'immagine. Poiché la rete deve riconoscere molte feature elementari, vengono usati più kernel, ognuno associato al proprio hidden layer. Rispetto a una DNN densa, questo approccio riduce drasticamente il numero di parametri da apprendere (rappresentazione sparsa). Dopo la convoluzione, una funzione di attivazione non lineare (ad es. ReLU) introduce la necessaria non linearità nel modello.

Sparsità delle interazioni. L'assunzione di località nelle CNN è legata alla nozione di *sparsità*: una funzione dipende solo da un sottoinsieme ristretto degli input. Un esempio è l'algoritmo k-NN, in cui la predizione $g(x)$ si basa solo sui k vicini più prossimi, anziché sull'intero dataset. In modo analogo, nelle CNN il numero di pixel coinvolti in una convoluzione è limitato al campo ricettivo locale. Lo stimatore a kernel di Nadaraya-Watson generalizza il k-NN estendendo la somma a tutti i punti, pesati in base alla distanza. Rendendolo convesso tramite softmax sui pesi, si ottiene il *meccanismo di attenzione*.

65 Operazione di convoluzione

65.1 Definizione matematica

Data un'immagine $I(x, y)$ e un kernel $K(a, b)$, la convoluzione discreta è definita come:

$$S(x, y) = (I * K)(x, y) = \sum_a \sum_b I(x - a, y - b)\, K(a, b).$$

65.2 Intuizione

- Il kernel agisce come un **filtro** che estrae pattern locali.
- Esempi: rilevazione di bordi (Sobel, Prewitt), sfocatura (Gaussian blur), sharpening.

—

66 Esempi di kernel classici

- **Filtro Laplaciano (edge detection)**:

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- **Filtro di smoothing (media)**:

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

—

67 Struttura di una CNN

Una CNN è tipicamente composta da una sequenza di layer:

1. **Convolutional layer**: applica diversi filtri all'input.
2. **Activation function**: ReLU è la più comune.
3. **Pooling layer**: riduce dimensionalità mantenendo informazione essenziale.
4. **Fully connected layer**: parte finale che combina le feature estratte per la classificazione.

—

68 Convolutional Layer

68.1 Operazione

Ogni convoluzione applica k filtri, producendo k *feature map*.

$$h_{ij}^k = a\left(\sum_m \sum_n x_{i+m,j+n} \cdot w_{mn}^k + b^k\right).$$

68.2 Proprietà

- Ogni filtro è appreso durante il training.
- I filtri nei primi layer catturano bordi e pattern semplici, nei layer profondi catturano strutture complesse (occhi, forme, texture).

—

69 Pooling Layer

69.1 Funzione

Il **pooling** riduce la dimensione spaziale delle feature map. Operazioni comuni:

- **Max pooling**: prende il massimo valore in una finestra.
- **Average pooling**: prende la media.

Pooling layers. Oltre ai layer di convoluzione, le CNN includono i *pooling layers*, tipicamente applicati dopo ciascuna convoluzione. Essi eseguono un'operazione di *downsampling*, riducendo la dimensionalità dell'output convoluzionale: ciò semplifica il modello (meno parametri) e lo rende meno sensibile a piccole traslazioni dell'immagine. Per compiti di classificazione, infatti, non è cruciale la posizione assoluta di una sub-feature, ma la sua posizione relativa rispetto alle altre.

69.2 Vantaggi

- Riduce la dimensionalità e i costi computazionali.
- Introduce **invarianza locale** a piccole traslazioni.

—

70 Esempio di architettura CNN

Una tipica CNN per classificazione immagini può essere:

- Input: immagine $32 \times 32 \times 3$.
- Conv layer (32 filtri 3×3) + ReLU.
- Max pooling 2×2.
- Conv layer (64 filtri 3×3) + ReLU.
- Max pooling 2×2.
- Fully connected (128 neuroni).
- Output layer con softmax per classificazione multi-classe.

—

71 Funzioni di attivazione in CNN

- **ReLU**: la più utilizzata per stabilità ed efficienza.
- **Varianti**: LeakyReLU, ELU, GELU, usate per ridurre problemi come "dying ReLU".
- Output layer: softmax per classificazione, sigmoide per classificazione binaria.

—

72 Training delle CNN

72.1 Procedura

1. Inizializzazione dei pesi (Glorot, He).
2. Forward pass (convoluzioni, attivazioni, pooling, FC).
3. Calcolo loss (es. cross-entropy).
4. Backpropagation con **convoluzioni trasposte** per propagare i gradienti.
5. Aggiornamento pesi (SGD, Adam).

72.2 Problemi comuni

- **Overfitting**: risolto con dropout, data augmentation, early stopping.
- **Vanishing gradients**: mitigato con ReLU e normalizzazione.

—

73 Esempio pratico in PyTorch

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.fc1 = nn.Linear(64 * 6 * 6, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 6 * 6)  # flatten
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

model = SimpleCNN()
```

—

74 Applicazioni delle CNN

- Visione artificiale: riconoscimento immagini, detection, segmentation.
- Elaborazione del linguaggio: estrazione di feature da sequenze (CNN 1D).

- Bioinformatica: analisi strutture proteiche, sequenze genomiche.
- Fisica delle alte energie: analisi di dati calorimetrici e immagini da rivelatori.

75 Introduzione agli Auto-Encoders

Gli **Auto-Encoders (AE)** sono reti neurali profonde (DNN) progettate per prendere un input x e predire lo stesso input come output. Questa struttura li rende particolarmente utili per:

- Apprendimento auto-supervisionato di rappresentazioni compatte dei dati.
- Compressione e riduzione dimensionale.
- Denoising e ricostruzione di immagini.
- Anomaly detection.

75.1 Architetture possibili

Gli AE possono essere implementati con vari tipi di layer: fully-connected (Dense), CNN, RNN, GNN, ecc. La flessibilità dell'architettura consente di adattare gli AE a diverse tipologie di dati (vettori di feature, immagini, sequenze temporali).

75.2 Under-complete AE

L'implementazione più semplice prevede un "bottleneck" nello spazio latente, cioè uno spazio z con dimensione $\dim(z) < \dim(x)$. Questo obbliga la rete a comprimere l'informazione eliminando il rumore inessenziale.

76 Definizione formale di un Auto-Encoder

Un AE è composto da due moduli principali:

- **Encoder** $g_\phi : \mathbb{R}^D \to \mathbb{R}^d$ che mappa l'input x nello spazio latente $z = g_\phi(x)$.
- **Decoder** $f_\theta : \mathbb{R}^d \to \mathbb{R}^D$ che ricostruisce l'input a partire dalla rappresentazione latente $\hat{x} = f_\theta(z)$.

La rete viene addestrata minimizzando la differenza tra input e output:

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 \tag{52}$$

oppure tramite una cross-entropy loss se i dati sono limitati in $[0, 1]$.

77 AE e PCA

Se consideriamo un AE lineare con attivazioni lineari, esso si riconduce a una **PCA (Principal Component Analysis)**. Infatti, se $\dim(z) < \dim(x)$, l'AE apprende una proiezione su un sottospazio S di dimensione ridotta che minimizza la distanza quadratica dai dati.

$$\hat{x} = UVx, \quad U = Q, V = Q^T \tag{53}$$

dove Q è una base ortonormale di S.

77.1 Non linearità vs PCA

Gli AE non lineari possono apprendere rappresentazioni latenti più potenti rispetto alla PCA, permettendo di catturare strutture complesse nei dati.

78 Qualità delle Rappresentazioni

78.1 Compressione latente

Lo spazio latente agisce da compressore dell'informazione. Tuttavia, negli AE standard lo spazio latente non è continuo, limitando interpolazioni e strutturazione dello spazio (problema risolto dai VAE).

78.2 AE profondi e manifold

Un AE profondo non proietta su un sottospazio lineare, bensì su un **manifold non-lineare**. Questo consente una rappresentazione gerarchica e non lineare dei dati.

79 Training degli Auto-Encoders

Durante l'addestramento, l'encoder cerca di "srotolare" il manifold dei dati, trovando una rappresentazione latente più semplice. Il decoder invece ricostruisce l'input a partire da z.

79.1 Problemi di stabilità

La distribuzione appresa nello spazio latente può dipendere da fattori casuali (inizializzazione pesi, iperparametri, rumore di training). Questo può portare a incoerenze tra spazi latenti appresi da reti addestrate in modo indipendente.

80 AE e Restricted Boltzmann Machine (RBM)

Le RBM sono modelli stocastici che apprendono distribuzioni di probabilità, mentre gli AE sono deterministici. Entrambi però introducono variabili nascoste. A differenza degli AE, le RBM richiedono procedure di addestramento complesse (MCMC, Gibbs sampling).

Autoencoder (AE) e Restricted Boltzmann Machine (RBM). Le RBM sono una variante delle Boltzmann Machine in grado di apprendere distribuzioni di probabilità su un dataset. Sono reti shallow con un layer visibile e uno nascosto, organizzati in un grafo bipartito (nessuna connessione intra-layer). L'idea è distinguere tra variabili osservabili e variabili nascoste di alto livello, analogo allo spazio latente z di un AE, ma con una struttura probabilistica. A differenza degli AE (deterministici, addestrati con SGD), le RBM usano un approccio stocastico e vengono allenate con procedure MCMC basate su Gibbs sampling, molto più costose. In un AE con unità sigmoidi, la funzione di energia coincide con l'energia libera di una RBM.

81 Implementazioni pratiche

81.1 Dense AE in PyTorch

```
import torch
import torch.nn as nn

class DenseAE(nn.Module):
    def __init__(self, input_dim, hidden_dim, latent_dim):
        super(DenseAE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, latent_dim)
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, input_dim),
            nn.Sigmoid()
        )
```

```
def forward(self, x):
    z = self.encoder(x)
    return self.decoder(z)
```

81.2 Convolutional AE

Gli auto-encoder convoluzionali usano layer di convoluzione e upsampling per ricostruire immagini.

Decoder e upsampling layers. Nelle fasi di decoding (es. in AE, VAE e GAN) è necessario eseguire un'operazione di *upsampling*. Nei layer densi ciò è immediato, mentre nei convoluzionali esistono diverse tecniche: dalle più semplici, prive di parametri da apprendere, fino a trasformazioni di upsampling apprese durante il training. Esempi di metodi basici sono il *nearest neighbor* e il *bed of nails* upsampling.

Transpose Convolutional Layers. La tecnica più usata per l'upsampling è la *fractionally-strided convolution*, anche detta *transpose convolution*. Mentre una convoluzione standard riduce la dimensione spaziale (es. $5 \times 5 \rightarrow 3 \times 3$), una transpose convolution può aumentarla (es. $3 \times 3 \rightarrow 5 \times 5$). L'idea è inserire pixel vuoti tra quelli originali e applicare una convoluzione tradizionale, ottenendo così un tensore più grande.

Dal punto di vista formale, una transpose convolution è l'operatore inverso della convoluzione in termini di propagazione dei gradienti: infatti è implementata come la convoluzione standard del layer precedente con i pesi trasposti. Rispetto a semplici tecniche di upsampling (nearest neighbor, bed of nails), questo approccio introduce parametri addestrabili, consentendo al modello di apprendere come ricostruire strutture e dettagli fini. Tuttavia, può introdurre artefatti tipici, come i *checkerboard patterns*, che vengono spesso mitigati combinando la transpose convolution con upsampling esplicito seguito da una convoluzione standard.

82 Denoising Auto-Encoders (DAE)

Un DAE viene addestrato a ricostruire versioni pulite degli input corrotti da rumore. Questa tecnica evita che la rete apprenda la semplice copia dell'input e aumenta la robustezza. Si aggiunge rumore (tipicamente gaussiano) all'input per creare una versione corrotta dell'input x'. il target non è l'input x' ma l'input originale x.

82.1 Funzionamento intuitivo

Il DAE spinge i punti corrotti fuori dal manifold dei dati, e impara una funzione che li riporta verso il manifold.

83 Over-complete e Regularized AE

Negli AE over-complete ($\dim(z) > \dim(x)$), la capacità del modello viene limitata tramite regolarizzazione piuttosto che tramite dimensione del bottleneck.

83.1 Sparse AE

Si impone un vincolo di sparsità sulle attivazioni medie dei neuroni nascosti, tipicamente usando la divergenza di Kullback-Leibler.

Sparse Autoencoder (Sparse AE). Gli *Sparse AE* impongono un vincolo di sparsità sui neuroni nascosti. Per ogni neurone j del layer nascosto, si calcola l'attivazione media sulla batch:

$$\hat{\rho}_j = \frac{1}{M} \sum_{i=1}^{M} a_j^{(i)},$$

dove M è la dimensione della batch e $a_j^{(i)}$ l'attivazione del neurone j per l'esempio i. Si aggiunge al costo originale una penalità di sparsità:

$$\Omega = \sum_{j=1}^{s} \mathrm{KL}(\rho \,\|\, \hat{\rho}_j) = \sum_{j=1}^{s} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j},$$

dove s è il numero di neuroni nascosti e ρ il parametro target di sparsità (tipicamente piccolo, es. 0.05). La funzione di perdita complessiva diventa:

$$L(x, g(f(x))) + \lambda \Omega,$$

dove λ controlla il peso del vincolo di sparsità. La penalità Ω dipende dai pesi e bias del layer nascosto. L'uso della divergenza di Kullback-Leibler garantisce che le attivazioni medie $\hat{\rho}_j$ rimangano vicine al valore desiderato ρ.

83.2 Contractive AE

Si aggiunge una penalizzazione basata sulla norma del Jacobiano dell'encoder, forzando la rete a imparare rappresentazioni robuste a piccole variazioni degli input.

Contractive Autoencoder (CAE). Un AE standard può ricostruire bene i dati, ma non garantisce che lo spazio latente z sia *robusto*: piccoli cambiamenti nell'input possono causare grandi variazioni in z. Il CAE introduce una penalizzazione che forza l'encoder a essere insensibile a piccole perturbazioni dell'input, aggiungendo al costo originale una regolarizzazione sul jacobiano delle attivazioni dei layer nascosti:

$$L = L(x, g(f(x))) + \lambda\,\Omega, \quad \Omega = \sum_i \|\nabla_x h_i\|_F^2 = \sum_{i,j} \left(\frac{\partial h_i(x)}{\partial x_j}\right)^2,$$

dove h_i è l'attivazione del neurone i del layer nascosto e $\|\cdot\|_F$ indica la norma di Frobenius.

Vantaggi: il CAE produce uno spazio latente più stabile e modella meglio la distribuzione di input rispetto a un DAE.

Svantaggi: implementazione più complessa, richiede algoritmi di ottimizzazione di secondo ordine (es. conjugate gradient).

84 Architetture Varianti

84.1 Stacked What-Where AE (SWWAE)

Introducono variabili "what" (contenuto) e "where" (posizione) per migliorare la ricostruzione e la classificazione.

Stacked What-Where Autoencoder (SWWAE). Proposta da Hinton, la *capsule* è implementata come stack di AE convoluzionali, in cui encoder e decoder combinano convolutional e pooling layer. Ogni pooling layer genera due set di variabili:

- *what*: feature incomplete riguardo alla posizione (es. max del max pooling);
- *where*: informazioni sul posizionamento delle feature più significative (es. argmax del pooling), usate dal decoder.

Le SWWAE possono essere addestrate sia in modalità non supervisionata (ricostruzione come in un AE standard) sia supervisionata (feed-back path riceve la label corretta). L'algoritmo rimane identico, fornendo un approccio unificato. La funzione di perdita complessiva è:

$$L = \lambda_{\text{NLL}} L_{\text{classification}} + \lambda_{\text{rec}} L_2^{\text{rec}} + \lambda_{\text{M}} L_2^{\text{M}},$$

dove si combinano *classification loss*, *reconstruction loss* e *intermediate reconstruction loss*.

84.2 Adversarial AE

Uniscono un AE con un discriminatore avversario per far corrispondere la distribuzione latente a una distribuzione prior desiderata.

Adversarial Autoencoder (AAE). Un AAE combina un autoencoder con una loss avversaria per far sì che la distribuzione dello spazio latente z corrisponda a una distribuzione di probabilità desiderata. Applicazioni comuni includono: classificazione semi-supervisionata, disentangling di stile e contenuto nelle immagini, clustering non supervisionato, riduzione dimensionale e visualizzazione dei dati.

Architettura:

- Un autoencoder convenzionale che ricostruisce x dallo spazio latente z.
- Una seconda rete (*discriminatore*) addestrata a distinguere se z proviene dall'AE o da un campione di una distribuzione prior definita.

L'AAE utilizza la tecnica avversaria per far corrispondere il posterior $q(z)$ ad un prior arbitrario $p(z)$, senza necessità di conoscere la forma analitica di $p(z)$.

85 Auto-Encoders per Anomaly Detection

Gli AE sono molto usati per l'**Anomaly Detection** in scenari non supervisionati. La pipeline tipica è:

1. Addestrare l'AE solo su dati normali.
2. Definire un anomaly score (es. errore di ricostruzione).
3. Confrontare lo score di nuovi dati con la distribuzione attesa per dati normali.

85.1 Esempi pratici

- Dataset MNIST con una classe esclusa dall'addestramento.
- Applicazioni in Fisica delle Alte Energie (es. ricerca di particelle long-lived).

85.2 Altre architetture per AD

- VAE: aggiungono una struttura probabilistica nello spazio latente.
- GAN: stimano la probabilità dei dati tramite un modello generativo.
- Seq-to-Seq e Transformer-based AE per dati sequenziali.
- One-Class Classifier (OCC): addestrati solo su una classe, stimano la coerenza con essa.

86 Apprendimento non supervisionato e Clustering

Nel **learning non supervisionato** l'obiettivo è scoprire strutture latenti o regolarità nei dati senza disporre di etichette o target. Gli approcci principali sono:

- **Riduzione della dimensionalità**: rappresentare i dati in uno spazio ridotto preservando l'informazione (esempi: PCA, t-SNE, UMAP, AE).
- **Apprendimento di rappresentazioni tramite reti neurali**: auto-encoders (AE) e variational auto-encoders (VAE), con applicazioni anche nei modelli di self-supervised learning moderni.
- **Clustering**: raggruppare i dati in cluster sulla base di metriche di distanza o densità.

Apprendimento non supervisionato. Più complesso del supervisionato, richiede valutazione empirica dei risultati. È utile quando il dataset è troppo grande per il labeling o il numero di classi non è noto a priori. Oltre al clustering, può servire a scoprire nuove categorie o come input per classificatori supervisionati, rappresentando una direzione promettente per superare i limiti del deep learning supervisionato.

86.1 Clustering come alternativa pratica

- Tecniche come PCA o t-SNE possono essere usate come strumenti "artigianali" per il clustering, associando manualmente cluster in spazi 2D.
- Tuttavia, oltre le 2 dimensioni, l'identificazione manuale diventa impraticabile.
- Il clustering fornisce quindi un modo più sistematico per scoprire nuove categorie latenti nei dati.

86.2 Sfide del non supervisionato

- Risultati meno immediati da interpretare rispetto al supervisionato.
- I risultati devono sempre essere validati sul campo.
- Fondamentale in scenari con dati non etichettati o con classi non note a priori (es. astronomia, immagini mediche).

87 Definizione di Clustering

Idea: eventi dello stesso cluster sono simili, mentre eventi di cluster diversi sono dissimili.

87.1 Ingredienti di un algoritmo di clustering

1. Una misura di vicinanza (distanza o similarità).
2. Una funzione obiettivo per valutare la bontà del clustering.
3. Un algoritmo per calcolare i cluster (ottimizzazione della funzione obiettivo).
4. Un criterio per determinare il numero di cluster (fisso o scelto empiricamente).

87.2 Metriche di distanza comuni

- Distanza euclidea, Manhattan, Chebyshev.
- Distanza coseno: $d(x_i, x_k) = 1 - \cos(x_i, x_k)$.
- Coefficiente di correlazione di Pearson: $R(x, y) = \frac{C(x,y)}{C(x,x)C(y,y)}$, con

$$C(x, y) = \frac{1}{N-1} \sum (x_i - \bar{x})(y_i - \bar{y}) \tag{54}$$

88 Algoritmi di Clustering

88.1 Clustering elementare

- Si costruiscono cluster unendo punti con distanza inferiore a una soglia d_0.
- Dipendenza critica dalla scelta della soglia.

88.2 Funzione obiettivo SSE (Sum of Square Error)

$$J_{SSE} = \sum_{i=1}^{c} \sum_{x \in D_i} \|x - \mu_i\|^2 \tag{55}$$

dove μ_i è la media del cluster D_i.

Problemi: favorisce cluster compatti e simili in dimensione, può fallire con cluster non compatti o outlier.

88.3 Ottimizzazione iterativa

- Evitare ricerca esaustiva, troppo costosa.
- Algoritmi iterativi (tipo discesa del gradiente): inizializzazione + miglioramento progressivo della funzione obiettivo.
- Possibile convergenza a minimi locali.

Ottimizzazione iterativa nel clustering. Una volta definita la distanza e la funzione obiettivo, il clustering ottimale si ricerca con algoritmi iterativi, poiché una ricerca esaustiva è impraticabile. Procedura tipica:

- scegliere una partizione iniziale ragionevole;
- ripetutamente spostare elementi tra cluster per migliorare la funzione obiettivo, simile a una discesa lungo il gradiente.

Poiché la funzione obiettivo spesso non è differenziabile, la soluzione dipende dalle condizioni iniziali e l'algoritmo può non convergere a un minimo globale, lasciando un grado di arbitrarietà nelle scelte finali.

89 K-Means Clustering

89.1 Procedura base

1. Fissare k, numero di cluster.
2. Scegliere casualmente k centroidi iniziali.
3. Assegnare ogni punto al centroide più vicino.
4. Ricalcolare i centroidi.
5. Ripetere fino a convergenza o step massimo.

89.2 Formulazione formale

Dato un training set $T = \{x_i\}_{i=1}^{N}$ e K cluster:

$$L(x, \mu) = \sum_{k=1}^{K} \sum_{i=1}^{N} r_{ik} \|x_i - \mu_k\|^2 \tag{56}$$

con $r_{ik} = 1$ se x_i appartiene al cluster k, altrimenti 0.

89.3 Algoritmo EM per k-means

- **E-step**: fissati i centroidi $\{\mu_k\}$, assegnare ogni punto al cluster più vicino.
- **M-step**: fissata l'assegnazione $\{r_{ik}\}$, aggiornare $\mu_k = \frac{1}{N_k} \sum r_{ik} x_i$.

K-Means Clustering: EM-like Optimization. L'algoritmo K-Means procede iterativamente alternando due fasi:

Expectation (E-step): dato un set di assignment $\{r_{ik}\}$, si minimizza la funzione obiettivo L rispetto ai centroidi μ_k.

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} r_{ik} x_i, \quad N_k = \sum_{i=1}^{N} r_{ik}$$

Maximization (M-step): dato un set di centroidi $\{\mu_k\}$, si aggiorna l'assignment $\{r_{ik}\}$ associando ogni x_i al centroide più vicino:

$$r_{ik} = \begin{cases} 1 & \text{se } k = \arg\min_{k'} \|x_i - \mu_{k'}\|^2, \\ 0 & \text{altrimenti.} \end{cases}$$

L'algoritmo ripete questi due passi fino al numero massimo di iterazioni o finché L scende sotto una soglia prefissata.

89.4 Scelta di k: Metodo dell'Elbow

- Eseguire k-means per vari valori di k.
- Calcolare la somma delle distanze quadratiche (distortion score).
- Trovare il punto di "gomito" nel grafico: trade-off ottimale tra complessità e accuratezza.

si può usare anche silhouette score per valutare bontà del clustering

89.5 Quando evitare K-Means.

K-Means può non essere adatto se:

- il numero di cluster k è sconosciuto e metodi come elbow o silhouette non danno indicazioni chiare;
- i cluster non sono compatti o i dati contengono molti outlier;

- il dataset è molto grande, poiché il costo computazionale scala come $O(tkn)$ (t = numero iterazioni).

In questi casi è preferibile utilizzare algoritmi di clustering alternativi più flessibili.

90 Algoritmi alternativi

90.1 Clustering gerarchico (Agglomerative)

- Approccio bottom-up: iniziare con ogni punto come cluster separato, unire iterativamente cluster più vicini.
- Criteri di linkage: single, complete, average, centroid, ward.

Clustering gerarchico (metodi agglomerativi). I metodi agglomerativi seguono un approccio *bottom-up*: ogni punto inizia come cluster separato e i cluster più vicini vengono iterativamente uniti secondo un criterio di *linkage*. Algoritmo tipico:

1. inizializza ogni punto come cluster separato;
2. calcola le distanze inter-cluster;
3. unisce i due cluster più simili;
4. ripete fino a ottenere un unico cluster.

Criteri di linkage comuni:

- **single, maximum, average, centroid**: distanza minima, massima, media o tra centroidi;
- **Ward**: minimizza l'aumento della somma delle distanze quadratiche dai centroidi (EES), favorendo cluster compatti e omogenei, il più usato.

Esistono anche approcci *top-down* che partono da un unico cluster e lo suddividono iterativamente.

90.2 DBSCAN (Density Based Spatial Clustering of Applications with Noise)

- Definisce cluster come regioni dense di punti, senza assumere forma convessa.
- Parametri principali: ϵ (raggio), `minPts` (numero minimo di vicini).
- Punti non assegnati a cluster diventano "rumore" (label -1 in scikit-learn).

Metodi density-based (DBSCAN). DBSCAN identifica cluster come regioni ad alta densità separate da zone a bassa densità, senza assumere forme convesse o un numero di cluster noto a priori.

Parametri principali:

- ε (EPS): raggio dell'intorno di un punto;
- minPts: numero minimo di punti (incluso se stesso) per formare un *Core Point*.

Algoritmo: per ogni punto non visitato:

1. costruire l'intorno ε;
2. se è un *core point*, iniziare un nuovo cluster;
3. aggiungere ricorsivamente tutti i punti *density-reachable* da catene di core point.

I punti non assegnati a nessun cluster sono etichettati come *noise*.

90.3 Gaussian Mixture Models (GMM)

- Modelli generativi basati su combinazioni di gaussiane.
- Ogni cluster rappresentato da una gaussiana con parametri μ_k, Σ_k, π_k.
- Training tramite Expectation-Maximization (EM).

Gaussian Mixture Models (GMM) per il clustering. I GMM sono modelli generativi utilizzati per il *soft clustering*, in cui ogni punto ha una probabilità di appartenenza a ciascun cluster.

Definizione: una GMM è una mistura di K gaussiane, ciascuna definita dai parametri μ_k (media), Σ_k (covarianza) e π_k (mixture weight). Ogni gaussiana rappresenta i dati di un cluster, mentre i mixture weight indicano la probabilità che un punto appartenga a quella gaussiana.

Addestramento: i parametri vengono stimati tramite *Expectation-Maximization (EM)*:

- **E-step:** assegnazione dei punti ai cluster in base ai parametri attuali;
- **M-step:** aggiornamento dei parametri delle gaussiane per migliorare la probabilità dei dati.

I GMM permettono di catturare cluster di forma arbitraria e di ottenere informazioni probabilistiche sull'appartenenza dei punti.

91 Implementazioni pratiche con scikit-learn

- **k-means**: `sklearn.cluster.KMeans`
- **Agglomerative clustering**: `sklearn.cluster.AgglomerativeClustering`
- **DBSCAN**: `sklearn.cluster.DBSCAN`
- **GaussianMixture**: `sklearn.mixture.GaussianMixture`

91.1 Esempio k-means

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0)
y_pred = kmeans.fit_predict(X)
```

91.2 Esempio DBSCAN

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
labels = db.labels_
```

91.3 Esempio Gaussian Mixtures

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3).fit(X)
labels = gmm.predict(X)
```

92 Introduzione al Deep Learning Generativo

Uno dei problemi noti del Deep Learning discriminativo è la mancanza di comprensione semantica dei dati di input.
Esempio: una CNN (Convolutional Neural Network) può predire correttamente la probabilità associata ad un'immagine in input, ma può produrre risultati completamente errati se all'immagine viene aggiunto un rumore banale. Questo fenomeno è collegato alla natura discriminativa del modello.

Approccio generativo

L'obiettivo del **Deep Learning generativo** è apprendere la distribuzione congiunta dei dati, cioè:

$$p(x, y) = p(y|x)p(x) \tag{57}$$

Conoscere questa distribuzione permette:

- una migliore comprensione dei dati;
- la quantificazione delle incertezze e delle confidenze nelle predizioni;
- la possibilità di svolgere compiti aggiuntivi: generazione di dati artificiali, rilevamento di outlier (*anomaly detection*), ecc.

93 Approccio Discriminativo vs Generativo

Modelli Discriminativi

I modelli discriminativi apprendono la probabilità condizionata $p(y|x)$. Tuttavia, spesso tendono a essere **over-confidenti**. In altre parole, possono assegnare probabilità molto alte ad una classe anche in condizioni di incertezza.

Modelli Generativi

I modelli generativi apprendono invece la distribuzione congiunta:

$$p(x, y) = p(y|x)p(x) = p(x|y)p(y) \tag{58}$$

Questo consente di rappresentare in modo più realistico l'incertezza, ed è la base per la generazione di nuovi campioni di dati.

94 Modelli Generativi

Un modello generativo viene progettato e addestrato per apprendere la distribuzione di probabilità che descrive il campione di training. In altre parole, si assume che i dati osservati siano un campionamento rappresentativo di una distribuzione ignota.

Una volta appresa tale distribuzione, è possibile:

- campionare nuovi dati artificiali;
- usarla come modello della densità di probabilità.

Tipologie di modelli generativi

Esistono varie architetture:

- **Variational Auto-Encoders (VAE)**;
- **Generative Adversarial Networks (GAN)**;
- Modelli autoregressivi e reti invertibili (normalizing flow);
- Modelli di diffusione e di flow matching;
- Modelli energetici (RBM, Boltzmann Machines, ecc.).

95 Applicazioni del DL Generativo

Il DL generativo ha numerose applicazioni pratiche:

- **Generazione di immagini e video**: (es. DALL-E 3, Stable Diffusion, Midjourney).
- **Emulatori basati su AI** per sostituire simulatori numerici classici molto lenti (es. emulazione fluidodinamica, modelli in biofisica).
- **Previsioni climatiche**: creazione di digital twin atmosferici in grado di emulare la dinamica meteorologica con grande efficienza.
- **Generazione di nuove molecole**: raffinamento iterativo che rispetta le simmetrie fisiche (roto-traslazioni, riflessioni) e sfrutta architetture di tipo graph neural network.

Esempio intuitivo

Un modello di diffusione per molecole può aggiungere progressivamente rumore ad una struttura fino a renderla un ammasso casuale. Poi, tramite il processo inverso $p(z_{t-1}|z_t)$, cerca di rimuovere il rumore ricostruendo la molecola.

96 Generatori latenti basati su reti neurali

Tecniche principali: GMM, VAE, GAN, Normalizing Flow, Diffusion Models. Tutte queste condividono la stessa idea:

- costruire uno spazio latente compatto sulla variabile nascosta z;
- comprimere dati complessi nello spazio latente;
- ricostruire o generare nuovi campioni a partire da z.

Strategie

1. Trasformare campioni da z in distribuzioni su x, cioè approssimare $p(x)$ (es. VAE, GMM).
2. Trasformare campioni da z direttamente in campioni x (es. GAN).

97 Modelli a Variabili Latenti

Molti fattori nei dati non sono espliciti (es. colore capelli, posa di un volto). Introduciamo quindi variabili latenti z che:

- forniscono una rappresentazione compatta ed astratta dei dati;
- catturano feature di alto livello (età, espressione, illuminazione).

$$p(x) = \int p(x|z)p(z)dz \tag{59}$$

Se z è scelto opportunamente, diventa più facile apprendere $p(x|z)$ rispetto ad apprendere direttamente $p(x)$.

98 Modelli Deep a Variabili Latenti

L'idea è assumere uno spazio latente $p(z)$ (tipicamente gaussiano) e apprendere il processo:

$$z \sim p(z) \tag{60}$$
$$x \sim p(x|z) = N(\mu_\theta(z), \Sigma_\phi(z)) \tag{61}$$

Il modello generativo può quindi campionare nuovi dati da z. L'ipotesi è che dopo l'addestramento, le variabili latenti descrivano effettivamente la variabilità del campione reale.

Esempio

Se definiamo:

$$p(x|z) = N([\sin(z), \cos(z)], \sigma^2 I) \tag{62}$$

Il risultato è una distribuzione che disegna una circonferenza nello spazio dei dati.

99 Gaussian Mixture Models (GMM)

Processo generativo

1. Si sceglie una componente k campionando z.
2. Si genera un nuovo dato campionando dalla gaussiana corrispondente.

100 VAE e GAN

Variational Auto-Encoder

- L'encoder mappa i dati nello spazio latente producendo μ e σ.
- Il decoder genera i dati ricostruiti.
- Si massimizza l'ELBO (Evidence Lower Bound), che bilancia:
  - la fedeltà della ricostruzione;
  - la regolarizzazione della distribuzione latente verso una gaussiana standard.

Generative Adversarial Network

- Non modella esplicitamente la probabilità.
- Composta da due attori:
  - Generatore: trasforma z in x_{gen}.
  - Discriminatore: distingue tra x_{vero} e x_{gen}.
- L'allenamento è un gioco a somma zero.

101 Auto-Encoder (AE)

Struttura

Un AE è composto da:

$$x \to g_\phi(x) = z \tag{63}$$
$$z \to f_\theta(z) = \hat{x} \tag{64}$$

Funzione di Loss

$$L(x, \hat{x}) = ||x - \hat{x}||^2 \tag{65}$$

Oppure, per dati binari:

$$L(x, \hat{x}) = -\sum_k \Big[x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k) \Big] \tag{66}$$

102 AE: Qualità della Ricostruzione

Lo spazio latente agisce da compressore di informazione. Tuttavia:

- la rappresentazione non è continua;
- non permette interpolazioni.

Soluzione: passare alle VAE.

103 Variational Auto-Encoder (VAE)

Struttura

- L'encoder produce μ e σ.
- Si campiona $z \sim N(\mu, \sigma)$.
- Il decoder genera $\hat{x}$.

Definizione Generale

Una VAE apprende la distribuzione congiunta $p_\theta(x, z)$. Le distribuzioni condizionali sono:

$$p_\theta(x|z) \quad \text{(decoder)} \tag{67}$$
$$q_\phi(z|x) \quad \text{(encoder approssimato)} \tag{68}$$

Problema: $p_\theta(z|x)$ è intrattabile. Soluzione: approssimare con $q_\phi(z|x)$.

104 Evidence Lower Bound (ELBO)

Variational Autoencoder (VAE). Una VAE apprende la distribuzione congiunta $p_\theta(x, z)$ tra dati x e variabili latenti z con prior $p_\theta(z)$. Le distribuzioni condizionali $p_\theta(x|z)$ e $p_\theta(z|x)$ possono essere viste come decoder e encoder stocastici, legati dal teorema di Bayes.

Calcolare la massima verosimiglianza diretta è intrattabile, perché richiederebbe l'integrazione su tutto lo spazio latente. La soluzione VAE è introdurre un'approssimazione trattabile $q_\phi(z|x)$ della vera distribuzione posteriore $p_\theta(z|x)$, rappresentata tramite una rete neurale.

Si ottimizza $q_\phi(z|x)$ minimizzando la divergenza di Kullback-Leibler $\mathrm{KL}(q_\phi(z|x)\|p_\theta(z|x))$, che corrisponde a massimizzare l'*Evidence Lower Bound (ELBO)*:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)\|p_\theta(z)).$$

Il primo termine è la *loss di ricostruzione* e misura quanto z permette di ricostruire x, il secondo è la KL divergence e regolarizza $q_\phi(z|x)$ verso il prior di z.

Si dimostra che:

$$\log p(x) \geq \mathcal{L}_{ELBO} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\|p(z)) \tag{69}$$

Interpretazione

- **Termine di ricostruzione**: incoraggia il decoder a ricostruire fedelmente x.
- **Termine di regolarizzazione**: spinge $q_\phi(z|x)$ ad avvicinarsi al prior $p(z)$.

105 Ottimizzazione della VAE

La loss della VAE è:

$$\mathcal{L}(\phi, \theta, x) = \text{Loss}_{\text{ricostruzione}} + D_{KL}(q_\phi(z|x) \| p(z)) \tag{70}$$

106 Reparameterization Trick

Problema: come propagare il gradiente attraverso il sampling?
Soluzione: il reparameterization trick.

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim N(0, 1) \tag{71}$$

107 Organizzazione dello Spazio Latente

Nelle VAE, differenti direzioni nello spazio latente corrispondono a feature differenti. È possibile:

- interpolare tra campioni;
- mantenere le variabili scorrelate;
- generare varianti realistiche.

108 Conditional VAE (CVAE)

Una CVAE condiziona il modello su ulteriori variabili (etichette, attributi). La struttura rimane analoga, ma:

- sia encoder che decoder ricevono in input l'informazione condizionale.
- utile per generare campioni controllati (es. cifre specifiche da MNIST).

Esempio in PyTorch

```
class CVAE(nn.Module):
    def __init__(self):
        super(CVAE, self).__init__()
        # Definizione encoder e decoder

    def encode(self, x, y):
        # produce mu e logvar

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def decode(self, z, y):
        # genera x_hat condizionato

    def forward(self, x, y):
        mu, logvar = self.encode(x, y)
        z = self.reparameterize(mu, logvar)
        return self.decode(z, y), mu, logvar
```

109 Introduzione alle GAN

Le **Generative Adversarial Networks** (GAN), introdotte da **Goodfellow et al. (2014)**, rappresentano un approccio alternativo alle **Variational Autoencoders (VAE)** per il *learning generativo*. A differenza delle VAE, le GAN non si basano sulla massimizzazione della verosimiglianza: parliamo infatti di **likelihood-free learning**.

109.1 Idea di base

L'idea principale è quella di addestrare **due modelli in competizione**:

- **Generatore** G: prende in input un vettore di rumore casuale $z \sim p(z)$ e produce un campione sintetico $x = G_\theta(z)$.
- **Discriminatore** D: riceve in input un campione x e stima la probabilità che sia reale piuttosto che generato.

Il generatore cerca di ingannare il discriminatore creando campioni **indistinguibili** dai dati reali, mentre il discriminatore cerca di riconoscere correttamente i campioni veri dai falsi. Questa competizione è formulata come un **gioco a somma zero** (**minimax**).

110 Funzione obiettivo

110.1 Definizione della loss

Abbiamo due sorgenti di dati:

- Dati reali: $x \sim p_{\text{data}}(x)$
- Dati generati: $x \sim p_G(x)$ con $p_G(x) = \int G_\theta(z)p(z)\,dz$

Il discriminatore D_ϕ assegna probabilità alta ai dati reali e bassa a quelli fake. La funzione obiettivo è:

$$\max_\phi V(G_\theta, D_\phi) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D_\phi(G_\theta(z)))] \quad (72)$$

110.2 Obiettivo del generatore

Il generatore minimizza la stessa loss rispetto ai propri parametri θ:

$$\min_{\theta} V(G_\theta, D_\phi) = \mathbb{E}_{z\sim p(z)}[\log(1 - D_\phi(G_\theta(z)))] \tag{73}$$

Quando G è ottimale, vale $p_G = p_{\text{data}}$.

111 Equilibrio ottimale

111.1 Discriminatore ottimale

In un contesto di classificazione binaria ideale, il discriminatore ottimale è dato da:

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)} \tag{74}$$

111.2 Connessione con la Divergenza di Jensen-Shannon

Sostituendo D^* nell'equazione della loss, otteniamo:

$$V(G_\theta, D^*) = 2D_{\text{JSD}}\big(p_{\text{data}} \,\|\, p_G\big) - \log 4 \tag{75}$$

Dove D_{JSD} è la **Jensen-Shannon Divergence**, che è nulla se e solo se $p_G = p_{\text{data}}$.

112 Esempio pratico: dataset MNIST

112.1 Setup

L'obiettivo è generare immagini di cifre scritte a mano (MNIST).

- **Task di** D: classificare correttamente immagini reali e fake.
- **Task di** G: generare immagini realistiche che ingannino D.

112.2 Implementazione

1. G prende in input un vettore di rumore z.
2. Produce un'immagine sintetica $G(z)$.

3. D riceve sia immagini reali che sintetiche e restituisce una probabilità di autenticità.

113 Training alternato

L'addestramento alterna due fasi:

1. Aggiornamento di D: fissiamo G e massimizziamo

$$\log D(x) + \log(1 - D(G(z)))$$

2. Aggiornamento di G: fissiamo D e minimizziamo

$$\log(1 - D(G(z)))$$

Un trucco comune è scambiare le label (label flipping) per stabilizzare il training.

114 Evoluzioni moderne

114.1 DCGAN

Le Deep Convolutional GAN (DCGAN) usano CNN invertite nel generatore per produrre immagini ad alta risoluzione.

114.2 StyleGAN

Introduce un controllo fine sullo spazio latente per generare immagini fotorealistiche.

114.3 CycleGAN

Permette trasformazioni di dominio senza dati appaiati, imponendo consistenza ciclica.

115 Problemi comuni e soluzioni

115.1 Model collapse

Il generatore produce campioni troppo simili. Soluzioni:

- Loss di Wasserstein
- Aggiunta di rumore ai dati
- Regularizzazione e minibatch discrimination

115.2 Ottimizzazione instabile

La loss oscilla senza convergere. Si usano scheduler, penalità sui gradienti e alternative come le WGAN.

Difficoltà nelle GAN. L'addestramento delle GAN è complesso per diversi motivi:

- **Ottimizzazione instabile:** le loss di generatore e discriminatore oscillano spesso senza convergere, rendendo difficile stabilire un criterio di stop robusto.
- **Collasso del modello (mode collapse):** il generatore può produrre sempre lo stesso tipo o pochi tipi di esempi, perdendo variabilità.
- **Valutazione difficile:** misurare oggettivamente qualità e varietà dei campioni generati è complicato.

Molti sforzi empirici sono stati dedicati allo sviluppo di nuove architetture e schemi di regolarizzazione per mitigare questi problemi.

GAN: Mode Collapse. In un esempio toy 1D, la distribuzione dei dati reali è fissata ma sconosciuta, mentre quella generativa iniziale è molto diversa. All'inizio dell'addestramento, il discriminatore può apprendere facilmente la separazione, producendo gradienti quasi nulli vicino ai campioni reali e sintetici, bloccando l'aggiornamento del generatore.

Strategie per mitigare il problema:

- utilizzare un discriminatore smoothed (MSE loss + logits) o aggiungere rumore a dati reali e fake;
- usare una loss basata sulla *Wasserstein distance* per misurare la distanza tra distribuzioni e penalizzare l'aumento di distanza durante l'addestramento.

GAN Training Tricks. Le GAN sono modelli complessi e instabili: la doppia loss può creare vaste zone piatte nel *loss landscape*, causando collasso se l'addestramento parte da tali zone.

Per mitigare questi problemi sono necessari vari trucchi empirici, oltre a pazienza; non esiste garanzia di successo in tutti i contesti. Alcuni esempi classici di *GAN training tricks* sono riportati in letteratura, ad esempio nel libro di F. Chollet.

116 GAN per la fisica delle alte energie

116.1 Simulazioni LHC

Esperimenti come ATLAS e CMS producono miliardi di eventi. Le GAN offrono simulazioni più rapide rispetto a GEANT4.

116.2 CaloGAN

Esempio di GAN condizionate per simulare sciami elettromagnetici nei calorimetri. Velocità fino a 1000× rispetto a GEANT4.

116.3 Metriche fisiche

CaloGAN viene validata confrontando:

- Distribuzione dell'energia depositata
- Centroide spaziale degli sciami
- Confronto diretto con GEANT4