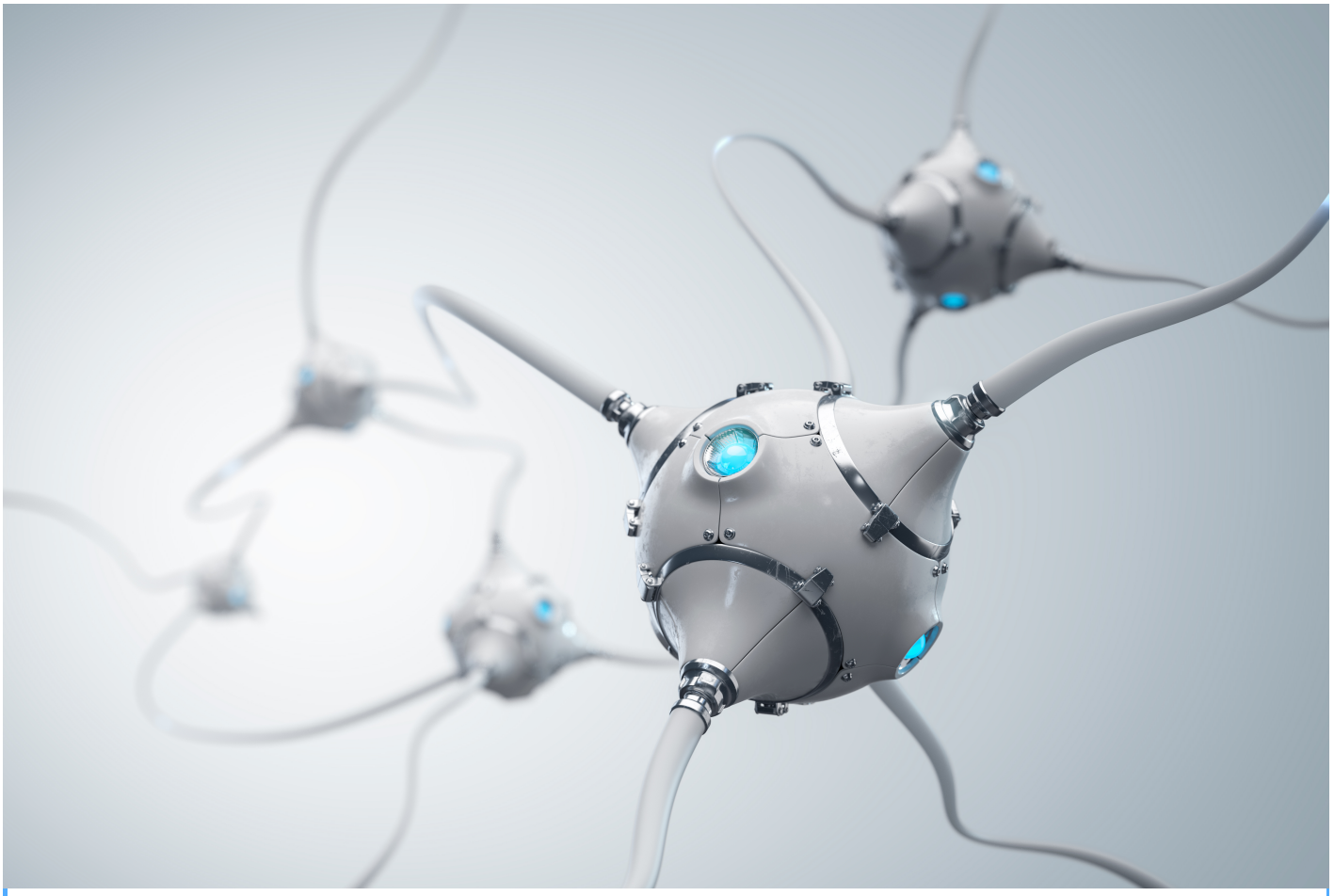
Introduzione al Machine Learning: il percettrone come primo passo verso le reti neurali

DI MASSIMO VAILATI
02/10/2025

MACHINE LEARNING PERCETTRONE



Il **percettrone** è il punto di partenza delle reti neurali: un singolo neurone artificiale che combina ingressi pesati e bias, applica una funzione soglia e decide tra 0 e 1. Utile per problemi **linearmente separabili**, ha ispirato architetture più profonde per superarne i limiti. L'addestramento è supervisionato: a ogni errore si aggiornano pesi e bias con semplici regole di apprendimento, migliorando progressivamente le previsioni.

Il testo propone esempi pratici e codice in **C** per costruire un percettrone, leggere pesi da file, generare decisioni e allenarlo su un dataset logico, trasformando concetti teorici in attività operative.

Il **Machine Learning** (apprendimento automatico) è un settore dell'informatica che si occupa di sviluppare algoritmi e modelli statistici per consentire ai computer di apprendere dai dati, senza la necessità di essere esplicitamente programmati per ogni compito. In altre parole, consente ai sistemi informatici di migliorare le proprie prestazioni nel tempo, elaborando e analizzando le informazioni disponibili.

Si tratta di un importante passo verso la **Intelligenza Artificiale (AI)**, in quanto consente alle macchine di imitare, in parte, alcuni comportamenti tipici dell'intelligenza umana, come il riconoscimento di schemi, la previsione di eventi futuri o il processo decisionale.

Il *Machine Learning* è dunque un sottoinsieme dell'**Intelligenza Artificiale**, ed è utilizzato per sviluppare programmi in grado di **analizzare dati** e **imparare a prevedere risultati**, migliorando le proprie capacità man mano che vengono forniti nuovi dati.

Le Reti Neurali (Neural Networks)

Le Reti Neurali Artificiali sono una tecnica di programmazione ispirata al funzionamento del cervello umano e rappresentano uno degli strumenti fondamentali all'interno del Machine Learning.

Il concetto alla base si ispira al funzionamento del cervello umano: i neuroni artificiali (detti nodi) sono organizzati in strati e connessi tra loro. Ogni nodo riceve segnali da altri nodi, li elabora, e trasmette un risultato.

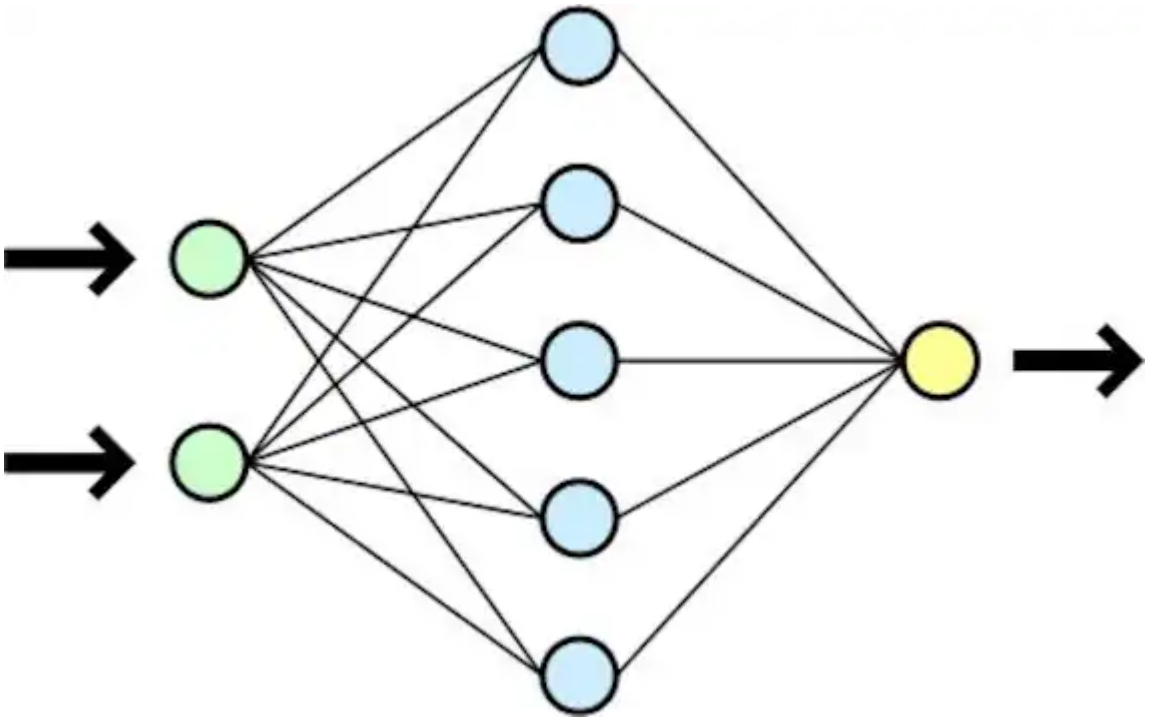


Fig.1

Durante l'addestramento, la rete neurale cerca ripetutamente di risolvere un problema. Quando produce una risposta corretta, rafforza le connessioni (pesi) tra i nodi coinvolti; quando sbaglia, indebolisce quelle connessioni. Questo processo di aggiustamento continuo è chiamato apprendimento.

Proprio come nel cervello umano, le reti neurali migliorano attraverso l'esperienza: imparano sperimentando, commettendo errori, e correggendosi nel tempo.

Grazie a questa capacità adattiva, le reti neurali sono usate in moltissime applicazioni pratiche, tra cui: riconoscimento vocale, visione artificiale, traduzione automatica e guida autonoma.

Il Percettrone: il primo passo verso le Reti Neurali

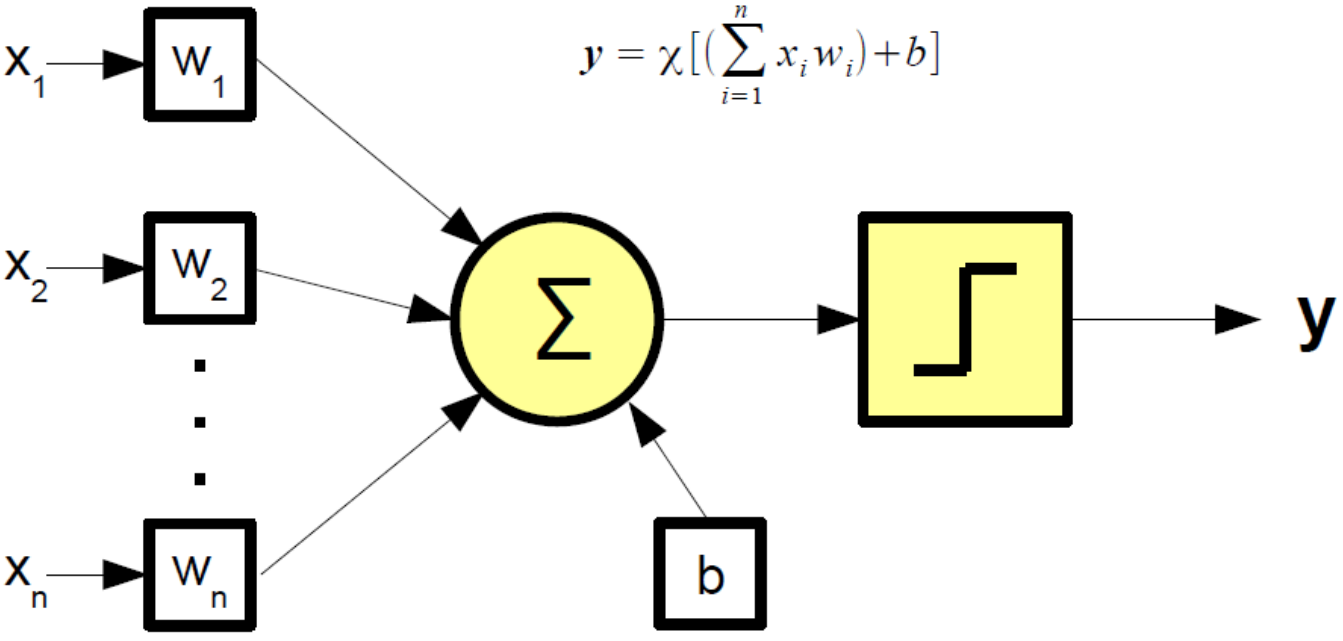
Il percettrone costituisce l'elemento fondamentale e originario delle reti neurali artificiali. Introdotto alla fine degli anni '50 da Frank Rosenblatt, il modello è stato inizialmente accolto con grande entusiasmo, poiché prometteva di emulare alcuni aspetti dell'apprendimento umano. Tuttavia, si è presto evidenziato un limite strutturale significativo: il percettrone semplice è in grado di classificare correttamente solo insiemi di dati linearmente separabili. Questa limitazione ne ha fortemente ridotto l'applicabilità a problemi più complessi, rendendolo inadatto per numerosi contesti reali.

Nonostante ciò, il concetto di percettrone ha rappresentato un'importante pietra miliare nello sviluppo delle reti neurali moderne, ispirando architetture più sofisticate e stratificate, capaci di superare i limiti del modello originale.

Un percettrone è costituito da un singolo nodo (o neurone artificiale) che riceve in ingresso un insieme di segnali $x1,x2,\ldots,x_n$, ciascuno dei quali è associato a un peso $w_1,w_2,\ldots,w_n$. L'unità computa una combinazione lineare di questi ingressi pesati, calcolando una somma ponderata:

$$z = \sum_{i=1}^{n} x_i \cdot w_i + b$$

dove *b* è un termine di bias. L'output del percettrone è determinato dall'applicazione di una funzione di attivazione, tipicamente una funzione soglia (step function). Se il valore della somma *z* supera una soglia prestabilita, il neurone si attiva (output = 1); altrimenti resta inattivo (output = 0). Questo comportamento binario lo rende un semplice classificatore.



Il percettrone può essere addestrato tramite un algoritmo supervisionato, utilizzando esempi noti costituiti da coppie di input e output desiderati. Durante l'addestramento, i pesi vengono aggiornati iterativamente con l'obiettivo di minimizzare l'errore tra l'output

prodotto e quello atteso. In questo modo, il modello apprende una funzione di classificazione che può essere generalizzata per prevedere l'etichetta di nuovi dati.

Il percettrone è in grado di risolvere **problemi linearmente separabili**, determinando un iperpiano di separazione dei dati. Tuttavia, non può risolvere problemi più complessi, per i quali sono necessarie architetture più sofisticate con strati nascosti - da qui lo sviluppo delle reti neurali multilivello.

Esempio: decidere se andare a un concerto

In questo esempio realizziamo e proviamo un percettrone per decidere se andare o meno a un concerto. Il percettrone valuta diversi fattori tra loro indipendenti (come meteo, prezzo, compagnia, ecc.), ognuno con un peso che rappresenta quanto è rilevante nella decisione finale.

Criteri di decisione e pesi:

CRITERIO	Ingresso (x_i)	Peso (w_i)
L'ARTISTA È BRAVO	x_1 = 0 o 1	w_1 = 0,7
IL METEO È FAVOREVOLE	x_2 = 0 o 1	w_2= 0,6
UN AMICO VIENE	x_3 = 0 o 1	w_3 = 0,5
VIENE SERVITO CIBO	x_4 = 0 o 1	w_4 = 0,3
VIENE SERVITO ALCOL	x_5 = 0 o 1	w_5 = 0,4

Bias b = **0,0**

Supponiamo il seguente caso:

- L'artista è bravo → x_1 = 1
- Il meteo è brutto → x_2 = 0
- Un amico viene → x_3 = 1
- Niente cibo → x_4 = 0
- C'è alcol → x_5 = 1

Calcolo:

$$z = 1 \bullet 0{,}7 + 0 \bullet 0{,}6 + 1 \bullet 0{,}5 + 0 \bullet 0{,}3 + 1 \bullet 0{,}4 + 0 = 1{,}6$$

Se **θ = 1.5 → Vai al concerto**
Se **θ =2.0 →Non andare**

Considerazioni:

- I **pesi** possono variare da persona a persona.
  Ad esempio, se per te il **meteo** ha un peso di 0.6, per qualcun altro potrebbe valere 0.2 (cioè è meno importante) oppure 0.9 (molto importante). Un peso più alto indica maggiore rilevanza del criterio nella decisione.
- Anche il **valore di soglia** (**θ**) può cambiare.
  Se per te la soglia è 1.5, un'altra persona potrebbe avere una soglia più bassa (es. 1.0), il che significa che è più **incline ad andare a qualsiasi concerto**, anche se ci sono pochi motivi validi.

Attività pratica n.1

Realizzazione ed uso di un percettrone

In questa attività, **scriviamo un programma in linguaggio C** per implementare il **funzionamento di un percettrone**: l'obiettivo è tradurre in codice l'esempio esposto precedentemente.
Iniziamo a scrivere alcune funzioni di utilità (file *percettrone_utils.c*). Il codice comprende:
1. una **funzione di attivazione step** (che decide l'output del neurone);
2. una funzione per **caricare i pesi e il bias da un file**;
3. una funzione per **calcolare la previsione** del percettrone dato un input.

Il numero di caratteristiche (**FEATURES**) è definito come 5, quindi ogni input al percettrone avrà 5 valori. La soglia scelta (**THRESHOLD**) ha valore 0.5.

```
#define FEATURES 5
#define THRESHOLD 0.5
```

```
// Funzione di attivazione (step)
int activation(float x) {
    if (x > THRESHOLD) return 1;
    else return 0;
}
```

- Questa è una **funzione di attivazione** chiamata *step function* (a gradino).
- Prende un valore x (la somma pesata dell'input più il bias) e restituisce:
  - se x > THRESHOLD
  - 0 altrimenti
- Serve per **decidere l'output** finale del percettrone, in base alla una soglia.

```
// Carica pesi e bias da file
int carica_pesi(const char *filename, float weights[], float *bias) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Errore: file %s non trovato!\n", filename);
        return 0;
    }

    for (int i = 0; i < FEATURES; i++) {
        if (fscanf(file, "Peso %*d: %f\n", &weights[i]) != 1) {
            printf("Errore nella lettura del peso %d\n", i);
            fclose(file);
            return 0;
        }
    }

    if (fscanf(file, "Bias: %f", bias) != 1) {
        printf("Errore nella lettura del bias\n");
        fclose(file);
        return 0;
    }
```

```
    fclose(file);
    return 1;
}
```

- **Apre un file di testo** (nome specificato da `filename`) in lettura,
- **Legge i pesi** e li salva nell'array `weights[]`,
- **Legge il bias** e lo memorizza nella variabile `bias`.

Nel nostro esempio il file *pesi_concerto.txt* con i parametri avrà il seguente contenuto:

```
Peso 1: 0.7
Peso 2: 0.6
Peso 3: 0.5
Peso 4: 0.3
Peso 5: 0.4
Bias: 0.0

// Funzione di previsione
int prevedi(float weights[], float bias, int input[]) {
    float somma = bias;
    for (int i = 0; i < FEATURES; i++) {
        somma += input[i] * weights[i];
    }
    return activation(somma);
}
```

- La funzione riceve l'array dei **pesi**, il **bias**, e l'array di **input** (valori 0 o 1).
- Calcola la **somma pesata**: moltiplica ogni input per il suo peso e li somma tutti insieme al bias.
- Passa il risultato alla funzione di attivazione per ottenere l'output binario del percettrone (0 o 1).

Scriviamo ora il programma principale (file *usa_percettrone.c*) che con il quale potremo provare il percettrone configurato.

```
#include <stdio.h>
#include "percettrone_utils.c"

#define FEATURES 5
#define THRESHOLD 5

int main() {
    float weights[FEATURES];
    float bias;

    if (!carica_pesi("pesi_concerto.txt", weights, &bias)) {
        return 1;
    }

    printf("Inserisci i dati:\n");
    int input[FEATURES];
    printf("Artista famoso? (1=Si, 0=No): ");
    scanf("%d", &input[0]);
    printf("Bel meteo? (1=Si, 0=No): ");
    scanf("%d", &input[1]);
    printf("Amici presenti? (1=Si, 0=No): ");
    scanf("%d", &input[2]);
    printf("Cibo buono? (1=Si, 0=No): ");
    scanf("%d", &input[3]);
    printf("Alcool disponibile? (1=Si, 0=No): ");
    scanf("%d", &input[4]);

    int decisione = prevedi(weights, bias, input);

    if (decisione) {
        printf("\n?? Vai al concerto!\n");
    } else {
        printf("\n?? Resta a casa!\n");
    }

    return 0;
}
```

Eseguiamo il programma e verifichiamo il funzionamento del percettrone in alcuni casi:

```
Inserisci i dati:
Artista famoso? (1=Si, 0=No): 1
Bel meteo? (1=Si, 0=No): 1
Amici presenti? (1=Si, 0=No): 1
Cibo buono? (1=Si, 0=No): 0
Alcool disponibile? (1=Si, 0=No): 0

?? Vai al concerto!

Inserisci i dati:
Artista famoso? (1=Si, 0=No): 1
Bel meteo? (1=Si, 0=No): 0
Amici presenti? (1=Si, 0=No): 1
Cibo buono? (1=Si, 0=No): 1
Alcool disponibile? (1=Si, 0=No): 0

?? Vai al concerto!

Inserisci i dati:
Artista famoso? (1=Si, 0=No): 0
Bel meteo? (1=Si, 0=No): 0
Amici presenti? (1=Si, 0=No): 1
Cibo buono? (1=Si, 0=No): 0
Alcool disponibile? (1=Si, 0=No): 0

?? Resta a casa!
```

Allenamento di un percettrone (training)

Allenare un **percettrone** significa individuare i corretti **pesi sinaptici** (e il bias) per permettere al modello di classificare correttamente gli input. L'algoritmo di addestramento del percettrone è piuttosto semplice e si basa su un metodo di **apprendimento supervisionato**.

La funzione di addestramento (training)

La funzione di addestramento ha il compito di far "imparare" il percettrone. In pratica, il percettrone fa una previsione (cioè una "ipotesi") del risultato utilizzando la sua funzione di attivazione.

Ogni volta che la previsione è sbagliata, il percettrone deve correggere i **pesi** associati agli input. Questi pesi sono ciò che permette al modello di capire quali input sono più importanti per ottenere il risultato corretto.

Con ogni previsione e correzione successiva, il percettrone impara gradualmente. Dopo un gran numero di tentativi (anche migliaia), i pesi saranno sempre più vicini ai valori corretti.

Backpropagation (retropropagazione dell'errore)

Durante l'allenamento dopo ogni previsione, il percettrone misura **quanto è sbagliata** rispetto al risultato corretto. Questo "errore" viene poi utilizzato per aggiornare sia i **pesi** che il **bias** (un valore aggiuntivo che aiuta a regolare la previsione).

Se l'errore è grande, i cambiamenti nei pesi saranno più significativi; se l'errore è piccolo, le modifiche saranno più leggere. Questo processo di aggiustamento continuo, che parte dall'errore finale e "torna indietro" verso i pesi, si chiama **backpropagation**, ovvero **retropropagazione** dell'errore.

Dopo aver ripetuto questo ciclo migliaia di volte, il percettrone migliorerà notevolmente la sua capacità di fare previsioni corrette.

Dati richiesti

- Un insieme di dati di addestramento: coppie (x,y) dove:
  - $x \in R^n$: vettore di input con n caratteristiche
  - $y \in \{0,1\}$: etichetta target
- Tasso di apprendimento $\eta > 0$ (tipicamente piccolo, es. 0.1)
- Inizializzazione casuale dei pesi $w=(w_1,w_2,\ldots,w_n)$ e del bias b

Passaggi dell'algoritmo

- **Per ogni epoca** (numero di passaggi di allenamento):
- Per ogni esempio (x,y) nel dataset:
  a. Calcola l'output del percettrone: $o(x)$

b. Aggiorna i pesi se c'è un errore di classificazione:
$errore = y - o(x)$
$w_i \leftarrow w_i + \eta \cdot errore \cdot x_i$
c. Aggiorna anche il bias
$b \leftarrow b + \eta \cdot errore$

Attività pratica n.2

Training di un precettrone

In questa attività, **scriviamo un programma in linguaggio C** per **allenare un percettrone** utilizzato per rispondere a questo problema: considera le variabili di input *sole*, *tempo* e *stanco* che indicano la situazione "c'è il sole", "ho tempo libero", "sono stanco" e considera l'output *correre* che indica se "vado a correre". La risposta deve essere positiva se ho tempo libero e non sono stanco, oppure c'è il sole e non sono stanco. In pratica il percettrone dovrà realizzare la funzione logica: (tempo OR sole) AND NOT stanco.

Il dataset è costituito da tutte le combinazioni delle tre variabili di input (in totale 8 esempi) e il relativo output atteso è dato dall'espressione logica suddetta. Scegliamo un tasso di apprendimento di 0.1 e un numero di epoche pari a 100.

```
#define N 8     // numero di esempi
#define EPOCHS 100
#define LEARNING_RATE 0.1
```

Il codice che definisce il dataset è il seguente:

```
// Dati di addestramento
   int input[N][3] = {
      {0, 0, 0}, {0, 0, 1},
      {0, 1, 0}, {0, 1, 1},
      {1, 0, 0}, {1, 0, 1},
      {1, 1, 0}, {1, 1, 1}
   };
   int expected[N];
```

```
// Crea il dataset in base alla logica
for (int i = 0; i < N; i++) {
   int sole = input[i][0];
   int tempo = input[i][1];
   int stanco = input[i][2];
   if ((tempo == 1 && stanco == 0) || (sole == 1 && stanco == 0)) {
    expected[i] = 1;
   } else {
    expected[i] = 0;
   }
}
```

Stabiliamo i valori iniziali casuali dei pesi e del bias.

```
// Pesi casuali iniziali
   float weights[3] = {0.0, 0.0, 0.0};
   float bias = 0.0;
```

L'algoritmo di addestramento è il seguente:

```
// Addestramento
   for (int epoch = 0; epoch < EPOCHS; epoch++) {
      for (int i = 0; i < N; i++) {
         float sum = bias;
         for (int j = 0; j < 3; j++) {
            sum += weights[j] * input[i][j];
         }

   // predizione e calcolo dell'errore
         int output = activation(sum);
         int error = expected[i] - output;

  // Aggiornamento dei pesi
         for (int j = 0; j < 3; j++) {
            weights[j] += LEARNING_RATE * error * input[i][j];
```

```
        }
        bias += LEARNING_RATE * error;
    }
}
```

La funzione di attivazione è la stessa vista nell'attività 1 con soglia pari a 0,5.

Al termine dell'allenamento visualizziamo i pesi finali e facciamo un test di funzionamento.

```
// Output dei pesi finali
printf("Pesi allenati:\n");
for (int i = 0; i < 3; i++) {
    printf("Peso %d: %f\n", i, weights[i]);
}
printf("Bias: %f\n", bias);

// Test finale
printf("\nTest del percettrone:\n");
for (int i = 0; i < N; i++) {
    float sum = bias;
    for (int j = 0; j < 3; j++) {
        sum += weights[j] * input[i][j];
    }
    int output = activation(sum);
    printf("Input [%d, %d, %d] => Correre: %d (Atteso: %d)\n",
           input[i][0], input[i][1], input[i][2], output, expected[i]);
}
```

Eseguiamo il programma e verifichiamo il funzionamento del percettrone:

```
Pesi allenati:
Peso 0: 0.100000
Peso 1: 0.200000
Peso 2: -0.300000
Bias: 0.400000
```

```
Test del percettrone:
Input [0, 0, 0] => Correre: 0 (Atteso: 0)
Input [0, 0, 1] => Correre: 0 (Atteso: 0)
Input [0, 1, 0] => Correre: 1 (Atteso: 1)
Input [0, 1, 1] => Correre: 0 (Atteso: 0)
Input [1, 0, 0] => Correre: 1 (Atteso: 1)
Input [1, 0, 1] => Correre: 0 (Atteso: 0)
Input [1, 1, 0] => Correre: 1 (Atteso: 1)
Input [1, 1, 1] => Correre: 0 (Atteso: 0)
```

Il percettrone si è dimostrato capace di apprendere correttamente la funzione logica desiderata: (*tempo* OR *sole*) AND NOT *stanco*. Utilizzando il dataset il modello ha allenato i pesi fino a convergere a valori che permettono di produrre output corretti per ogni input testato. Come mostrato dai risultati finali, l'output del percettrone coincide perfettamente con quello atteso in tutti gli otto casi, dimostrando così il corretto funzionamento del modello e la sua capacità di apprendere una semplice funzione logica tramite il processo di apprendimento supervisionato.