UNIVERSITÀ
DI PARMA

Dipartimento di Ingegneria e Architettura

Corso di Laurea in Ingegneria Informatica, Elettronica e delle Telecomunicazioni

---

Programmazione Genetica per l'ottimizzazione della Rappresentazione Latente di pattern

GP-based pattern Embedding optimization

Relatore:
Prof. Stefano Cagnoni

Tesi di Laurea di:
Federico Brandini

ANNO ACCADEMICO 2022/2023

La **Programmazione Genetica** (*Genetic Programming, GP*) è una tecnica di risoluzione per problemi di ottimizzazione in spazi di dimensione infinita e ricchi di discontinuità, nei quali non sarebbe possibile applicare i comuni metodi basati sul gradiente. Si basa sulle leggi che governano l'evoluzione biologica (*e.g.* la selezione naturale di Darwin), per le quali una popolazione di individui migliora di generazione in generazione attraverso mutazioni e ricombinazioni genotipiche casuali unite a un meccanismo di selezione che privilegia gli individui migliori. Gli individui utilizzati nella GP sono **alberi sintattici**, ossia rappresentazioni di funzioni dove i nodi foglia costituiscono valori costanti o variabili di input, mentre quelli interni degli operatori. Per valutare la bontà delle candidate soluzioni migliori viene impiegata una funzione di fitness f, che, nel nostro caso, sarà da minimizzare quanto più possibile.

L'**obiettivo** di questa tesi è dimostrare sperimentalmente che un embedding in grado di trasformare opportunamente un pattern $w = (w_1, w_2, ..., w_n)$ in un unico valore w_e può effettivamente migliorare le prestazioni di un classificatore stand-alone (*i.e.* che riceve il dato originale w, senza embedding). Anche se il sistema realizzato è generale, nel nostro caso tratteremo la classificazione delle cifre, rappresentate in immagini di 8×13 pixel; l'obiettivo sarà realizzare 10 classificatori binari c_i (uno per ogni cifra i) e determinare per ciascuno di essi (con la GP) un embedding ottimo e_i che sostituisca l'input con la migliore rappresentazione latente. Per brevità, trattiamo della sola codifica delle immagini su quattro word a 32 bit ciascuna (che mostra prestazioni più elevate). L'immagine è divisa in quattro parti; i bit relativi ai pixel delle prime tre giacciono sui 24 bit meno significativi di tre word, mentre l'ultima parte occupa interamente la quarta word.

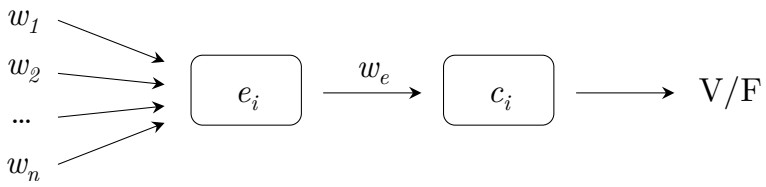

Figura 1: Classificatore con embedding dell'input

Il dataset sarà quindi composto da patterns di quattro numeri naturali ($n = 4$), ognuno etichettato con la classe di appartenenza (*Supervised Learning*), ossia la cifra che rappresenta.

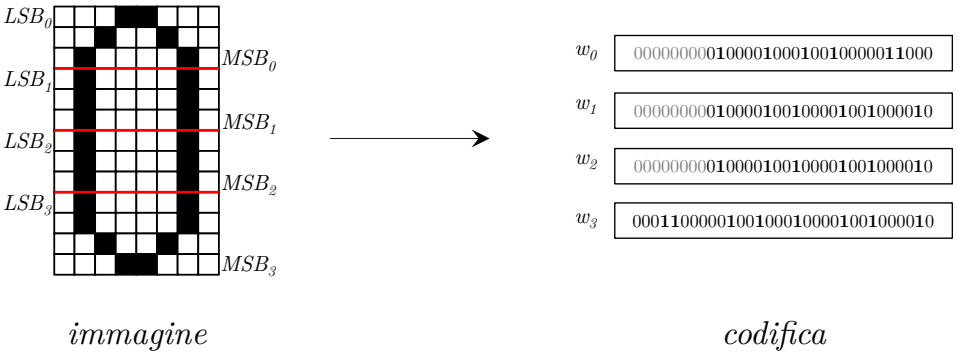

Figura 2: Codifica delle immagini (4 word a 32 bit)

Dividiamo il dataset in training set (per la fase di evoluzione), validation set (per prevenire l'overfitting) e test set (per la fase di testing). In seguito definiamo i nodi operatori (function set) da applicare agli input (w_i per l'embedding e w_e per il classificatore): AND, OR, NOT, NAND, NOR, XOR e CSH (*i.e. circular shift*, essenziale per variare le dipendenze fra bit). Infine, definiamo la funzione di fitness per la valutazione del classificatore binario. Siccome il risultato del corrispondente albero sintattico è un'intera parola a 32 bit, è come se si evolvessero parallelamente 32 classificatori, ma senza pesare sulla CPU (**Sub-Machine-Code GP**, che sfrutta il parallelismo intrinseco degli operatori bitwise); la valutazione $f(x)$ del classificatore sarà quindi il minimo fra le 32 valutazioni $f_{sub}(x, i)$ sull'i-esimo bit dell'albero sintattico x. Nello specifico,

$$f_{sub}(x,i) = \sqrt{50\frac{FP_{x,i}^2 + FN_{x,i}^2}{(NP+NN)^2}}, \tag{1}$$

con $FP_{x,i}$, $FN_{x,i}$, NP, NN rispettivamente numeri di falsi positivi, falsi negativi, positivi totali e negativi totali sul training set. In realtà, a f_{sub} è stata aggiunta una penalità che, a parità di obiettivo, induce l'algoritmo a preferire alberi sintattici di dimensione (numero di nodi) inferiore.

La strategia di determinazione della migliore coppia $C_i = (e_i, c_i)$ avviene tramite **coevoluzione**, un processo che si articola ciclicamente in tre fasi:

1. **fase E**, per la determinazione dell'embedding ottimo per il classificatore attualmente migliore;
2. **trasformazione dei dati**, per la trasformazione del training set tramite l'embedding appena determinato;
3. **fase C**, per la determinazione del miglior classificatore in cascata all'attuale miglior embedding.

L'algoritmo termina dopo aver raggiunto il massimo numero di iterazioni consentite, oppure se si entra in overfitting. Per non perdere le caratteristiche genetiche degli individui migliori, è stato introdotto l'**elitarismo**, ossia la sopravvivenza dell'individuo migliore nelle singole evoluzioni (tra una generazione e l'altra) e la sopravvivenza di parte della popolazione fra due iterazioni successive.

Confrontando i **risultati** ottenuti con quelli del classificatore stand-alone (sulle cifre più "ostiche"), i valori massimi, sia di TPR (True Positive Rate) che di TNR (True Negative Rate), sono stati superati; in particolare per il TNR (massimi e minimi), solo in un caso si scende al di sotto del 99%. Possiamo dunque concludere che un embedding ottimale può amuentare le prestazioni di un classificatore stand-alone. Per accrescerle ulteriormente, il prossimo passo potrebbe essere una **coevoluzione multialbero**, nella quale un individuo della popolazione sarebbe composto da più alberi sintattici.