Debreceni Egyetem
Informatikai Kar

Hibrid fordítás HPC környezetben

Témavezető:
Dr Kovács László
adjunktus

Készítette:
Palkovics Dénes
mérnökinformatikus BSc

Debrecen, 2019

Tartalomjegyzék

Bevezetés

A Deep learning avagy a mélytanulás a gépi tanulás neurális hálózatokat alkalmazó technikája napjaink egyik legnépszerűbb technológiája, melynek fejlesztését számos kutató intézmény és nagyvállalat végzi. Megjelent a polgári életben is. Felhőalapú alkalmazások hátterében működik, többek között a Google®online szolgáltatásaiban és már alkalmazzák a hordozható eszközök, táblagépek és mobiltelefonok biometrikus személyazonosításra.

A mélytanulás használata rengeteg számítási kapacitást igényel —ez bizonyos alkalmazások esetén költséges és nagy méretű számítógépeket jelent— így sok helyen kiszorul a használata, illetve telemetria formájában érhető el csak. Az 5G-nek hála, komolyabb alkalmazásokhoz is felhasználható lesz a felhő technológián működő gépi tanulás. Ennek ellenére igény volna arra, hogy helyben elérhető legyen ez a technika. Ilyen lehet az orvosi alkalmazás, ahol számít a magas rendelkezésre állás vagy az autonóm robotok és önvezető autók, melyeknek bizonyos helyzetekben ott is kell működniük, ahol nincs rádiókapcsolat vagy internet elérés, nem is beszélve a hordozható eszközök olyan funkcióiról melyek használata frekventált.

Mikor fellendült a kutatása, legjobb hardverek erre a feladatra a fejlett grafikus kártyák voltak, melyek processzorainak számítási kapacitása és utasításkészlete alkalmassá tette, hogy a neurális hálózatokkal kapcsolatos számításokat hatékonyan végezze. Azonban az iparban megjelentek speciális hardverek kifejezetten neurális hálózatok futtatására optimalizálva, hogy ki tudják elégíteni a megnövekedett számítási igényt, amit a technológia egyre szélesebb körű bevezetése generál. A fenntarthatóság végett azonban nem szabad kihasználatlanul hagyni a már meglévő erőforrásokat. Témavezetőm, Dr. Kovács László projektje, a HuSSar nevet viselő hibrid architektúrájú szuperszámítógép is részben ebből az indíttatásból született. A HuSSar olyan hardverekből tevődik össze, melyek szerverek komponenseként régóta ott van az iparban. Egyedi hibrid architektúrája lehetőséget ad arra, hogy a neurális hálózatokkal kapcsolatos különféle számításokat olyan processzoron futtassuk, melyek azt optimálisan képesek végrehajtani így jelentős teljesítménynövekedés érhető el vele. Ehhez szükséges még egy olyan keretrendszer, mely képes ezeket a számításokat ekképpen optimalizálni.

A Deep learning a szemem láttára fejlődött ki a kezdeti kísérletekből, a mindennapi életben is használt csúcstechnológiává. Úgy érzem leendő szakemberként most van arra alkalmam, hogy közelebbről is megismerkedjek vele, kivehessem részem a fejlesztésében. Észrevettem,

hogy az ipar is nagy erőkkel fejleszti, ezért úgy vélem, hogy ez a tudás számomra nagyon jövedelmező lehet a munkaerőpiacon is. Témavezetőm fejlesztésével, a HuSSar-ral az egyik általa tartott egyetemi kurzus során találkoztam, mikor azt megmutatta nekünk. Beszélt az eszköz felépítéséről és arról, milyen célból kezdte a fejlesztést. Továbbá látom unokaöcsém sikereit, aki ezen a területen kutat. Ezek miatt éreztem úgy, hogy ebben a témában szeretnék dolgozni, ha lehet az egyetemi tanulmányaim után is.

Ebben a szakdolgozatban szeretnék beszámolni, mit sikerült megtudnom a Deep Learningről, milyen új megvalósítások születtek az iparban és hogyan boldogultam ezekkel a technológiákkal. Eredeti célkitűzésem az Intel®fejlesztés alatt álló *nGraph* nevű környezetének fordítása és telepítése volt a fentebb említett HuSSar-ra. Ez a keretrendszer kifejezetten a neurális hálózatok olyan módú futtatására lett fejlesztve, ahol a hardver több típusú processzort tartalmaz. Ezzel szerettük volna, ha sikerül a mélytanulás során alkalmazott neurális hálózatokat az összes processzortípuson elosztottan tanítani és futtatni. Hosszas próbálkozás után sem sikerült ez ügyben eredményt elérni, azonban a munka során megismerkedtem más az Intel®által fejlesztett és fejlesztés alatt álló eszközeivel.

1. fejezet

Neurális hálózatok és a Deep Learning

Ebben a fejezetben szeretném összegezni megszerzett tudásomat a neurális hálózatokról és a Deep Learning-ről, magyarul mély tanulásról.

1.1. A neurális hálózatok elmélete

Olyan számítási modellel, amelynek alapját az idegrendszer hálózata adja először Warren McCulloch és Walter Pitts 1943-ban foglalkozott az „A Logical Calculus of the Ideas Immanent in Nervous Activity” című publikációjukban. Később Donald Hebb tanulással kapcsolatos megfigyeléseivel elindultak a mesterséges neurális hálókkal kapcsolatos kísérletezések.[1]

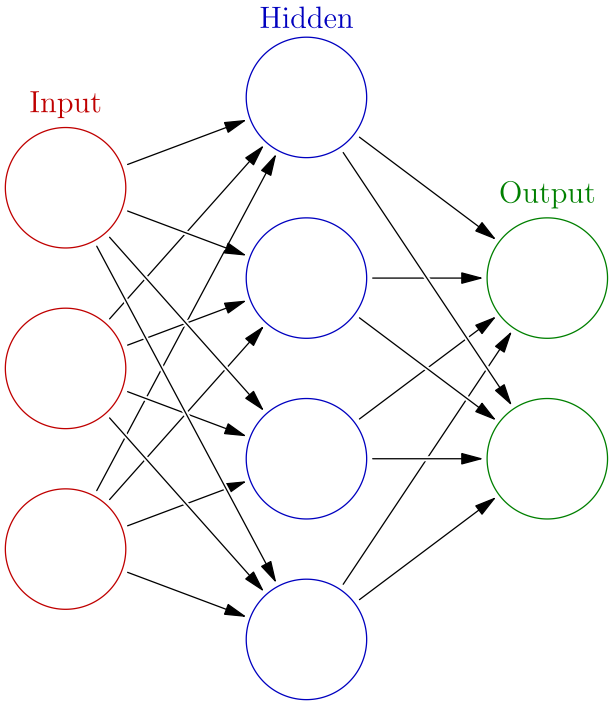
A mesterséges neurális hálózatok egy viszonylag egyszerű modellen alapulnak. Minden neuron a hozzá kapcsolódó neuronok ingereinek összessége alapján ingerli a többi neuront melyekhez ő kapcsolódik, ekképpen az ingerület egy irányba halad a kapcsolatok mentén.

Hogy a hálózat áttekinthető legyen, rendezzük a neuronokat rétegekbe úgy, hogy egy réteg neuronjai az ingerületet a közvetlen felső réteg neuronjaitól kapja, és a válasz ingert a közvetlenül alatta lévő réteg neuronjainak továbbítja. A 1.1 ábrán a csúcsok jelentik a neuronokat és az élek a szinapszisok, melyeken az ingerület vándorol. Egy hálózat 3 nagyobb részre tagolódik: **a)** bemeneti réteg **b)** rejtett rétegek **c)** kimeneti réteg. A bemeneti réteg csúcsai legtöbbször az adatot reprezentáló konstansok jelentik, tehát az egy egyszerű vektor. Egy neuronban két művelet történik: a bementek összegzése és egy aktiváló függvény kiértékelés. Az összegzést a felsőbb rétegből érkező jelekre elvégezzük:

$$s = \sum_i w_i x_i = \vec{w} \cdot \vec{x} \tag{1.1}$$

ahol x_i a felső réteg i.-ik neuronjának kimenete, w_i az i.-ik neuron szinapszisához tartozó súly, mellyel a szinapszis "erősségét" határozzuk meg. Az "s" összeghez hozzáadunk még egy b

[1] forrás: https://en.wikipedia.org/wiki/Artificial_neural_network



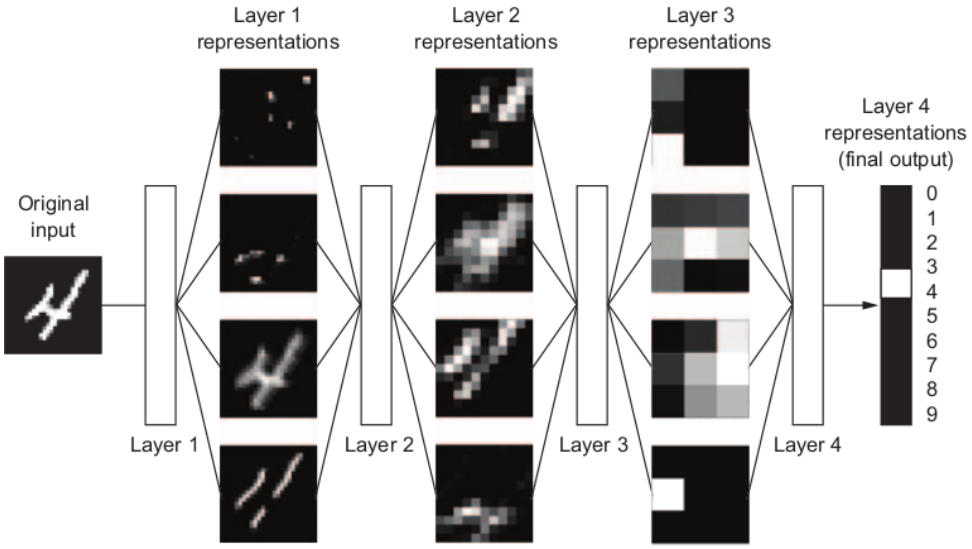
1.1. ábra. neurális hálózat réteges szerkezete [1]

értéket, a neuron aktiválási küszöbértéke lesz. Az aktivációs függvény adja a neurális hálózat kimenetét, paramétere $s+b$. A neurális hálózatok fejlesztésekor sokféle függvényt találtak alkalmasnak aktivációs függvény gyanánt. Közös jellemzőjük, hogy inflexiós pontjuk $x=0$ helyen van, illetve 0-ban nem deriválható függvények esetén a töréspont esik ide. Leggyakoribbak talán az egységugrás, a $\tanh(x)$, az ún. ReLU vagyis a $\max\{0,x\}$, szigmoid ($\sigma(x)=\frac{1}{1+e^{-x}}$) és a *softmax* függvények. Utóbbi kettőt az utolsó neuron réteg neuronjainak aktivációs függvényként szokták választani.

A szemléletesség kedvéért tekintsünk meg az egyrétegű perceptront, vagyis egy egyetlen rétegből álló neurális hálózatot k darab neuronnal. A bemenet legyen az $\vec{x}=(x_1,\ldots,x_n)$ vektor (a gyakorlatban bemeneti rétegként szokták hívni). A szinapszisok súlyait a $W=\{w_{ij}:i=1\ldots n, j=1\ldots k\}$ mátrix ($\vec{w}_i$ az i. bemeneti adatból kiinduló szinapszisokhoz tartozó súlyok vektora lesz), a neuronok küszöbértékeit a $\vec{b}=(b_1,\ldots,b_k)$ vektor tartalmazza. Az aktivációs függvény f. A hálózat kimenetét, vagyis a $\vec{y}=(y_1,\ldots,y_k)$ vektor elemeit megkapjuk a következőképpen:

$$y_i = f(\vec{w}_i \cdot \vec{x} + b_i) \tag{1.2}$$

A fentiekből látszik, hogy a hálózat tervezésénél annak négy tulajdonságát kell meghatároznunk: 1. a rétegek és azok neuronjainak számát 2. a neuronok aktivációs függvényét (rétegenként egy típusú függvény az összes neuronra) 3. a szinapszisok súlyát (W) 4. a neuronok aktiválási küszöbét ($\vec{b}$). Minden réteghez külön W mátrixot és $\vec{b}$ vektort kell meghatározni. Ez



1.2. ábra. Írott szám hozzárendelése az árbázolt számértékhez
[2]

nagyon sok külön meghatározandó változót jelent, tehát csak az 1. és 2. tulajdonság meghatározása elvárható. Kell egy algoritmus, mellyel w és b paraméterek sokasága meghatározható.

1.2. Deep Learning

A Deep Learning-ről szerzett tudásom javát F. Cholett könyvéből[2] szereztem, melynek a témához kapcsolódó részleteit alább bemutatok.

A gépi taunlás egy teljesen más programozási paradigmát jelent, ugyanis a klasszikus programozás során a feldolgozandó adatokhoz a programozó adja az adat feldolgozásának szabályait, amit végig követve a gép kiszámítja a kívánt eredményt. Ezzel szemben a gépi tanulás során a programozó az adathoz a kívánt erdményt adja meg, amiből a gép felálíttja a megoldáshoz vezető szabályokat.

A Deep Learning más néven a mély tanulás a gépi tanulás egy fajtája. Chollet szerint a név eredendően arra utal, hogy a kezdeti adaton több transzformációt végrehajtva egymás után egyre közelebb kerülünk egy olyan reprezentációhoz, ami megfelel a kívánalmainknak. Ezzel kontrasztban beszélhetünk sekély tanulásról, amikor kevés, egy vagy két transzformáció után kapjuk meg az adat megfelelő reprezentációját.A neurális hálózatok rétegeltsége adja a *mélységet* a gépi tanulásban. Eredeti elgondolás szerint minden egyes neuron-réteg egyre összetettebb tulajdonságokat ismer fel a bemeneti adatból. Valójában a rétegenkénti transzformációk egyre kisebb összetettségű hipotézis térbe visznek át, a reprezentáció egyre kevesebb a felhasználó számára fölösleges információt tartalmaz. Minél több a réteg a hálózatban, annál *mélyebb* a modell.

Itt kapcsolódik össze a neurális hálózat és a mély tanulás. Az 1.1 szakaszban kifejtettem, hogy a neurális hálózat szinapszisainak paraméterezéséért felelős W mátrixok és a neuronok küszöbszintjének állítására szolgáló $\vec{b}$ vektorok változóinak száma hatalamas lehet — alkalmazástól függően több százezer, akár millió, egymástól független változóról beszélünk—, tehát beállításukhoz valamilyen algoritmusra van szükség. Ezért a neurális hálózatok másik komponense, egy tanulási algoritmus, mely beállítja ezen paramétereket. Négy megközelítés létezik, amikor gépi tanulásról van szó. *Ellenőrzött tanulás* során a neurális hálózatnak felcímkézett adatokat adunk meg, tehát olyan értéket rendelünk hozzájuk, amilyet szeretnénk, hogy a hálózat produkáljon. A hálózat leképezi az adatot a meghatározott reprezentációvá. A tanuló algoritmus ebből és a címkéből egy *veszteség függvény* kiszámításával meghatározza, hogy mekkora az eltérés, a valamilyen értelemben vett távolság a kapott és az elvárt eredmény között. Ez alapján frissíti a W mátrixokat és $\vec{b}$ vektorokat. *Ellenőrizetlen tanulás*, mely során az adatokat nem címkézzük fel, hanem arra vagyunk kíváncsiak, hogy miféle összefüggések vannak közöttük. Ezt a módszert adatbányászat során alkalmazzák. Az *Önellenőrzött tanulás* hasonló az ellenőrzötthöz, azonban az adatok felcímkézését nem emberi erővel végezzük, hanem az adatokból állítjuk elő valamilyen heurisztikát felhasználva. Egyik alakalmazási területe az autóenkóderek tanítása. *Megerősítéses tanulás* egy újfajta megközelítése a neurális hálózatok alkalmazásának. Ennél a metodikánál a hálózatot egy ágens alkalmazza, így a hálózat bemenete az ágens által megfigyelt környezet a kimenete pedig valamilyen cselekedet, beavatkozás és tanítás során az ágens igyekszik valamilyen környezetbeli értéket maximalizálni. Gyakori alkalmazás valamilyen játékot játszó ágens, ahol azt tanulja, adott helyzetekre milyen reakcióval tudja maximalizálni játékbeli pontszmát. Vizsgálódásomat az *ellenőrzött tanulásra* korlátoztam, így a továbbiakban ennek tükrében folytatom dolgozatomat.

1.3. Függvények, algoritmusok

1.3.1. Neuronok aktivációs függvényei

1.3.2. Veszteség függvények

1.3.3. „Backpropagation” algoritmus

1.3.4. A kernel-trükk

[2]Forrás:[2]

2. fejezet

Mélytanulásos alkalmazások fejlesztése

2.1. Keras

Keras Keras Keras Keras Keras

2.2. TensorFlow

3. fejezet

Specializált hardver megjelenése

3.1. hybrid deep learning

A ha

3.2. nGraph

A most leírtak alapját az nGraph hivatalos dokumentációja adja[3]. Az nGraph az Intel®által fejlesztett programkönyvtár és egy futtatási környezet/fordító készlet melyet mélytanulásos projektekhez. Legszembetűnőbb tulajdonsága, hogy képes többféle hardver architektúrán futtatni és beépíthető számos keretrendszerbe. Ezek azok a jellemzők, amiket kerestünk témavezetőmmel, hogy mélytanulást tudjunk végeztetni hatékonyan a *HuSSar-on*. Ezen túlmenően a jövőbeli fejlesztések az iparban egyre jelentősebben igényelik, hogy a modern MI rendszerek skálázhatóak legyenek, mert egyrészt a neurális hálók egyre komplexebbé válnak, másrészt az általuk feldolgozott adatok mennyisége is rohamosan növekszik.

Jelenleg két bevett gyakorlat van a mélytanulás felgyorsítására:

1. **Dedikált hardver tervezése a mélytanulásos számításokhoz** – Sok vállalkozás tervez *Alkalmazásspecifikus integrált áramköröket* (ASIC) neurális hálózatok betanítására és futtatására.

2. **Szoftver optimalizáció** – Programkönyvtárakat tartalmazó fejlesztési keretrendszerek fejlesztése, melyek képesek a hálózatokkal kapcsolatos számításokat több szálon, optimalizáltan futtatni. Az nGraph, mint fordító is egy ilyen megoldás.

3.2.1. Motiváció

A mai legmodernebb szoftveres megoldás mélytanulásra az, ha integrálunk kernel könyvtárakat[1] mélytanulásos keretrendszerekbe. Ilyen integráció lehet például ha a Tensorflow keretrendszer alatt az Nvidia CuDNN könyvtárát használjuk. A kernel könyvtárak egy adott célarchitektúrára optimalizált kernelekből és egyéb műveleti szintű optimalizálásokból állnak, ezekkel érve el teljesítménynövekedést. Azonban a kernel könyvtáraknak három fő problémája van:

1. Nem biztosítanak gráf szintű optimalizálást

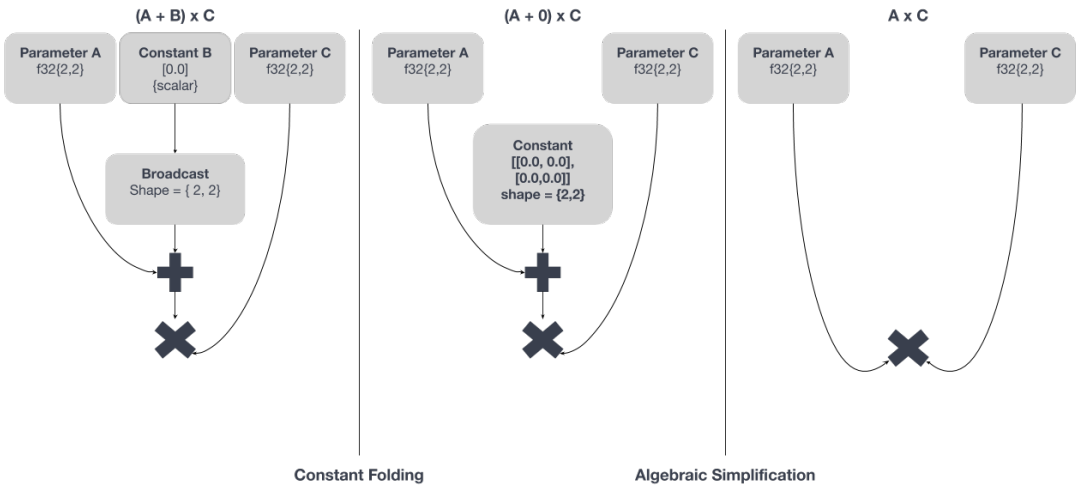
2. A keretrendszerek integrációja a kernel könyvtárakkal nem skálázható

3. A szükséges lefordított kernel könyvtárak száma növekszik, ahogy új processzorok jelennek meg

Az nGraph fordító megoldás az első két problémára, és a *PlaidML*-el ötvözve kiküszöbölhető a harmadik probléma.

3.2.2. Gráf szintű optimalizálás

Az nGraph dokumentációjában áll egy példa arra vonatkozóan, hogyan lehet, hogy egy mélytanulásos keretrendszer integrálva egy kernel könyvtárral a számításokat optimálisan végzi, mégis a számítási gráf a csúcsait alkotó műveletek szempontjából mégsem optimális. A fenti ábra be-



3.1. ábra. két optimalizálás módszer: konstans összehajtás és algebrai egyszerűsítés a gráfon. [2]

[1] Programkönyvtár, mely neurális hálókkal kapcsolatos elemi, *mag* függvényeket tartalmazzák

mutatja, hogyan egyszerűsíthetünk az A,B és C tenzorokat feldolgozó, $(A+B)*C$ tenzorműveletet végrehajtó számítási gráfon. Fordítási időben megállapítható, hogy B egy skalár konstans, így a *konstans összehajtásnak* nevezett optimalizálás elvégezhető, és a 2 dimenziós vektorrá való kiterjesztés művelete elhagyható (helyette inkább közvetlenül létrehozunk egy 2×2 tenzort). Ebben a példában $B = 0$ skalár volt, így a belőle létrejött tenzor egy nullmátrix, így az semleges a kifejezés kiértékelésének szempontjából. *Algebrai egyszerűsítést* végezve az $(A+0)*C$ leegyszerűsíthető az $A*C$ kifejezésre, így összesen két csúccsal csökkentettük a gráfunkat. Ez az optimalizáció tehát a számítási gráf szintjén lett elvégezve. Így belátható, hogy a mélytanulásos keretrendszerbe integrált kernel programkönyvtárak nem optimális futást végeznek, hiába a műveletek szintjén elért optimalizálás.

3.2.3. Skálázható keretrendszer integráció

Ahogy gyarapodik a mélytanuláshoz használható gyorsítókártya architektúrák és keretrendszerek száma, a meglévő mélytanulást alkalmazó fejlesztési platformok bővítése egyre több munkát igényel és egyre nő a hibák megjelenésének a valószínűsége. Az integráció kapható készen, szakértő fejlesztőcsapatoknak kell implementálnia. Minden új keretrendszert manuálisan kell integrálni a meglévő hardverek kernel könyvtárával és minden újonnan megjelenő hardvercsalád meghajtó programkönyvtárát be kell integrálni egyesével a meglévő keretrendszerekbe. Ez a munka önmagában is hatalmasra tud nőni, de egy sok eszközből álló összeállítás nagyon törékeny és költséges a fenntartása. Az nGraph úgy oldja meg ezt a problémát, hogy ún. *hidakat* alkalmaz, amikkel integrálható valamelyik mélytanulásos keretrendszerbe. A híd megkapja a keretrendszerben megalkotott számítási gráfot vagy ahhoz hasonló struktúrát és átalakítja egy ún. *közbenső reprezentációvá*[3]. Ezzel kaptunk egy egységes, platformfüggetlen számítási gráfot, így nem kell egy új programkönyvtárat beintegrálni minden egyes meglévő keretrendszer alá, elegendő csak az, hogy az nGraph-ban, mint programkönyvtárban implementált *primitív műveleteket* támogassa az új programkönyvtár.

3.2.4. Növekvő kernel szám

Egy kernel könyvtár integrálása egyszerre több mélytanulásos keretrendszerrel nehéz feladat és egyre komplexebbé válik, ahogy növekszik az optimális teljesítményhez szükséges kernelek száma. Régen a mélytanulásos kutatások egy kis számú *primitív* számítást használtak, mint a konvolúció, általános mátrixszorzás, stb. Az MI kutatás előrehaladtával és az ipari mélytanulásos alkalmazások továbbfejlesztésével, a szükséges kernelek száma (k) exponenciálisan nő. Ez

[2] forrás: [3]
[3] IR: Intermediate Representation

a szám a processzor architektúrák számán (h), adattípusokon (t), műveleteken (p) és az egyes paraméterek számosságán (p) alapul ($k = h \times t \times m \times p$).

Hardver	Művelet	Adattípus	Paraméterek
CPU	konvolúció	16 bites lebegőpontos	NCHW vagy NHWC
GPU	MatMul	32 bites lebegőpontos	2D, 3D és 4D tenzorok
FPGA	Normalizálás	8 bites egész	...
...	...	...	

3.1. táblázat. Néhány példa, tényezőnként hányféle esetre kell külön fordítani kernelt könyvtárt

Ezen probléma megoldásához jön képbe a PlaidML. Ez egy *tenzor fordító*[4], mely azt célozza, hogy képes legyen neurális hálózatokat tanítani és futtatni bármilyen típusú hardveren. Más szavakkal segíti a magas szintű keretrendszerek (Keras, ONNX, nGraph) integrálni olyan ezsközökkel, melyekhez nincs meg a szükséges támogatás vagy a meglévő szoftverkészlet hozzájuk szigorúan linceszelt.[4][5]

Az nGraph tehát integrálható a PlaidML-el. Elsősorban az nGraph a platform független IR-rel igyekszik orvolsoni a skálázható backend-el kapcsolatos kihívást. A PlaidML ezt megtámogatja azzal, hogy képes az IR-ből származó gráfokból LLVM, OpenCL, OpenGL, CUDA és Metal kódot generálni melyek a megfelelő hardveren futtathatóak. Így egy magas szintű keretrendszerben írt neurális háló lefordul Intel és AMD processzrokon valamint grafikus processzorokon, az nVidia processzorain, továbbá az Apple cég által feljelsztett eszközökön.

Az nGraph gráf szintű optimalizációját ráadásul kiegészíti automatikusan a PlaidML alacsonyabb szinten, ezzel teljesítmény növekedést érve el.

Összegzésül tehát az nGraph feldarabolja a neurális hálózathoz tartozó számítási gráfot processzor architektúrának megfelelően, majd ezen gráfokat a PlaidML lefordítja a megfelelő kódokra, melyeket aztán a célprocesszorokra lefordítunk és futtatunk.

3.3. Myriad X és az Intel Neural Computer Stick 2

3.4. Google ???

3.5. Új gyorsítók: Intel Nervana Neural Network Processor

[4]Olyan fordító, melynek nyelve arra lett fejlesztve, hogy főleg tenzorműveleteket igénylő számításokat tudjunk hatékonyan programozni

Összefoglalás

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Irodalomjegyzék

[1] Altrichter Márta & Horváth Gábor & Pataki Béla & Strausz György & Takács Gábor & Valyon József. *Neurális hálózatok*. Panem, 2006. [Szakkönyv].

[2] Francois Chollet. *Deep Learning with Python*. Manning, 2018.

[3] Many. Introduction — documentation for the nGraph library and compiler stack, 2019. [meglátogatva 2019. október 08.].

[4] Many. Plaidml. `https://github.com/plaidml/plaidml`, 2019.

[5] Vertex.AI. Plaidml – home. `https://vertexai-plaidml.readthedocs-hosted.com/en/stable/`, 2018.

Függelék

F.1. Példa