Architetture e Programmazione dei Sistemi di Elaborazione
Progetto a.a. 2018/19

Algoritmo "*Product Quantization* for Nearest Neighbor Search" in linguaggio assembly x86-32+SSE e x86-64+AVX

Fabrizio Angiulli
fabrizio.angiulli@unical.it

1 Decrizione del problema

Dato un *dataset*, ovvero un insieme $Y \subset \mathbb{R}^d$ di n vettori d-dimensionali (detti anche *punti*), ed un punto $x \in \mathbb{R}^d$ (detto anche *query* o *interrogazione*), il *nearest neighbor* $\mathrm{NN}(x)$ di x in Y è il punto $y \in Y$ che minimizza la distanza Euclidea $dist(x, y)$ da x:

$$dist(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2},$$

ovvero:

$$\mathrm{NN}(x) = \arg\min_{y \in Y} dist(x, y).$$

L'approccio *brute force*, che consiste nel calcolare la distanza tra x ed ogni punto y di Y per poi restituire il punto y^* avente distanza minima, rappresenta la tecnica più semplice per il calcolo del nearest neigbhor. Tale approccio ha complessità $O(nd)$, dove n è il numero di punti in Y.

Nonostante siano state proposte in letteratura diverse tecniche volte e ridurre tale complessità, all'aumentare della dimensionalità d dei dati tali tecniche degenerano in una scansione sequenziale di tutti i punti di Y, risultando quindi non più efficienti dell'approccio brute force.

Poichè nelle applicazioni reali il problema della ricerca del nearest neighbor richiede di lavorare con dataset enormi, composti da un numero n di punti dell'ordine delle centinaia di migliaia o dei milioni e da un numero di dimensioni d dell'ordine delle centinaia, nella pratica anche un algoritmo di costo lineare in n ed in d risulta insoddisfacente.

Al fine di alleviare tale problematica, si può far ricorso alla ricerca dell'*approximate nearest neighbor* (ANN). Una tecnica di ricerca dell'ANN è un algoritmo che non garantisce di restituire l'esatto nearest neighbor del punto query, ma bensì solo una sua approssimazione, che in genere si assume sufficientemente accurata per gli scopi applicativi. Naturalmente, per avere senso una tale tecnica deve presentare un costo temporale inferiore all'approccio brute force.

Vector Quantization (VQ). Un *codebook* $\mathcal{C}$ è un insieme $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ di k vettori d-dimensionali $c_i \in \mathbb{R}^d$, detti anche *centroidi*. Un *quantizzatore vettoriale* (VC), o solo *quantizzatore* per semplicità, q è una funzione che mappa ogni punto $x \in \mathbb{R}^d$ in un centroide, ovvero

$q(x) \in \mathcal{C}$. In particolare, q (oppure $q_{\mathcal{C}}$ se si vuole enfatizzare l'insieme di centroidi $\mathcal{C}$ su cui q è definito) resituisce il centroide di $\mathcal{C}$ che risulta essere più vicino ad x:

$$q(x) = \arg\min_{c_i \in \mathcal{C}} dist(x, c_i).$$

Un buon insieme di centroidi $\mathcal{C}$ per un dataset Y è tale da minimizzare la somma delle distanze al quadrato tra ogni punto y di Y ed il corrispondente centroide $q_{\mathcal{C}}(y)$, ovvero

$$\mathcal{C}^* = \arg\min_{\mathcal{C} \subseteq \mathbb{R}^d : |\mathcal{C}| = k} \sum_{y \in Y} dist(y, q_{\mathcal{C}}(y))^2.$$

Determinare l'ottimo globale $\mathcal{C}^*$ per la precedente funzione obiettivo è un problema intrattabile. Nella pratica si utilizza come codebook un ottimo locale di tale funzione obiettivo, che può essere efficientemente calcolato utilizzando l'*algoritmo di clustering k-means*.

L'algoritmo k-means inizializza i centrodi $c_1, c_2, \ldots, c_k$ di $\mathcal{C}$ selezionando k punti causali di Y e poi procede in maniera iterativa. Ad ogni iterazione ogni centroide viene sostituito dal centro geometrico dei punti ad esso più prossimi. L'algoritmo converge quando il valore della funzione obiettivo in due iterazioni successive non supera una determinata soglia.

Product Quantization (PQ). Dato un dataset Y è di interesse riuscire a costruire un codebook $\mathcal{C}$ tale che, per ogni punto y di Y, y e $q_{\mathcal{C}}(y)$ sono molto vicini. Purtroppo all'aumentare della dimensionalità d per ottenere una tale proprietà la dimensione k del codebook dev'essere esponenziale in d.

Un *quantizzatore prodotto* (PQ) fornisce una soluzione efficiente al suddetto problema. Dato un parametro m (in genere con d multiplo di m), ogni vettore $x \in \mathbb{R}^d$ viene spezzato in m sotto-vettori a $d^* = d/m$ dimensioni. Nel seguito $u_j(x)$ denota il j-esimo ($1 \leq j \leq m$) sotto-vettore di x, composto dal j-esimo gruppo di d^* elementi consecutivi di x. I sotto-vettori di ogni gruppo j vengono quantizzati separatamente ottenendo m distinti quantizzatori vettoriali $q_1, q_2, \ldots, q_m$, detti anche sotto-quantizzatori. La quantizzazione di x si ottiene quindi come segue:

$$q(x) = (q_1(u_1(x)), q_2(u_2(x)), \ldots, q_m(u_m(x))),$$

ovvero è data dalla concatenazione degli m centroidi d^*-dimesionali restituiti dagli m distinti quantizzatori q_j applicati ognuno al rispettivo sotto-vettore $u_j(x)$, $j = 1, 2, \ldots, m$.

Si assume che i codebook $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ associati agli m quantizzatori $q_1, q_2, \ldots, q_m$ siano formati dallo stesso numero k^* di centroidi. Quindi, il numero totale k di centroidi che devono essere memorizzati da un PQ è dato da mk^*, mentre il numero totale di distinti centroidi che possono essere ottenuti mediante la loro concatenazione è di gran lunga più elevato, ovvero pari a $(k^*)^m$.

Ricerca ANN esaustiva. Utilizzando un PQ è possibile calcolare una distanza Euclidea approssimata tra il punto query x ed ogni punto y del dataset utilizzando due strategie: calcolo della *distanza simmetrica* (SDC) oppure calcolo della *distanza asimmetrica* (ADC).

La *distanza simmetrica* si ottiene come segue

$$dist(x, y) \approx dist_S(x, y) = dist(q(x), q(y)) = \sqrt{\sum_{j=1}^{m} dist(q_j(u_j(x)), q_j(u_j(y)))^2}. \quad (1)$$

Poichè $q_j(u_j(x)), q_j(u_j(y)) \in \mathcal{C}_j$, le distanze tra ogni coppia di centroidi $c', c'' \in \mathcal{C}_j$ possono essere precalcolate (ad un costo $O(mk^*d)$) e riutilizzate, per ogni punto query x, per calcolare la distanza simmetrica ad un costo ridotto, ovvero $O(m)$ anzichè $O(d)$.

La *distanza asimmetrica* si ottiene come segue

$$dist(x,y) \approx dist_A(x,y) = dist(x,q(y)) = \sqrt{\sum_{j=1}^{m} dist(u_j(x), q_j(u_j(y)))^2}. \quad (2)$$

In questo caso, dato un punto query x, le distanze $dist(u_j(x), q_j(u_j(y)))$ tra $u_j(x)$ ed ogni centroide $c' \in \mathcal{C}_j$ possono essere precalcolate e riutilizzate per calcolare la distanza asimmetrica ad un costo ridotto.

La differenza tra le due soluzioni è che nel caso della distanza simmetrica le distanze precalcolate sono indipendenti dalla query e quindi possono essere riutilizzate per ogni altra query, mentre la distanza simmetrica ha bisogno della query e quindi le distanze precalcolate non possono essere riutilizzate per altre query. Per contro, la distanza asimmetrica è più accurata di quella simmetrica.

Dato un parametro K, la ricerca restituisce i K punti del dataset che minimizzano la ADC oppure la SDC.

Ricerca ANN non esaustiva. La tecnica precedente consente di ridurre il costo della singola distanza, ma richiede che la query sia confrontata con la versione quantizzata di ogni punto del dataset. Quando il numero n di punti del dataset è molto grande si rende necessario ridurre anche il numero di punti da confrontare con la query, adottando una tecnica di *ricerca non esaustiva*.

A questo scopo si utilizzano due quantizzatori: un quantizzatore vettoriale cosiddetto *coarse* (ovvero grossolano) q_c (VQ) ed un quantizzatore prodotto (più accurato) q_p (PQ). Il quantizzatore vettoriale q_c utilizza k_c centroidi d-dimensionali $\mathcal{C}_c$ e viene utilizzato per definire il *vettore dei residui*

$$r(y) = y - q_c(y),$$

corrispondente al vettore y nel caso di origine dello spazio coincidente con il suo centroide $q_c(y)$, mentre il quantizzatore prodotto q_p corrisponde alla quantizzazione dei residui $r(y)$. Utilizzando q_c e q_p il vettore y può essere approssimato come segue

$$y \approx q_c(y) + q_p(y - q_c(y))$$

e di conseguenza la distanza $dist(x,y)$ tra x ed y può essere approssimata come segue

la distanza è tra x-qc(y) e qp(y-qc(y))

$$dist(x,y) \approx dist(x - q_c(y), y - q_c(y)). \quad (3)$$

Il quantizzatore q_p è unico per tutti i punti del dataset ed i suoi centroidi vengono determinati facendo uso di un campione R_Y di $n_r \leq n$ residui del dataset, ovvero $R_Y \subseteq \{r(y) : y \in Y\}$ e $|R_Y| = n_r$.

La tecnica di indicizzazione non esaustiva opera come segue: (i) si determinano i w centrodi grossolani $c_i \in \mathcal{C}_c$ che risultano essere più vicini alla query x; (ii) per ogni centroide c_i determinato al passo (i) si calcolano le distanze approssimate sfruttando il quantizzatore prodotto q_p — utilizzando l'Eq. (3) in congiunzione con l'Eq. (1) (SDC) oppure con l'Eq. (2) (ADC) — tra x ed ogni altro punto y tale che $q_c(y) = c_i$ e si collezionano i K punti associati alle distanze complessivamente più piccole; (iii) i K punti determinati al passo (ii) vengono restituiti come ANN approssimati della query x.

Per ulteriori dettagli si rimanda all'articolo [1] che descrive la tecnica di Product Quantization.

2 Descrizione dell'attività progettuale

Obiettivo del progetto è mettere a punto un'implementazione dell'algoritmo `PQNN` in linguaggio C e di migliorarne le prestazioni utilizzando le tecniche di ottimizzazione basate sull'organizzazione dell'hardware.

L'ambiente sw/hw di riferimento è costituito dal linguaggio di programmazione C (`gcc`), dal linguaggio assembly x86-32+SSE e dalla sua estensione x86-32+AVX (`nasm`) e dal sistema operativo Linux (`ubuntu`).

In particolare il codice deve consentire di effettuare sia la *ricerca ANN esaustiva* (parametro `-exaustive`, default) che la *ricerca ANN non esaustiva* (parametro `-noexaustive`). Inoltre, deve permettere di scegliere la distanza approssimata da utilizzare, ovvero la *distanza simmetrica* (parametro `-sdc`, default) oppure la *distanza asimmetrica* (parametro `-adc`).

Inoltre il codice deve supportare i seguenti parametri:

`-knn <K-value>`: numero K di ANN approssimati restituiti per ogni query x (default $K = 1$);

`-m <m-value>`: numero m di gruppi utilizzati dal quantizzatore prodotto (PQ) (default $m = 8$);

`-k <k*-value>`: numero di k^* centroidi utilizzati da ogni sotto-quantizzatore del quantizzatore prodotto (default $k^* = 256$);

`-kc <kc-value>`: numero k_c di centroidi utilizzati dal quantizzatore coarse q_c nel caso di ricerca non esaustiva (default $k_c = 8192$);

`-w <w-value>`: numero w di centrodi del quantizzatore coarse q_c da selezionare per effettuare la ricerca non esaustiva (default $w = 16$);

`-nr <n_r-value>`: dimensione n_r (default $n_r = \lfloor n/20 \rfloor$) del campione di residui utilizzati per costruire il quantizzatore prodotto q_p utlizzato nella ricerca non esaustiva;

`-kmeans <eps-value> [<t_min-value> [<t_max-value>]]`: soglia ϵ (default $\epsilon = 0.01$) per la terminazione dell'algoritmo k-means e numero minimo t_{min} (default $t_{min} = 10$) e massimo t_{max} (default $t_{max} = 100$) di iterazioni eseguite dall'algoritmo. Sia t il numero corrente di iterazioni dell'algoritmo k-means e sia Δ la differenza relativa tra la sua funzione obiettivo valutata in due iterazioni successive. La condizione di terminazione da utilizzare è la seguente: $(t_{min} \leq t)$ AND $\Big((t_{max} < t)$ OR $(\Delta \leq \epsilon)\Big)$.

Qualora un valore di un parametro (sia esso di default o specificato dall'utente) non sia applicabile, il codice deve segnalarlo con un messaggio e terminare.

Di seguito si riportano ulteriori linee guida per lo svolgimento del progetto:

- Si consiglia di affrontare il progetto nel seguente modo:
  1. Codificare l'algoritmo interamente in linguaggio C, possibilmente come sequenza di chiamate a funzioni;
  2. Sostituire le funzioni scritte in linguaggio ad alto livello che necessitano di essere ottimizzate con corrispondenti funzioni scritte in linguaggo assembly.

Ciò consentirà di verificare che l'algoritmo che si intende ottimizzare è corretto e di gestire più facilmente la complessità del progetto.

- Al fine di migliorare la valutazione dell'attività progettuale è preferibile presentare nella relazione un confronto tra le prestazioni delle versioni intermedie, ognuna delle quali introduce una particolare ottimizzazione, e finale del codice. Obiettivo del confronto è sostanziare la bontà delle ottimizzazioni effettuate.

- Occorre lavorare in autonomia e non collaborare con gli altri gruppi. Soluzioni troppo simili riceveranno una valutazione negativa. I progetti verranno messi in competizione.

- Sono richieste due soluzioni software, la prima per l'architettura x86-32+SSE e la seconda per l'architettura x86-64+AVX.

  Per i dettagli riguardanti la redazione del codice fare riferimento ai files `pqnn32c.c`, `pqnn32.nasm`, `runpqnn32` (versione `x86-32+SSE`) e `pqnn64c.c`, `pqnn64.nasm`, `runpqnn64` (versione `x86-64+AVX`) disponibili sulla piattaforma `didattica.dimes.unical.it`.

  Per l'interfacciamento tra programmi in linguaggio C e programmi in linguaggio assembly fare riferimento al documento allegato alla descrizione del progetto.

- Le soluzioni base non devono far uso delle istruzioni `OpenMP`. Opzionalmente, è possibile consegnare delle soluzioni aggiuntive che facciano uso anche di istruzioni `OpenMP`. I nomi dei relativi file dovranno contenere il suffisso "`_omp`" (es. `pqnn32c_omp.c`).

- Il software dovrà essere corredato da una relazione. Per la presentazione del progetto è possibile avvalersi di slide.

- Prima della data di consegna del progetto verranno pubblicate le convenzioni da rispettare riguardanti i nomi e la collocazione dei file/directory al fine della compilazione e l'esecuzione dei codici di programma mediante script appositamente predisposti. **Dato l'elevato numero di progetti, sarà cura del candidato accertarsi di aver rispettato pienamente le convenzioni di consegna.**

Buon lavoro!

Riferimenti bibliografici

[1] H. Jégou, M. Douze, C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Volume 33, Issue 1, Jan. 2011.