

Universitat Politècnica de Catalunya

FACULTAT D'INFORMÀTICA DE BARCELONA

# DISTÀNCIES *word2vec*

*Projecte Targetes Gràfiques (TGA)*

Antoni Casas Muñoz  
Pol Martín Garcia

Maig 2020

## Introducció

## Implementació

Hi ha dues versions correctament implementades del projecte, amb resultats finals equivalents.

Aquestes versions també són equivalents al codi seqüencial, trobat al fitxer *main.cpp* en la funció `sequentialSearch()`. Aquesta, donat un vector d'*embeddings*, l'índex d'un d'aquests, i un nombre *N*, troba d'entre tots els *embeddings* del vector els *N* més semblants a l'*embedding* identificat per l'índex.

## Algoritme

Qualsevol dels algorismes implementats per a solucionar aquest problema es basen en 3 parts.

1. Trobar la paraula en el vector d'*embeddings*. Aquest pas sempre es du a terme amb una cerca binària en CPU, per tant no el discutirem en aquest document.
2. Dur a terme el còmput de les distàncies de cosinus (o similituds de cosinus) entre tots els *embeddings* i l'*embedding* de la paraula cercada.
3. Filtrar els resultats, i escollir les *N* paraules més semblants (amb major similitud) a la paraula cercada amb les dades calculades, de manera ordenada.

Els punts 2 i 3, es troben tan implementats per a CPU, a *main.cpp*, com per GPU, a *kernel.cu*.

## Càlcul de similituds

El càlcul de les distàncies o similituds es du a terme a GPU pel kernel `DotProduct()`, el qual calcula el producte escalar amb cadascun dels *embeddings*, i posteriorment en divideix el resultat pel producte de normes.

Això es du a terme movent el *embeddings* de la paraula cercada a memòria *shared*, i posteriorment l'usen tots els *threads* del bloc per a dur a terme el producte escalar amb un altre *embedding*.

D'aquesta manera cada *thread* computa una sola similitud.

## Filtrat i ordenació

El filtrat per GPU requereix conèixer les *N* paraules amb una major similitud. Per a això s'ha dividit el còmput d'aquest procés en dues funcions.

La primera, `FirstMerge()`, divideix el vector de similituds resultants en trossos de *N* elements, els quals són ordenats usant ordenació per inserció donat que *N* sempre serà un nombre petit. L'ordenació es du a terme *on-place*, de manera que es reutilitza la mateixa memòria per emmagatzemar el resultat.

D'aquesta forma, obtenim el vector de similituds en trossos de *N* elements internament ordenats. Evidentment, a part d'aquest vector de similituds s'emmagatzema un vector de índex a les paraules originals, per no perdre la relació entre valor de similitud i la respectiva paraula.

Seguida i finalment la funció `BotchedMergeSort()` aprofita els segments ordenats per a dur a terme l'ordenació en una reducció del problema. Cada *thread* compara dos dels pedaços de *N* elements preordenats en un de sol.

Aquesta funció redueix el nombre de similituds a comparar a la meitat per cada crida, i es va usant fins que només resta un sol vector de *N* elements, el qual identifica les *N* paraules amb major similitud.

## Canvis de versió

Les millores en la segona versió del programa són separables en canvis en el codi del kernel, i en carrega de les dades a memòria.

Pel que fa al kernel, s'ha reduït l'espai de memòria reservat, utilitzant memòria local de cada *thread* per emmagatzemar l'ordenació temporal en el mètode `BotchedMergeSort()`. A més, s'ha afegit control d'errors complet.

Per altra banda, s'ha millorat substancialment la càrrega a memòria separant l'arxiu d'input en dos, un que conté els *strings* de les paraules, i un altre que conté les normes i els *embeddings* ja en binari, per estalviar la conversió a float en temps d'execució, a més que el fitxer és de menor mida en binari.

Finalment, s'ha afegit l'opció d'usar o no memòria *pinned* segons una *flag* de compilació.

## Resultats

### Possibles millores