Perceptron na GPU

B4M39GPU 2019/2020 zimní semestr

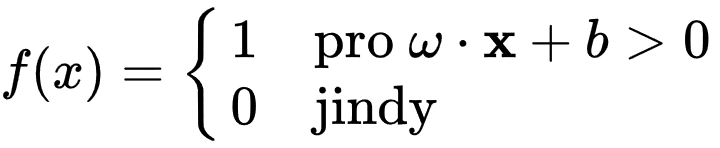
Jan Skala

# Abstrakt

Semestrální práce se zaměřuje na paralelní implementaci Perceptronu. Zahrnuje implementaci trénování a klasifikace dat. Obsahuje naivní CPU implementaci a paralelní GPU implementaci. Cílem je zjistit, zda se dosáhne, a pokud ano tak jakého, zrychlení výpočtu při použití grafické karty a paralelní implementace.

# Popis algoritmu

Perceptron je nejjednodušší model dopředné neuronové sítě. Sestává se pouze z jednoho neuronu a byl vynalezen Frankem Rosenblattem v roce 1957. Jedná se o binární klasifikátor, který mapuje vektor sloupců x na výstupní hodnoty f(x).



Rovnice 1 Klasifikace vektoru vstupů x

Kde **ω** je vektor vah a **b** je konstanta(někdy se jí také říká bias nebo práh) a **x** je vektor vstupních dat. Pro jednodušší implementaci se místo 0 vrací -1.

Hlavní nedostatek perceptronu spočívá v tom, že aby správně fungoval musí být data lineární separovatelná. Někdy lze tento nedostatek řešit částečnou úpravou vstupních dat. Máme-li dvourozměrná data a je patrná že existuje kružnice, která by data oddělila, můžeme data převést do polárních souřadnic. Dalším rozšířením je vícevrstevný perceptron, který má již mnohem širší možnosti použití. (Mařík, 1993)

## Trénování

V této práci bylo využito učení s učitelem. Síti byla nejdříve prezentována tréninková množina, která odpovídá správnému chování. Trénování probíhá v iteracích. Jejich počet můžeme stanovit. V každé iteraci je aktualizován vektor vah a bias. Zároveň pracujeme s konstantou **r**,která značí learning rate (rychlost učení, ale typicky se tento pojem nepřekládá).Dosazením menších hodnot docílíme pomalejšího učení, tzv fine tuning. Dosazením větších hodnot zrychlíme proces učení, ale riskujeme, že optimum nespočítáme správně. Hodnoty jsou v rozmezí od 0 do 1.

Nejprve je zvolen náhodný vektor vah (může se jednat o nulový vektor). Každý vektor vstupních dat klasifikujeme a porovnáme s očekávaným výsledkem. Pokud se výstupy neshodují provedeme upravíme váhy a bias. Učící algoritmus u perceptronu je konečný, pouze pokud řešení existuje, a navíc pokud je učící množina lineárně separovatelná.

Váhy perceptronu a bias jsou adaptovány dle následujícího pravidla:

# Datasety

Všechna data byla převedena do CSV pro snazší načítání. K trénování a testování byly použity následující Datasety:

* Iris
* Minst číslice 1
* Mnist číslice 2
* Mnist číslice 3

# Struktura projektu

* Data – složka obsahuje trénovací a testovací Datasety
* CL11.hpp – Header only knihovna pro zpracovávání vstupních argumentů
* CSV.cpp/h – pomocná třída, slouží k načítání CSV dat do struktury std::vector
* Debug.h – obsahuje skupinu funkcí pro debugování – vypisování dat z GPU do konzole
* default.h – obsahuje definici assert makra pro spouštění CUDA funkcí
* kernels.cu – implementace všech kernelů
* kernels.cuh – hlavičky a implementace spouštěcí funkcí pro kernely
* main.cpp – hlavní vstup programu – načítání argumentů a spuštění perceptronu
* Perceptron.cpp/h – Hlavní třída implementující funkcionalitu perceptronu

# CPU Implementace

CPU implementace je realizovaná ve třídě Perceptron. Konstruktor přijímá na vstupu 3 parametry:

* Počet iterací trénování
* Learning Rate
* Slovní výstup – true pokud požadujeme logování zpět do konzole

## Trénování

## Klasifikace

# GPU Implementace

GPU implementace je realizovaná ve třídě Perceptron. Konstruktor přijímá na vstupu 3 parametry:

* Počet iterací trénování
* Learning Rate
* Slovní výstup – true pokud požadujeme logování zpět do konzole

Zároveň využívá CUDA kernely a jejich pomocné a spouštěcí funkce.

## Trénování

## Klasifikace

# Použité Kernely

# Naměřené výsledky

CPU

GPU 1

GPU 2

# Závěr

# Reference

Mařík, V. (1993). *Umělá inteligence I.* Praha: Academia.