

Politechnika Warszawska

W Y D Z I A Ł E L E K T R Y C Z N Y



Instytut Sterowania i Elektroniki Przemysłowej

Praca dyplomowa inżynierska

na kierunku Informatyka Stosowana

w specjalności

Porównanie wydajności wybranych języków programowania w realizacji sieci neuronowych do przetwarzaniu obrazów

Piotr Heinzelman

numer albumu 146703

promotor

dr inż. Witold Czajewski

WARSZAWA 2024

Porównanie wydajności wybranych języków programowania w realizacji sieci neuronowych do przetwarzaniu obrazów

Streszczenie

W niniejszej pracy porównano języki programowania wraz z ich środowiskami w zastosowaniach tworzenia głębokich sieci neuronowych w dziedzinie widzenia komputerowego, czyli implementacji sieci CNN oraz MLP. Omówiono wybrane języki: Python, Matlab, Java, C++. Do budowania sieci wykorzystano biblioteki: YOLO11, (TensorRT? PyTorch) oraz TensorFlow, Scikit-Learn dla Python. LibTorch dla C++, Deep Learning Toolbox dla Matlab oraz własne implementacje dla języka Java. W porównaniu ujęto również takie własności języków jak: popularność, dostępność bibliotek, wygoda instalacji, wsparcie obliczeń na kartach graficznych, stopień trudności języka, koszt licencji.

Słowa kluczowe: uczenie głębokich sieci neuronowych, sieci splotowe CNN, YOLO, wydajność, klasyfikacja obrazów, obliczenia równoległe

Comparison of the performance of selected programming languages in the implementation of neural networks for image processing

Abstract

This paper compares programming languages and their environments in applications of creating deep neural networks in the field of computer vision, i.e. implementing CNN and MLP networks. Selected languages are discussed: Python, Matlab, Java, C++. The following libraries were used to build the network: YOLO11, (TensorRT? PyTorch) and TensorFlow, Scikit-Learn for Python.

LibTorch for C++, Deep Learning Toolbox for Matlab and our own implementations for Java.

The comparison also includes such language properties as: popularity, availability of libraries, convenience of installation, support for computing on graphics cards, level of language difficulty, and license cost.

Keywords: deep learning, convolution network, image classification, efficiency, parallel computing

Spis treści

1	Wprowadzenie	9
1.1	Cel pracy	10
1.2	Układ pracy	10
1.3	Kod źródłowy i dane uczące	10
2	Perceptron wielowarstwowy - MLP	11
2.1	Propagacja sygnału	11
2.2	Propagacja sygnału wstecz - uczenie sieci	13
3	Sieć splotowa CNN	15
3.1	Operacja splotu	17
3.2	ReLU i warstwy redukujące wymiar	18
3.3	Przetwarzanie w przód	18
3.4	Modyfikacja filtra w warstwie splotowej	18
3.5	Propagacja błędu przez warstwę splotową	19
3.6	Yolo 11 / 12	19
4	Przetwarzanie równoległe	21
4.1	Maszyna Turinga	21
4.2	Szybsze obliczenia grafiki 3D	21
4.2.1	Funkcje procesora MMX, SIMD, SSE, AVX	22
4.3	Obliczenia na GPU	23
4.4	Dodawanie równoległe	23
5	Prezentacja języków	25
5.1	Wstęp	25
5.1.1	Popularność języka	25
5.1.2	Dostępność bibliotek	25
5.1.3	Wygodna instalacja	25
5.1.4	Wykorzystanie GPU	25
5.1.5	Koszt licencji	26

5.1.6	Łatwość użytkowania	26
5.2	Matlab	26
5.3	Python	26
6	Porównanie języków	27
7	Jednowymiarowa regresja liniowa	29
7.1	Wprowadzenie	29
7.2	Prowadzenie badania	30
7.3	Kod realizujący obliczenia	30
7.4	Uzyskane wyniki	31
8	Klasyfikowanie pisma odręcznego MLP	33
8.1	Wstęp	33
8.2	Dane treningowe	34
8.3	Uzyskane wyniki	35
8.4	Kod realizujący obliczenia w Matlab	36
8.5	Własna implementacja w Java	37
8.6	Pyton -sklearn	40
8.7	Pyton -tensorflow	41
8.8	C++ -on ? CUDA ?	43
9	Sieci głębokie	45
9.1	Wstęp	45
9.2	Struktura sieci CNN	45
9.3	Podstawowe operacje w sieciach CNN	46
9.4	Inne rodzaje warstw	46
9.5	Przetwarzanie wprost	46
9.6	Przetwarzanie wstecz	47
9.7	Uzyskane wyniki	47
9.8	Kod realizujący obliczenia w Matlab	48
9.9	CNN	51
9.10	Python TensorFlow	51
10	Podsumowanie	53
	Bibliografia	55
	Spis rysunków	55

Rozdział 1

Wprowadzenie

Zainteresowanie sztucznymi sieciami neuronowymi w środowisku naukowców jak i inżynierów wynika z potrzeby budowania bardziej efektywnych i niezawodnych systemów przetwarzania informacji wzorując się na metodach jej przetwarzania w komórkach nerwowych. Fascynacje mózgiem człowieka i jego właściwościami w latach 40-tych dały początek pracom w zakresie syntezy matematycznej modelu pojedynczych komórek nerwowych, a później na tej podstawie struktur bardziej złożonych w formie regularnych sieci. Zespół kierowany przez prof. Ryszarda Tadeusiewicza z AGH w Krakowie prowadził już w latach '80 badania nad m.in. rozpoznawaniem ręcznego pisma, czy sterowaniem ruchem robota, a pierwsza książka ukazała się w 1993r.[**korbicz1994**]

Pierwsze prace prof. Kuniyuko Fukushimy nad głęboką siecią pojawiły się w 1975 roku. Jednak prawdziwy rozwój sieci głębokich zawdzięczamy profesorowi Yann A. LeCun, który zdefiniował podstawową strukturę i algorytm uczący sieci konwolucyjnej CNN. Aktualnie sieci CNN stanowią podstawową strukturę stosowaną na szeroką skalę w przetwarzaniu sygnałów i obrazów. W międzyczasie powstało wiele odmian sieci będącej modyfikacją struktury podstawowej (R-CNN, AlexNET, GoogLeNet, ResNet, U-Net, YOLO)[**ossowski2023**]

Ważnym rozwiązaniem jest sieć YOLO (ang. You Only Look Once), wykonująca jednocześnie funkcje klasyfikatora i systemu regresyjnego, który służy do wykrywania określonych obiektów w obrazie i określaniu ich współrzędnych. Obecnie dostępna jest już wersja 12.

Biblioteki dostarczające implementacje modeli sieci neuronowych powstały dla większości języków programowania ogólnego przeznaczenia. Niektóre z nich wykorzystują procesory graficzne do wysokowydajnych równoległych obliczeń, co powoduje gwałtowny wzrost wydajności i drastyczne obniżenie czasu uczenia sieci. Budowanie własnych modeli sieci jest dziś w zasięgu osób prywatnych, hobbystów, studentów czy małych zespołów badawczych i nie wymaga ogromnych nakładów finansowych.

1.1 Cel pracy

Podstawowym celem pracy jest ułatwienie podjęcia decyzji o wyborze języka i środowiska w fazie projektowej dla realizacji aplikacji wykorzystujących głębokie sieci neuronowe CNN.

Celem dydaktycznym jest dogłębne zapoznanie się z tematyką sieci MLP [korbicz1994], [wit1986] oraz CNN [MMuraszkiewiczRobertNowak] [russell2023] poprzez realizacje i testy własnego rozwiązania zwłaszcza z wykorzystaniem możliwości obliczeniowych karty graficznej.

1.2 Układ pracy

Problem obliczeniowy - W części pierwszej opisano stan wiedzy z zakresu działania głębokich sieci neuronowych tj. Perceptronu wielowarstwowego (ang. Multilayer Perceptron, MLP) oraz Konwolucyjnej sieci neuronowej (ang. Convolutional Neural Network, CNN). Zaprezentowano propagację sygnałów przez sieć, propagację wsteczną i oparty na niej proces uczenia sieci.

Problem wydajnościowy - Warunkiem niezbędnym do prowadzenia efektywnych badań nad głębokimi sieciami i dużymi modelami jest zdolność efektywnego wykorzystania systemów o dużych mocach obliczeniowych. W drugiej części opisano metody zwiększania wydajności systemów cyfrowych i zagadnienia przetwarzania równoległego.

W drugiej części przedstawiono języki, opisano wybrane cechy, informacje o wykorzystanych bibliotekach, pokazano fragmenty kodu.

W trzeciej części zastawiono cechy i wyniki pomiarów z podziałem na języki.

Wnioski na temat wyższości jednego rozwiązania nad drugim będą należeć od konkretnych twórców sieci i będą zależały od ich wymagań i możliwości.

1.3 Kod źródłowy i dane uczące

Przykłady rozwiązań w Python i Matlab zaczerpnięto z [ossowski2023] [ossowski2020] [guido2021] [raschka2021], a także z instrukcji i przykładów załączonych do wykorzystanych bibliotek.

Obrazy pisma odręcznego pochodzą z bazy MNIST (yann.lecun.com), Zdjęcia twarzy wykorzystane w treningu sieci osób pochodzą z internetu - głównie z serwisów google.com oraz filmweb.pl

Pełen kod dostępny na github: <https://github.com/piotrHeinzelman/inz/tree/main/MixedProj>

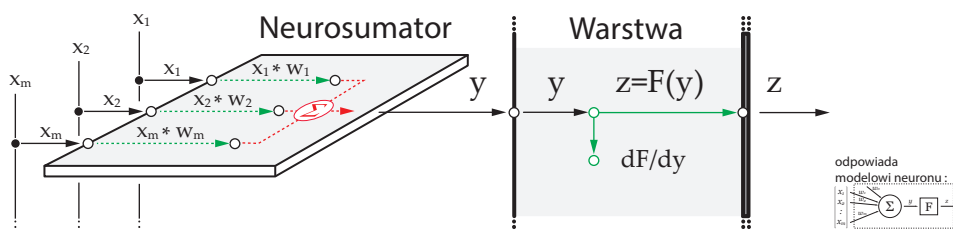
W analizie nie brano pod uwagę czasów czytania plików oraz przygotowania danych.

Rozdział 2

Perceptron wielowarstwowy - MLP

2.1 Propagacja sygnału

Model neuronu [korbicz1994], rys. 9 to wzorowany na komórce nerwowej przetwornik sygnałów. W modelu sygnały wejściowe i wyjściowe mają charakter ciągły tj. przyjmują wartości z określonych zakresów, co implikuje działania na wartościach zmiennoprzecinkowych kodowanych najczęściej w standardzie IEEE 754 i odpowiadających im typom *float* lub *double*. Dla czytelności rozbito neuron na Neurosumator oraz Warstwę będącą kontenerem neurosumatorów.



Rysunek 1. Model sztucznego neuronu - propagacja sygnału.

Proces propagacji sygnału przez pojedynczą komórkę w modelu polega na wyznaczeniu sumy ważonej sygnałów wejściowych:

$$y = \sum_{i=0}^m x_i * w_i \quad (1)$$

gdzie: x_i - i-ty sygnał wejściowy, w_i - i-ta waga w neuronie, wartość progowa (Bias): w_0 dla $z_0 = 1$

A następnie na wyliczeniu wartości funkcji aktywacji z , będącej wartością wyjściową neuronu.

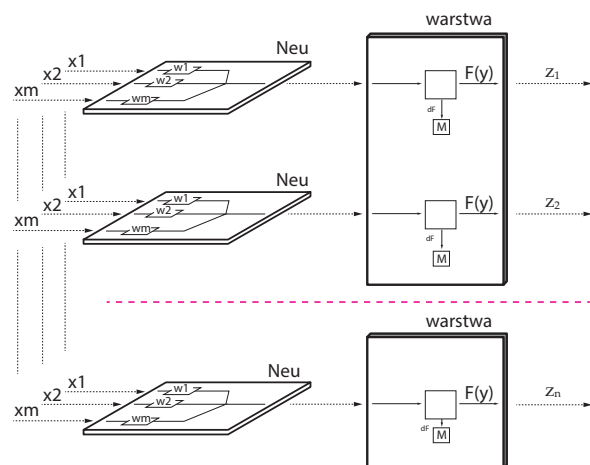
$$z = F(y) \quad (2)$$

gdzie: F - funkcja aktywacji, y - suma ważona, z - wartość wyjściowa.

W kodzie dla zwiększenia wydajności zapamiętano wartość pochodnej $\frac{\partial F}{\partial y}$ w punkcie z . Zostanie ona wykorzystana w późniejszych obliczeniach w procesie uczenia.

rodzaj warstwy	funkcja aktywacji $z = F(y)$	wartość pochodnej $\frac{\partial F}{\partial y}$
logistyczna (sigmoid)	$\frac{1}{1+e^{-y}}$	$(z)(1-z)$
ReLU	jeśli $y < 0$ to $z=y$, jeśli $y \geq 0$ to $z=0$	jeśli $y < 0$ to 1, jeśli $y \geq 0$ to 0
softmax	$z_i = \frac{e^{x_i}}{\sum e^{x_i}}$	dla prawidłowej klasy k : $y_k(1-y_k)$, dla pozostałych klas: $-y_i * y_k$

Wejście neuronu przyjmuje wartości $[x_1, x_2, \dots, x_m]$ (wektor), natomiast na wyjściu jednego neuronu pojawia się tylko jeden sygnał wyjściowy z .

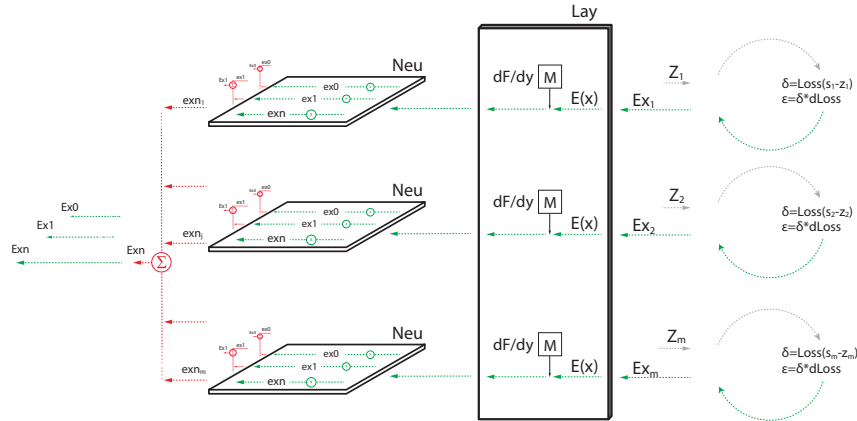


Rysunek 2. Przepływ sygnałów przez warstwę w czasie propagacji sygnału "w przód"

Odpowiedzią n neuronów zorganizowanych w warstwę jest zbiór wartości $[z_1, z_2, \dots, z_n]$ (wektor). Odpowiedzią warstwy typu „softmax” jest wektor rozkładów prawdopodobieństwa przynależności do klas.

Jeśli warstwa jest zbyt duża dla jednej jednostki obliczeniowej, można dokonać podziału na większą liczbę jednostek obliczeniowych, a przy podziale w miejscu oznaczonym na rysunku spadek wydajności będzie najmniejszy.

2.2 Propagacja sygnału wstecz - uczenie sieci

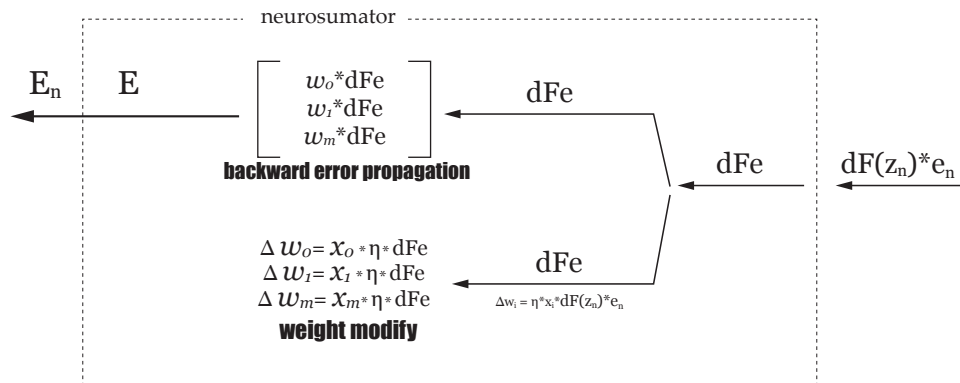


Rysunek 3. propagacja wstecz - uczenie sieci. (Operacje niezależne (mnożenie) oznaczone kolorem zielonym i operacje wymagające synchronizacji (dodawanie) oznaczone kolorem czerwonym)

Odpowiedzią sieci jest sygnał wyjściowy ostatniej warstwy. W procesie uczenia z nauczycielem podlega on ocenie, a wielkość błędu sieci jest przekazywana do warstwy wyjściowej.

Dla warstwy wyjściowej logistycznej może to być różnica między oczekiwaną odpowiedzią a odpowiedzią uzyskaną. $[e_1 = s_1 - z_1, e_2 = s_2 - z_2, \dots, e_n = s_n - z_n]$, lub w zapisie wektorowym: $E = S - Z$, lub gdzie S jest wielkością oczekiwaną. Dla warstwy typu softmax wyliczenie jest nieco bardziej skomplikowane.

Dla każdej wartości wyjściowej z_i zostaje zwrócona wielkość błędu e_i , ponadto dla każdej wartości z_i zapamiętano wcześniej wartość pochodnej $\frac{\partial F}{\partial y}$. W procesie uczenia do każdego z neurosumatorów zostaje przekazana odpowiadająca mu wielkość błędu: $dFe = e_i * \frac{\partial F}{\partial y}$.



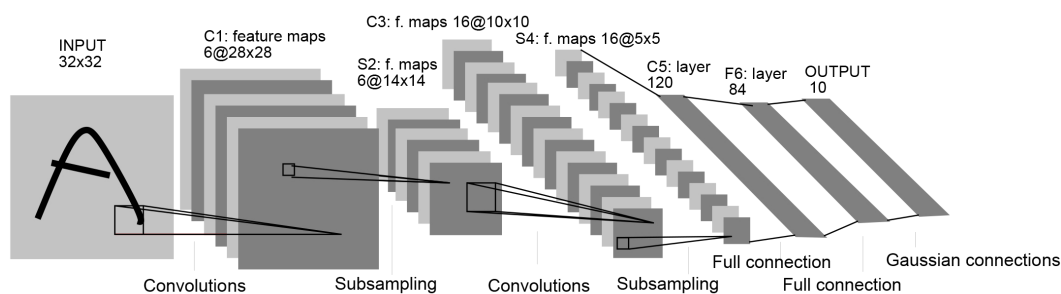
Rysunek 4. Przepływu sygnałów wstecz i proces uczenia neuronu

Uczenie pojedynczego neurosumatora polega na korekcie każdej z wag o $\Delta w_0 = x_0 * dFe * \mu$, gdzie μ jest współczynnikiem uczenia. W neurosumatorze obliczana jest także wielkość błędu dla warstwy wcześniejszej. I tak dla n – tego neurosumatora wynosi $E_n = [w_0 * dFe, w_1 * dFe, \dots, w_m * dFe]$. Błąd warstwy poprzedzającej jest sumą błędów ze wszystkich neurosumatorów warstwy następnej. Wartości błędów są przekazywane od warstwy wyjściowej, przez kolejne poprzedzające aż do warstwy wejściowej.

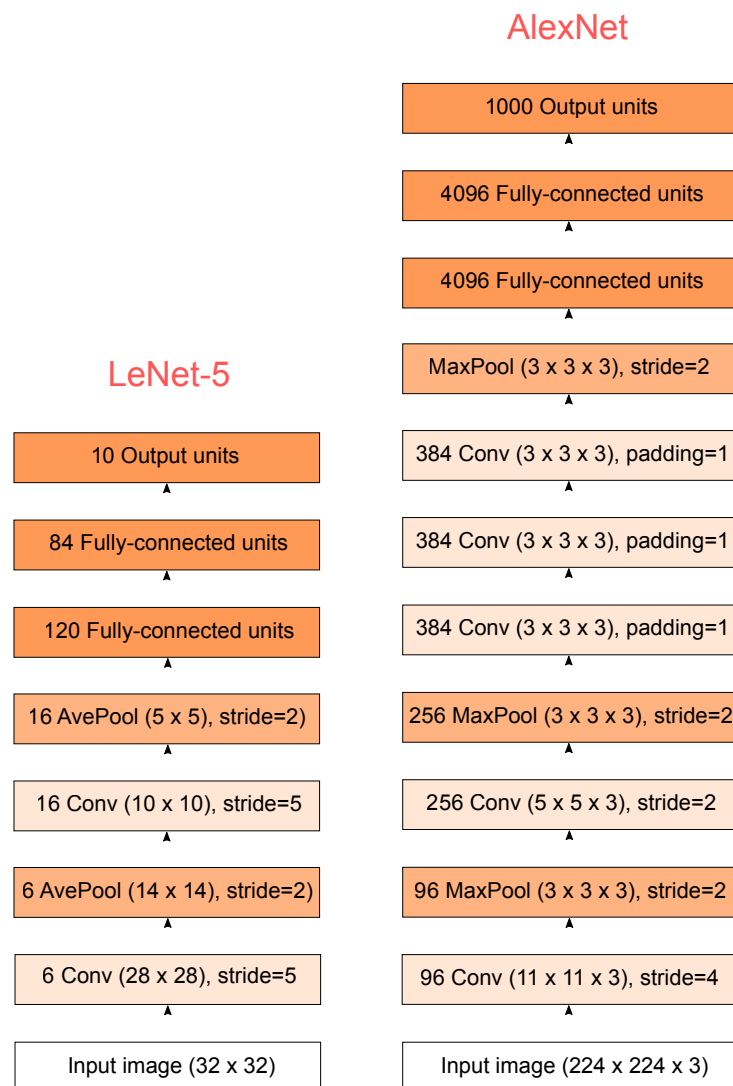
Rozdział 3

Sieć splotowa CNN

Cytat z [ossowski2023]. Sieci CNN powstały jako narzędzie analizy i rozpoznawania obrazów wzorowanym na sposobie działania naszych zmysłów. Wyeliminowały kłopotliwy proces manualnego opisu cech charakterystycznych obrazów. W tym rozwiązaniu sieć sama odpowiada za generację cech charakterystycznych dla klas. Przetwarzania obrazu przez zestaw filtrów i warstw generuje obrazy o coraz mniejszych wymiarach lecz o zwiększającej się liczbie kanałów, reprezentujące cechy charakterystyczne dla przetwarzanego zbioru. Próbkę danych w kolejnych etapach jest reprezentowana przez tensory (struktury 3-wymiarowe, których szerokość i wysokość odpowiada wymiarom obrazu, a głębokość jest równa liczbie kanałów obrazu). Wyjście z ostatniej warstwy sieci w procesie wypłaszczania jest przetwarzane są na postać wektorową (1-wymiarowa tablica liczb), a następnie przekazywane na wejście układu klasyfikującego - sieci MLP z wyjściem Softmax. [YannCnn]



Rysunek 5. Architektura LaNet-5 [YannCnn]



Notation:

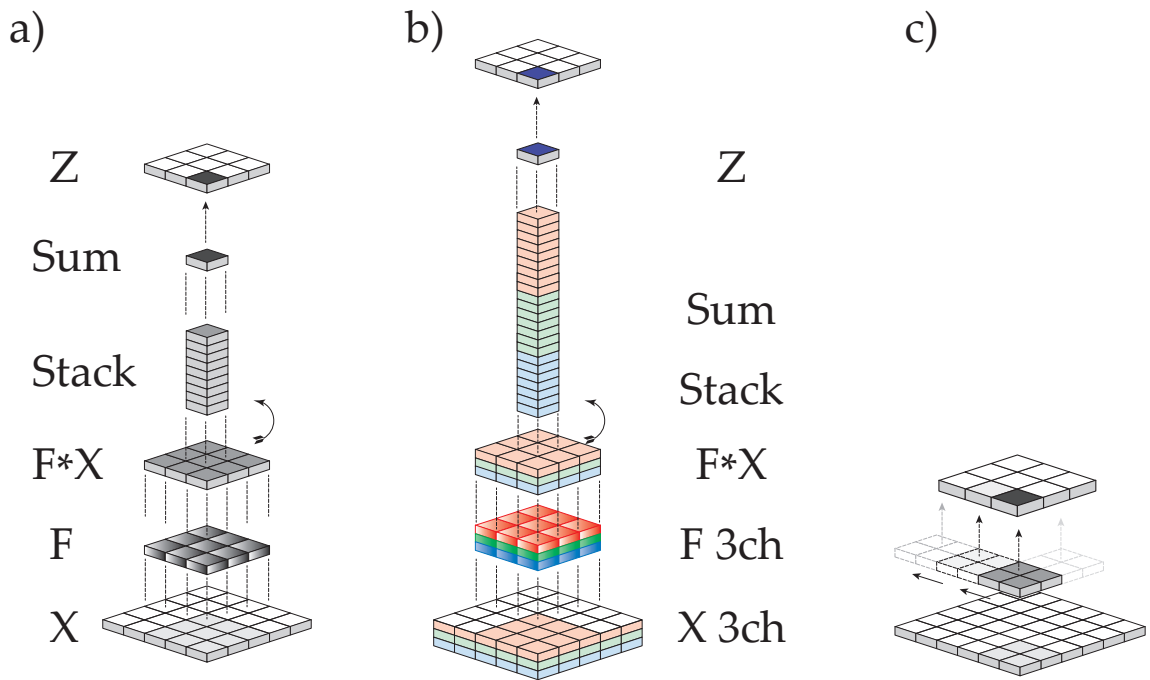
<number of planes> **type of layer** <width x height x RGB>, **stride/padding** <size>

Rysunek 6. Porównanie struktury LaNet i AlexNet [link](#)

3.1 Operacja splotu

Jeśli wartości pikseli obrazu wejściowego X oznaczmy $X_{(i,j)}$ a wartości filtra F oznaczmy jako $F_{(m,n)}$ obraz wyjściowy Y , $Y_{(o,p)}$ wówczas każdy piksel obrazu wyjściowego obliczymy:

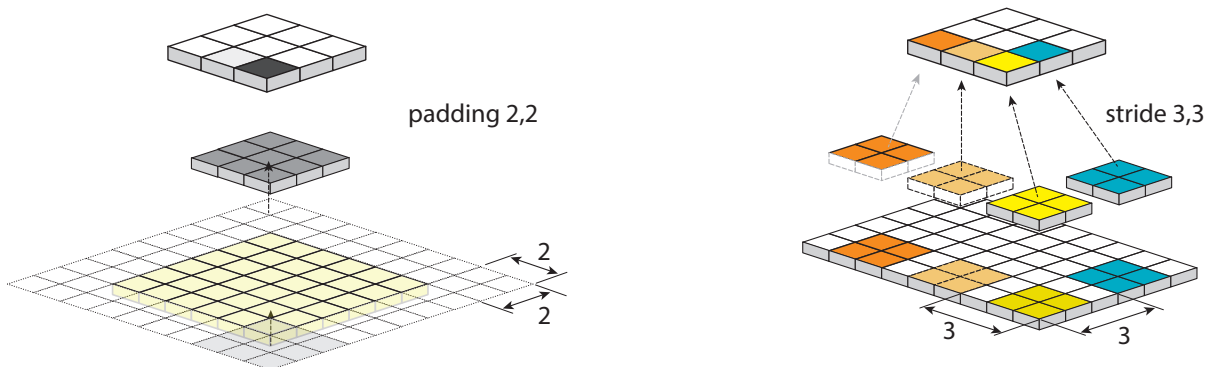
$$Y_{(O,P)} = \sum_{(n=1)}^{(N)} \sum_{(m=1)}^{(M)} F_{(m,n)} * X_{(m+O,n+P)} \quad (3)$$



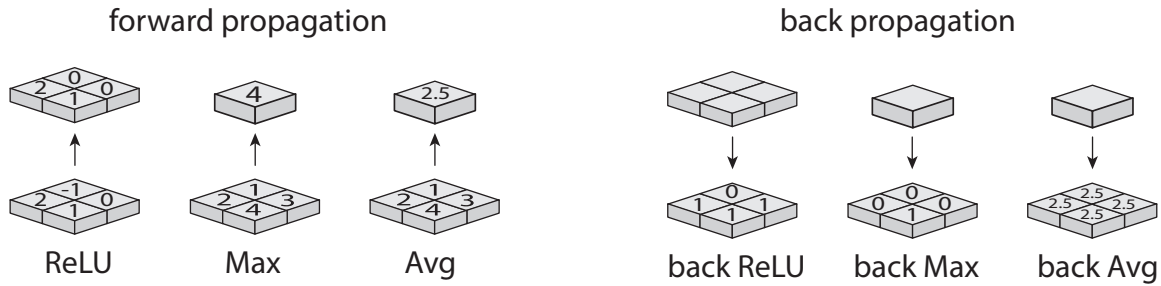
Rysunek 7. a) konwolucja 1 kanał 1 filtr, b) konwolucja 3 kanały 1 filtr, c) przesuwanie filtra nad obrazem

Padding oznacza wysunięcie filtra poza obraz przy obliczaniu konwolucji, stride oznacza krok przesunięcia filtra nad obrazem.

Wymiar wyjściowy obrazu $\dim Out = (\dim In + 1 - \dim F + 2 * padding) / stride$



3.2 ReLU i warstwy redukujące wymiar



Rysunek 8. wynik działania warstw ReLU, Avg, Max przy przepływie wprost i wstecz.

Po operacji splotu można zastosować warstwę ReLU. Funkcja aktywacji $\text{ReLU} = \max\{x, 0\}$. Zerowane są wartości ujemne, funkcja nie ma ciągłej pochodnej, przy obliczeniach przyjmowano że pochodna wynosi 1 gdy wartość wyjścia z była większa od 0, lub 0 jeśli wartość z była mniejsza od zera.

Można także zastosować warstwy redukującą rozmiar. Warstwa AVG - na wyjściu zwraca średnią wartości sąsiednich pól, pochodna przy przepływie wstecz dla każdego pola wynosi $1/n^2$ gdzie n jest wielkością filtra. Warstwa MAX przypisująca wartość maksymalną z sąsiednich pól. Pochodna przy propagacji wstecz wynosi 1 dla wartości maksymalnej, i 0 dla pozostałych.

3.3 Przetwarzanie w przód

Obraz wejściowy w skali szarości, lub obraz kolorowy rozbity na 3 kanały RGB zostaje przetworzony w operacji splotu przez grupy warstw filtrów i warstwy redukujące. Warstwa konwolucyjna może, a warstwa redukująca zmniejsza rozmiar obrazu, w efekcie otrzymujemy coraz mniejsze obrazy wyjściowe.

Wyjściem każdego filtra jest jeden kanał obrazu. W miarę zmniejszania obrazów stosowana jest większa liczba filtrów, co zwiększa liczbę kanałów. Przedostatnia warstwa spłaszcza wszystkie kanały obrazu do postaci pojedynczego jednowymiarowego wektora, warstwa ostatnia Full Connected z wyjściem Softmax dokonuje klasyfikacji obrazu.

3.4 Modyfikacja filtra w warstwie splotowej

Wektor wielkości błędu zostaje przekazywany z warstw MLP przez warstwy redukujące do warstw splotowych, w których następuje modyfikacja Filtra zgodnie z formułą.

$$\frac{\partial L}{\partial F} = \text{Conv}(\text{Input}.X, \text{Loss.Gradient} \frac{\partial L}{\partial O}); F = F - \mu * \frac{\partial L}{\partial F} \quad (4)$$

$$\frac{\partial L}{\partial \text{Bias}} = \text{Sum}(\text{Loss.Gradient} \frac{\partial L}{\partial O}); \text{Bias} = \text{Bias} - \mu * \frac{\partial L}{\partial \text{Bias}} \quad (5)$$

3.5 Propagacja błędu przez warstwę splotową

$$\frac{\partial L}{\partial X} = FullConv(180Rotated.filter.F, Loss.Gradient \frac{\partial L}{\partial O}) \quad (6)$$

3.6 Yolo 11 / 12

*** DO UZUPEŁNIENIA *** Opis sieci Yolo oraz ich realizacji w Python i C++ *** YoLo schemat [MMuraszkiewiczRobertNowak]

<https://com-cog-book.github.io/com-cog-book/features/cov-net.html>

Rozdział 4

Przetwarzanie równoległe

4.1 Maszyna Turinga

Zasada działania maszyny Turinga sprowadza się do cyklicznego wykonywania sekwencji operacji: odczytu wartości z pamięci, wykonaniu działania, zapisaniu wyniku w pamięci. Modelowanie tworzenia przetwarzającego informacje całą objętością w sposób jednoczesny używając maszyny sekwencyjnej w miarę powiększania modelu spowoduje lawinowy wzrost czasu przetwarzania.

4.2 Szybsze obliczenia grafiki 3D

W zastosowaniach inżynierskich (oraz rozrywkowych) maszyn liczących bardzo szybko pojawiała się potrzeba realizacji szybkich obliczeń translacji punktów w przestrzeni 3D. Operacje te sprowadzają się do mnożenia wektora współrzędnych znormalizowanego punktu przez macierz translacji:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}, O_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

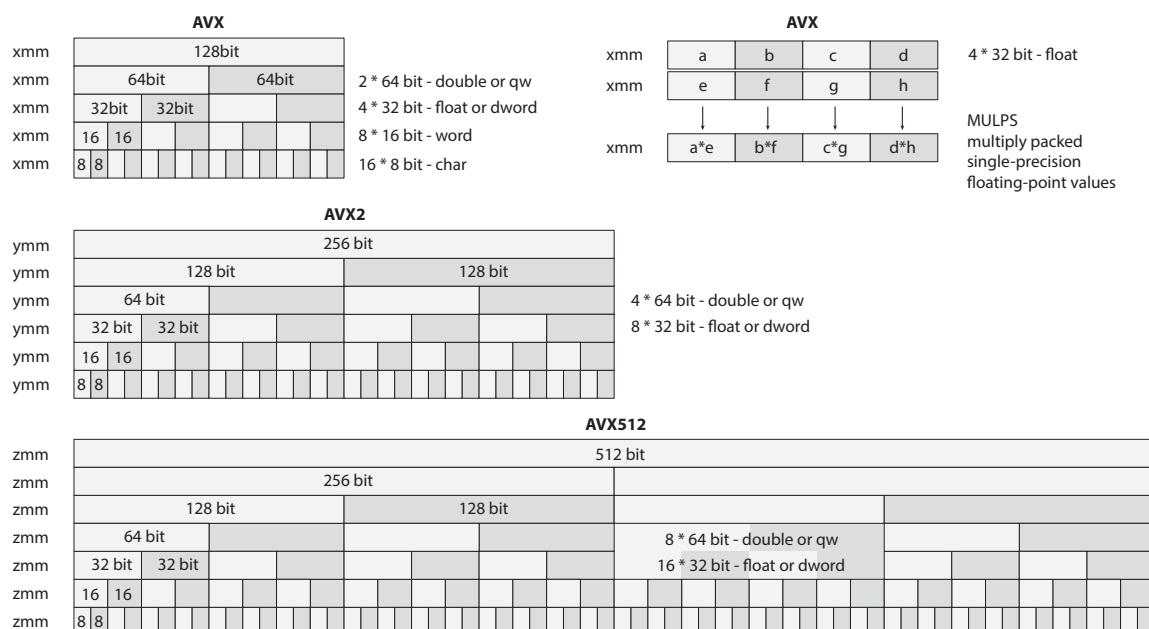
a po rozpisaniu mamy:

$$\begin{aligned} X &= x * m[0][0] + y * m[1][0] + z * m[2][0] + w * m[3][0] \\ Y &= x * m[0][1] + y * m[1][1] + z * m[2][1] + w * m[3][1] \\ Z &= x * m[0][2] + y * m[1][2] + z * m[2][2] + w * m[3][2] \\ W &= x * m[0][3] + y * m[1][3] + z * m[2][3] + w * m[3][3] \end{aligned}$$

Pierwszą odpowiedzią na to zapotrzebowanie był wprowadzony na rynek w 1980 roku koprocessor arytmetyczny Intel 8087, jego zadaniem było przyspieszenie operacji arytmetycznych zwłaszcza na liczbach zmiennoprzecinkowych w systemach opartych o procesor Intel 8088 i Intel 8086.

4.2.1 Funkcje procesora MMX, SIMD, SSE, AVX

SIMD (Single Instruction, Multiple Data) to funkcja procesora, która pozwala wykonywać jedną instrukcję na wielu 'strumieniach' danych. Może ona ewentualnie zwiększyć wydajność Twoich programów. SIMD to postać przetwarzania równoległego; jednak w niektórych przypadkach przetwarzanie różnych strumieni może odbywać się sekwencyjnie. Pierwszą implementacją był zbiór instrukcji MMX, zastąpiony przez strumieniowe rozszerzenia SIMD (ang. Streaming SIMD Extension, SSE). Później do SSE dodano zaawansowane rozszerzenia wektorowe (ang. Advanced Vector Extension, AVX).[asmx64] Procesor obsługujący SSE ma 16 dodatkowych rejestrów 128-bitowych (xmm0-xmm15). Procesor obsługujący AVX2 ma 256-bitowe rejestry ymm, AVX-256 i AVX-10 512-bitowe rejestry zmm.[asmx64]



Rysunek 9. Rejestry AVX, AVX2 i AVX512, operacja równoległego mnożenia AVX

Rozkaz MULPS wykonuje jednocześnie 4 mnożenia zestawu 8 liczb 32 bitowych. Instrukcje w AVX512 VMPADDWD, VMPADDUBSW - wykonują mnożenie wektorowe, następnie dodawanie sąsiednich elementów. $(a_i \cdot b_i + a_{i+1} \cdot b_{i+1})$

Funkcje te przyspieszają operacje iloczynu Hadamarda (.)[WKasprzak] czyli wielokanałowego mnożenia. W zależności od wersji dla zmiennych typu float uzyskano od 4 do 16 kanałów, czyli 16 operacji mnożenia w czasie jednego rozkazu.

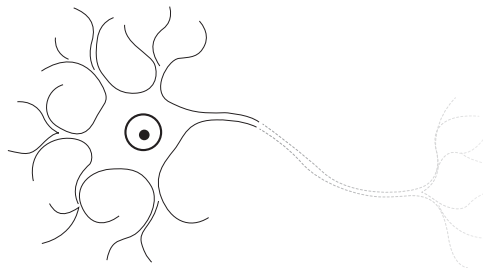
<https://www.youtube.com/watch?v=x9Scb5Mku1g>

4.3 Obliczenia na GPU

Wydajne obliczanie iloczynu Hadamarda (\cdot) wektorów o dużych rozmiarach mogą zapewnić procesory graficzne, w których znajdują się tysiące jednostek obliczeniowych, a dostęp do szybkiej pamięci jest bezpośredni. Czas wykonania kilku tysięcy mnożeń na kartach graficznych zajmuje tyle samo czasu co wykonanie jednego mnożenia. Użycie kart graficznych przy modelowaniu głębokich sieci jest koniecznością, ponieważ skraca czas trenowania sieci o rzędy wielkości w stosunku do obliczeń na CPU. W niniejszej pracy wykorzystano kartę NVIDIA GeForce RTX 4070 wyposażonej w 5888 rdzeni CUDA oraz 12282 MB pamięci na karcie graficznej.

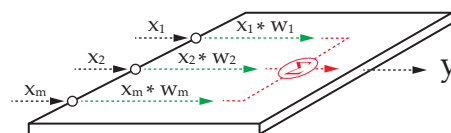
zapisy.ii.uni.wroc.pl/courses/kurs-obliczenia-rownolege-na-kartach-graficznych-cuda-q2-201920-zimowy
www.cs.put.poznan.pl/rwalkowiak/pliki/2019/pr/programowanie_PKG_CUDA_2019.pdf

4.4 Dodawanie równoległe



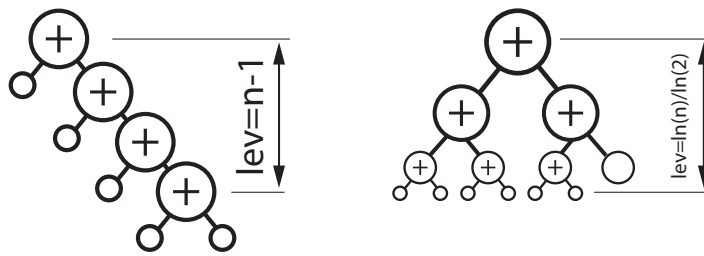
Rysunek 10. Sumator wielokanałowy

Operacje mnożenia - na rysunku oznaczone kolorem zielonym - obliczono równoległe niezależnie od siebie i w tym samym czasie. Natomiast wielokrotne dodawania - na rysunku oznaczone kolorem czerwonym - nie są operacjami niezależnymi.



Rysunek 11. Operacje niezależne i zależne sum

Sumowanie $a+b+c+d$ realizowane jest analogiczne do zdegenerowanego drzewa binarnego czyli $((a+b)+c)+d$. Przy dużej ilości składników możemy spróbować zastosować zwykłe drzewo binarne nie zdegenerowane, a operacja dodawania może stać się częściowo równoległa $((a+b) + (c+d))$. Maszyny cyfrowe obecnie nie są wyposażone w sumatory wielokanałowe pozwalające dodawać więcej niż 2 liczby jednocześnie. Można zasymulować quasi-równoległe dodawanie stosując nawiasy.



Rysunek 12. drzewo zdegenerowane i niezdegenerowane

$b = a1 + a2 + a3 + a4 + a5 + a6 + a7 + \dots + a1024$; czas wykonania: **0.247875 [sek.]**

$b = ((..((a1 + a2) + (a3 + a4)) + ((a5 + a6) \dots$ czas wykonania: **0.09568 [sek.]**

Rozdział 5

Przedstawienie języków

5.1 Wstęp

5.1.1 Popularność języka

Duża popularność rozwiązania zapewnia łatwy dostęp do dużej społeczności osób pracujących z językiem i nad rozwojem języka. Dla dynamicznie rozwijających się języków powstaje wiele bibliotek, środowisk IDE, narzędzi a także literatury i przykładów zastosowania w rozwiązywaniu konkretnych zadań. Przy wyborze języka do zastosowania w projektowanych rozwiązaniach warto wybierać języki popularne i nowoczesne aby uniknąć trudności ze znalezieniem specjalistów do pracy nad projektem.

5.1.2 Dostępność bibliotek

Dla otwartych języków programowania powstaje wiele narzędzi i bibliotek, z których duża część jest udostępniana społeczności na różne sposoby. Dla rozwiązań komercyjnych bibliotek jest mniej, natomiast ich jakość najczęściej jest bardzo wysoka, dokumentacja dopracowana a pomoc techniczna łatwo dostępna.

5.1.3 Wygoda instalacji

Instalacja języka wraz ze środowiskiem może różnić się znacznie zależnie od systemu operacyjnego. Języki niższych poziomów integrujące się bardziej z systemem operacyjnym mogą być trudniejsze w instalacji i konfiguracji do pracy z np. kartami graficznymi.

5.1.4 Wykorzystanie GPU

W modelowaniu sieci głębokich wykorzystanie potencjału kart graficznych ma znaczenie kluczowe.

5.1.5 Koszt licencji

Większość języków i środowisk jest dostępna na licencji otwartej. Wykorzystanie niektórych języków, bibliotek czy środowisk IDE wiąże się z koniecznością zakupu licencji. Wykorzystanie w celach komercyjnych może wymagać zakupu innej droższej licencji niż wykorzystanie w celach naukowych.

5.1.6 Łatwość użytkowania

W większości języków programowania ogólnego przeznaczenia stosuje się bardzo podobne konstrukcje i pracuje się na podobnych rodzajach danych. Języki programowania wysokiego poziomu bazują na bytach abstrakcyjnych, tworzenie kodu jest łatwe i nie wymagają od programisty / użytkownika znajomości dokładnej budowy komputera, znajomości sposobu działania pamięci czy rodzaju procesora na którym będzie uruchamiany program. Nauka programowania w językach wysokiego poziomu jest dużo szybsza niż w językach niższego poziomu jak np. C++ gdzie występuje ścisła kontrola typów, jednocześnie występują wskaźniki, referencje, ramki stosu. Nie wspominając o języku asemblera gdzie niezbędna jest wiedza nie tylko z zakresu np. reprezentacji liczb w pamięci, przeliczania liczb szesnastkowych na dziesiętne, ale i pracy na rejestrach, stosie, wywołaniach przerwań sprzętowych czy funkcji systemowych.

5.2 Matlab

Dodatek Parallel computing w Matlab umożliwia wykonywanie obliczeń równoległe, w osobnych wątkach, procesach a także z wykorzystaniem kart graficznych (obecnie tylko NVidia). Dodatki Optimization Toolbox i Optimization Computing Toolbox optymalizują tworzony kod zwiększając jego wydajność. Dodatek Deep Learning Toolbox dostarcza gotowych metod do obliczeń sieci neuronowych.

5.3 Python

Metody do obliczeń sieci neuronowych w języka Python dostarczone są m.in. w bibliotekach scikit-learn i TensorFlow2 a dostarczają one gotowych metod przydatnych w obliczeniach sieci neuronowych. Wykorzystują one możliwości równoległego przetwarzania, które oferuje sprzęt na którym uruchamiany jest kod.

Rozdział 6

Porównanie języków

Rozdział 7

Jednowymiarowa regresja liniowa

7.1 Wprowadzenie

Przykłady obliczeń Python i Matlab zostały zaczerpnięte z [ossowski2023]. **Pełen kod dostępny na github:** <https://github.com/piotrHeinzelman/inz/tree/main/MixedProj/01.polyfit> w przypadku Matlab i Python korzystam z dostępnych funkcji, w przypadku Java obliczam wg. wzoru `??`. Obliczenia różnymi metodami dają zbliżone wyniki, więc zakładam że moje implementacje są poprawne.

W pracy tam gdzie możliwe staram się wykorzystać dostarczone funkcje liczące. I tak dla Matlab i dla Python wykorzystałem zaimplementowaną funkcję "polyfit". W kodzie Java musiałem sam zaimplementować funkcję liczącą dla porównania wydajności. W programach nie porównuję czasu ładowania i przygotowania danych ponieważ chcę wykonywać obliczenia dla tych samych wartości czytanych z tych samych plików, natomiast nie jest moim celem optymalizacja wczytywania danych z pliku.

7.2 Prowadzenie badania

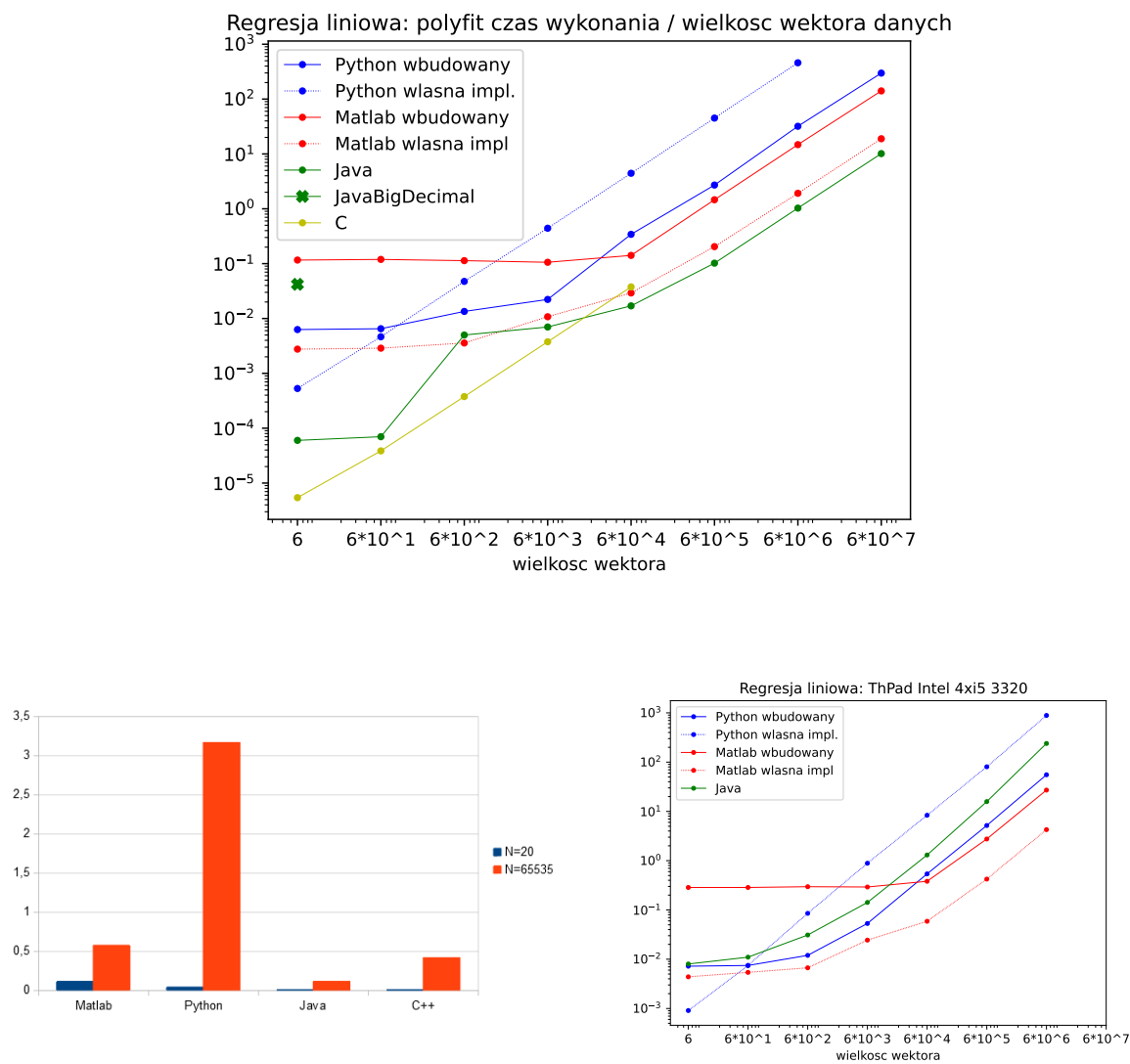
Programy w kolejnych językach uruchamiam poniższym kodem:

```
1 ./matRun >> output.txt
2 ./pyRun >> output.txt
3 ./javaRun >> output.txt
4
5 //matRun
6 echo "—— Matlab app start: ——"
7 /usr/local/MATLAB/R2024a/bin/matlab -nodisplay -nosplash -nodesktop -batch 'run matlab.m'
8
9 //pyRun
10 echo "—— Python app start: —— "
11 python main.py
12
13 // javaRun
14 javac Main.java
15 echo "—— Java app start: ——"
16 java Main
```

7.3 Kod realizujący obliczenia

```
1 //          —— MATLAB ——
2
3 a = polyfit(x,y,1);
4
5 //          —— Python ——
6
7 a = np.polyfit(x,y,1)
8
9 //          —— Java ——
10
11 for (int i = 0; i < x.length; i++) {
12     xsr += x[i];
13     ysr += y[i];
14 }
15 xsr = xsr / x.length;
16 ysr = ysr / y.length;
17
18 for (int i = 0; i < x.length; i++) {
19     sumTop += ((x[i] - xsr) * (y[i] - ysr));
20     sumBottom += ((x[i] - xsr) * (x[i] - xsr));
21 }
22 w1 = sumTop / sumBottom;
23 w0 = ysr - w1 * xsr;
24
25 //          —— C++ ——
26
27 for ( int i=0; i<len; i++) {
28     xsr += X[i];
29     ysr += Y[i];
30 }
31 xsr=xsr / len; ysr=ysr / len;
32 double sumTop=0.0;
33 double sumBottom=0.0;
34
35 for ( int i=0;i<len;i++) { // xtmp = X[i]-srx ! ;
36     sumTop += ((X[i]-xsr)*(Y[i]-ysr));
37     sumBottom += ((X[i]-xsr)*(X[i]-xsr));
38 }
39 w1 = sumTop / sumBottom;
40 w0 = ysr -(w1 * xsr) ;
41
42 .
```

7.4 Uzyskane wyniki



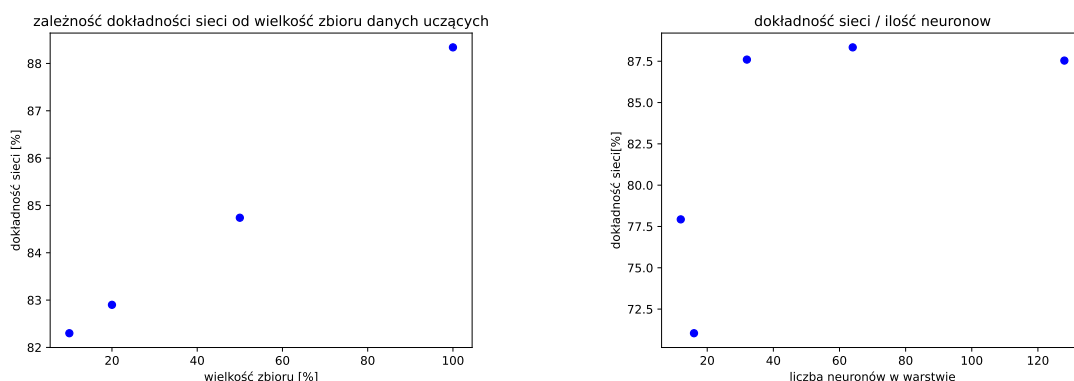
Rysunek 13. Porównanie czasów obliczania regresji liniowej

Rozdział 8

Klasyfikowanie pisma odręcznego MLP

8.1 Wstęp

Porównuję dokładność sieci rodzaju MLP w zależności od ilości danych - podczas uczenia sieci wycinkiem danych uczących, dokładność sieci wzrasta do granicznej 88,34% przy wykorzystaniu pełnego zestawu danych uczących. Zmiana ilości neuronów w dwu warstwach również wpływa na dokładność sieci i jest największa przy 64 neuronach. Zwiększenie lub zmniejszenie liczby neuronów zmniejsza dokładność sieci. Długość procesu uczenia wynosiła 5000 epok. Szacowanie przeprowadzałem w języku Matlab. Metodę uczenia wybrałem metodę największego spadku, ponieważ przy wyborze metody LM miałem za mało pamięci dla takiej konfiguracji sieci. (wymagane 21Gb)

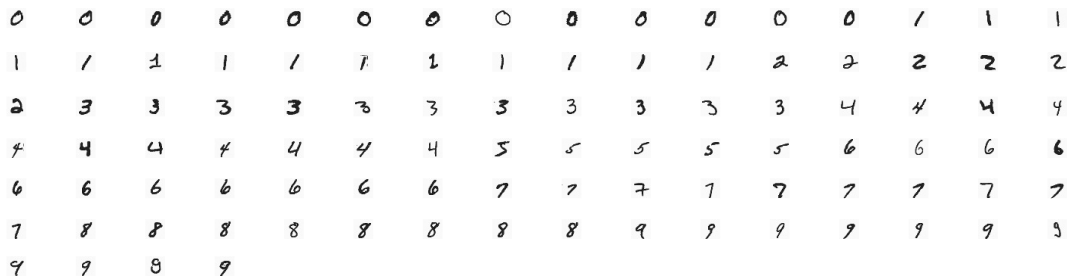


Rysunek 14. Szacowanie parametrów sieci

Porównanie czasów wykonania uczenia 64 neuronów w 2 warstwach i 5000 epok dla różnych języków. Dla Matlab wyniki uzyskane w systemie operacyjnym Windows 10 oraz Arch Linux w trybie tekstowym, dodatkowo wyniki przy obliczeniach domyślnych proponowanych przez program, w porównaniu z wymuszeniem obliczeń tylko na GPU. Dość zaskakująca jest różnica w szybkości obliczeń na GPU w zależności od platformy. Odpowiednio w systemie Windows 10 obliczenia trwają 419-425 sek., Linux Arch wykona te obliczenia w czasie 205-211 sek.

8.2 Dane treningowe

Dane treningowe i testowe - zestaw pisanych ręcznie cyfr - pochodzą z zasobów MNIST (yann.lecun.com). w kodzie Java dodałem zestaw metod umożliwiającą m.in. podejrzenie wektorów treningowych jako obrazy, zapisanie wag warstwy jako obraz, i inne.



Rysunek 15. podgląd próbki wektorów uczących

poniżej kod ładujący dane, oraz metody narzędziowe:

```

1 public void prepareData( int percent ){
2     trainYfile = loadBin( "../data/" + "t10k-images-idx3-ubyte", 8, percent*600 ); //offset=8, size=percent*600 // OK
3     testYfile = loadBin( "../data/" + "t10k-labels-idx1-ubyte", 8, percent*100 ); // offset=8, size=percent*100 // OK
4
5     trainY = new float[percent*600][];
6     testY = new float[percent*100][];
7     //train Y
8     for (int i=0;i<percent*600;i++){
9         trainY[i] = new float[] { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
10        trainY[i][ trainYfile[i] ]=1.0f;
11    }
12
13
14    byte[] trainXfile = loadBin(path + trainXname, 16, percent * 784 * 600); // offset=16 size=percent*784*600
15    trainX=new float[percent*600][784];
16    for (int i=0;i<percent*100;i++) {
17        int off=i*784;
18        for (int j=0;j<784;j++){
19            trainX[i][j]=Byte.toUnsignedInt( trainXfile[ off+j] )/16;
20        }
21    } catch (IOException e) { throw new RuntimeException(e); }
22 }
23
24
25 public static byte[] loadBin( String filename, int offset, int len ) throws IOException {
26     byte[] bytesBuf = new byte[ len ];
27     File f = new File( filename );
28     FileInputStream fis = new FileInputStream( filename );
29     fis.skip(offset);
30     fis.read( bytesBuf, 0, len );
31     return bytesBuf;
32 }
33
34 public static float[] vectorSubstSubZ( float[] s, float[] z ){
35     float[] out = new float[ z.length];
36     for ( int i=0;i<z.length; i++){
37         out[i] = ( s[i] - z[i] );
38     }
39     return out;
40 }
41
42 public static float meanSquareError( float[] s, float[] z ){
43     float out = 0.0f;
44     for ( int i=0;i<z.length; i++){
45         float delta = s[i] - z[i];
46         out+=delta*delta;
47     }
48     return out;
49 }

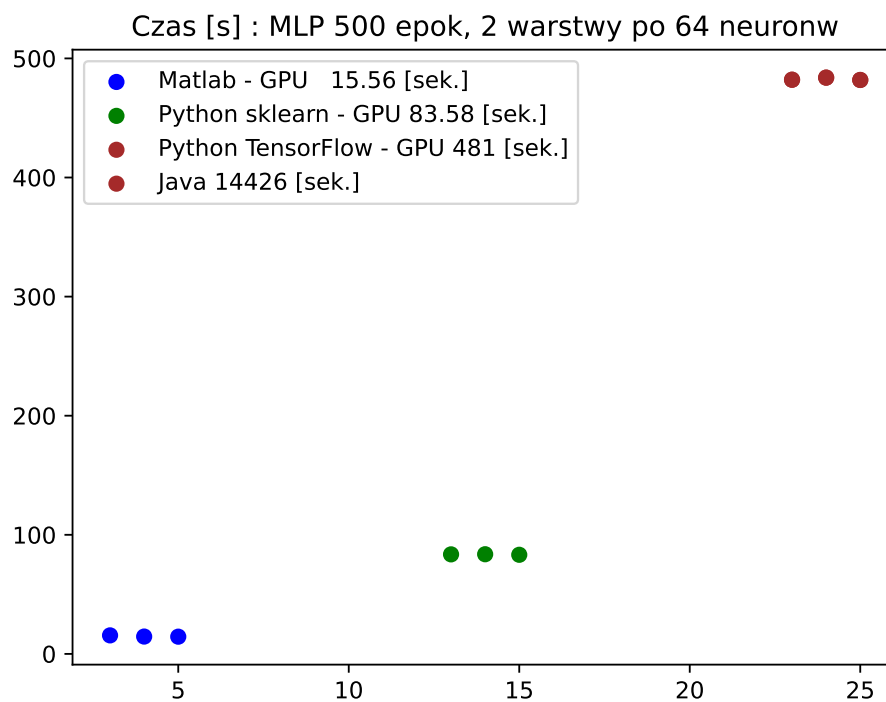
```

```

50 |
51 | public static BufferedImage arrayOfFloatToImage( float[] data , int xScale ){
52 |     int width = data.length/xScale;
53 |     int height = 510;
54 |     float min = data[0];
55 |     float max = data[0];
56 |     for ( int i=1;i< data.length;i++){
57 |         if ( data[i]<min ) { min=data[i]; }
58 |         if ( data[i]>max ) { max=data[i]; }
59 |     }
60 |     float delta=( max-min )/(height-10);
61 |     BufferedImage image = new BufferedImage( width , height , TYPE_INT_RGB );
62 |
63 |     int pointColor = (255*255*240)+(255*244)+244;
64 |     for ( int i=0;i<width;i++){
65 |         int val=(int) (( data[xScale*i]-min )/delta ) ;
66 |         image.setRGB( i, 5+val , pointColor );
67 |     }
68 |     return image;
69 | }
70 |
71 | public static void saveImg( BufferedImage image, String nameSuffix ){
72 |     File file = new File("image"+nameSuffix+".png");
73 |     try {
74 |         ImageIO.write(image , "png", file );
75 |     } catch (IOException e) {
76 |         throw new RuntimeException(e);
77 |     }
78 | }
79 | }

```

8.3 Uzyskane wyniki



8.4 Kod realizujący obliczenia w Matlab

```
1
2 net = feedforwardnet([ 64, 64 ],'traingd');
3 net.trainParam.epochs = 5000;
4 net.trainParam.goal    = 0.03;
5 net.input.processFcns = {'mapminmax'};
6
7 gxtrain = gpuArray( xtrain );
8 gytrain = gpuArray( ytrain );
9
10 ST = datetime('now');
11
12 gpu=true;
13 if ( gpu )
14     %GPU
15     net = configure(net,xtrain,ytrain);
16     net = train(net, gxtrain, gytrain,'useParallel','yes','useGPU','
yes');
17 else
18     %No GPU
19     net = train( net, xtrain, ytrain );
20 end
21
22 ED = datetime('now');
23
24 z = net( xtest );
25 flatZ = aryOfVectorToAryOfInt( z );
26 flatZtest = aryOfVectorToAryOfInt( ytest );
27 accuracy = accuracyCheck(flatZ, flatZtest);
28
29 D = duration( ED-ST );
30
31 fprintf('# MLP: 2x64Neu * 5000 cycles:\n' );
32 fprintf( '# accuracy: a:%f\n\n' , accuracy );
33 fprintf( 'm[]=%f\n' , seconds(D) );
```

8.5 Własna implementacja w Java

Do napisania programu użyłem własnego, opisanego we wcześniejszych rozdziałach modelu obiektowego sieci neuronowych. Sieć ta pracuje dużo wolniej niż rozwiązanie w matlab, uczenie sieci zajmuje aż 14400 sek. Najważniejszym celem napisania tego programu było sprawdzenie czy sieć oparta na modelu obiektowym działa, i czy osiągnie podobny poziom dokładności jak gotowe rozwiązanie z Matlab. Okazuje się, że podobnie jak sieć w Matlab, ta napisana w Java osiąga dokładność około 87%; Kod celowo nie jest optymalizowany pod kątem wydajności - chodziło mi o łatwość czytania i zrozumienia modelu, a także potwierdzenie że sieć oparta na tym modelu uczy się podobnie jak gotowe rozwiązanie z Matlab. W kodzie umieściłem metody dzięki którym mogłem sprawdzić czy dane wczytują się prawidłowo. Poniżej zamieszczam kod rozwiązania:

```

1 // main.java
2 private float[][] testX; private float[][] testY; private float[][] trainX; private float[][] trainY;
3
4 private Layer layer1; private Layer layer2; private Layer layer3;
5
6 private Tools tools = new Tools();
7 int numOfEpoch=500;
8 float[] CSBin_data=new float[numOfEpoch];
9
10 public void prepare() {
11     tools.prepareData( 50 );
12     testX = tools.getTestX();
13     testY = tools.getTestY();
14     trainX = tools.getTrainX();
15     trainY = tools.getTrainY();
16
17     layer1=new Layer( LType.sigmod , 64 ,784 ); layer1.setName("Layer1"); // n neurons
18     layer2=new Layer( LType.sigmod , 64 ,64 ); layer2.setName("Layer2"); // n neurons
19     layer3=new Layer( LType.softmaxMultiClass , 10 ,64 ); layer3.setName("Layer3"); // n neurons
20     layer1.rnd(); layer2.rnd(); layer3.rnd();
21 }
22
23 public void run() {
24
25     float Loss = 0.0f;
26     for (int epoch = 0; epoch < numOfEpoch; epoch++) {
27         for ( int index = 0; index < trainX.length; index++ ) {
28             layer1.setX( trainX[ index ] );
29             layer1.nForward();
30             layer2.setX( layer1.getZ() );
31             layer2.nForward();
32             layer3.setX( layer2.getZ() );
33             layer3.nForward();
34
35             float[] S_Z = tools.vectorSubstSubZ( trainY[ ind_ex ], layer3.getZ() );
36             layer3.nBackward( S_Z );
37             layer2.nBackward( layer3.getEout() );
38             layer1.nBackward( layer2.getEout() );
39         }
40     }
41     // check accuracy
42     int len = testX.length;
43     int accuracy = 0;
44     for (int i = 0; i < len; i++) {
45         layer1.setX( testX[i] );
46         layer1.nForward();
47         layer2.setX( layer1.getZ() );
48         layer2.nForward();
49         layer3.setX( layer2.getZ() );
50         layer3.nForward();
51
52         int netClassId = tools.getIndexMaxFloat( layer3.getZ() );
53         int fileClassId = tools.getIndexMaxFloat( testY[i] );
54         if ( fileClassId == netClassId ) { accuracy++; }
55     }
56     System.out.println(100.0f * accuracy / len + "%");
57 }

```

```

1 // neuron.java
2 public class Layer {
3     private String name;
4     private LType lType;
5     private Neuron[] neurons;
6     private float X[];
7     private float Y[];
8     private float Z[];
9     private float dFofZ[];
10    private float Eout[]; // S-Z for last
11
12    public Layer( LType lType, int n, int m ) { // n - number of neurons & output size Y[n], Z[n]
13        this.lType=lType; // m - number of inputs = input size X[m]
14        this.neurons = new Neuron[n];
15        for (int i=0; i<n; i++){
16            this.neurons[i]=new Neuron(m, this);
17        }
18        X = new float[m];
19        Y = new float[n];
20        Z = new float[n];
21        dFofZ = new float[n];
22        Eout = new float[m];
23    }
24
25    public void setWmn( int n, int m, float wji ){ neurons[n].setWm( m, wji ); }
26
27    public void rnd(){
28        Random random=new Random();
29        for ( Neuron neu : neurons ) {
30            for ( int m=0; m<X.length; m++ ) {
31                neu.setWm( m , (float)( -1.0f+2.0f*random.nextFloat() ) );
32            }
33        }
34    }
35
36    public float[] nForward() {
37        switch (lType) {
38            case sigmod:
39                case sigmod_CrossEntropy_Binary:{
40                    for (int n = 0; n < neurons.length; n++) {
41                        Y[n] = neurons[n].Forward(X);
42                        Z[n] = F(Y[n]);
43                        dFofZ[n] = dF(Z[n]);
44                    }
45                    for (int m=0;m<Eout.length;m++){ Eout[m]=0; }
46                    return Z;
47                }
48                case softmaxMultiClass: {
49                    int len = neurons.length;
50                    float sum = 0.0f;
51                    float max = Y[0] = neurons[0].Forward(X);
52                    for (int i=1;i<len;i++){
53                        dFofZ[i] = 1;
54                        Y[i]=neurons[i].Forward(X);
55                        if (Y[i]>max) { max=Y[i]; }
56                    }
57                    for (int i = 0; i < len; i++) {
58                        Y[i] = (float) Math.exp( Y[i]-max );
59                        sum += Y[i];
60                    }
61                    for (int i = 0; i < len; i++) {
62                        Z[i] = Y[i] / sum;
63                    }
64                    return Z;
65                }
66            default: { return Z; }
67        }
68    }
69
70    public void nBackward( float[] Ein ){ // S-Z or Ein
71        for ( int i=0;i<neurons.length;i++){ Eout[i]=0.0f; }
72        for ( int n=0; n< neurons.length; n++){
73            neurons[n].Backward( Ein[n] * dFofZ[n] );
74        }
75    }
76
77
78
79

```

```

80 private float F ( float y ){
81     float z;
82     switch (this.lType) {
83         case sigmod:
84             case sigmod_CrossEntropy_Binary:
85                 { z = (float) (1.0f/(1.0f + Math.exp( -y ))); break; }
86             case linear:
87                 default: { z=y; break; }
88     }
89     return z;
90 }
91
92 private float dF ( float z ){
93     float df;
94     switch (lType) {
95         case sigmod:
96             { df = z*(1-z); break; }
97         case linear:
98             case sigmod_CrossEntropy_Binary:
99                 default: { df=1; break; }
100     }
101     return df;
102 }
103
104 // getters / setters
105 public void setX( float[] _x ) {
106     for ( int m=0; m<X.length; m++){
107         this.X[m] = _x[m];
108     }
109 }
110 public float[] getZ() { return Z; }
111 public float[] getX() { return X; }
112 public float[] getEout() { return Eout; }
113 }

```

```

1 // neuron.java
2 public class Neuron {
3     private float bias=0;
4     private final float[] W;
5     private final Layer parent;
6     private final static float mu=0.001f;
7
8     public void setBias( float b ) { this.bias=b; }
9     public float getBias(){ return bias; }
10
11     public Neuron( int m, Layer parent ) {
12         this.parent=parent;
13         this.W = new float[m];
14     }
15
16     public void setWm( int m, float wji ){
17         W[m] = wji;
18     }
19
20     public float Forward( float[] X ) {
21         float res=bias;
22         for ( int m=0; m<W.length; m++) {
23             res+= X[m]*W[m];
24         }
25         return res;
26     }
27
28     public void Backward( float en_x_dFlznl ) {
29         float[] X = parent.getX();
30         for ( int m=0; m<W.length; m++) {
31             parent.getEout()[m] += ( W[m] * en_x_dFlznl );
32             W[m] += mu * en_x_dFlznl * X[m];
33         }
34     }
35 }

```

8.6 Python -sklearn

```
1 import numpy as np
2 import time
3 import struct
4 import sklearn.neural_network as snn
5
6 def readFileX ( fileName , offset, percent, multi ):
7     file=open( fileName, 'rb' )
8     file.read( offset )
9     data=np.fromfile( fileName, np.uint8, percent*100*784*multi, '',
10         offset )
11     data=data.reshape(percent*100*multi, 784)
12     data=1-(data/128)
13     file.close()
14     return data
15
16 def readFileY ( fileName , offset, percent, multi ):
17     file=open( fileName, 'rb' )
18     file.read( offset )
19     len=percent*100*multi
20     data=np.fromfile( fileName, np.uint8, len, '', offset )
21     out=[]
22     for i in range ( len ):
23         tmp=[0,0,0,0,0,0,0,0,0,0,0]
24         tmp[ data[i]] = 1
25         out.append( tmp )
26     file.close()
27     return out
28
29 percent=50
30 trainX = readFileX ( 'data/train-images-idx3-ubyte', 16, percent ,6 )
31 trainY = readFileY ( 'data/train-labels-idx1-ubyte', 8, percent, 6 )
32 testX = readFileX ( 'data/t10k-images-idx3-ubyte', 16, percent, 1 )
33 testY = readFileY ( 'data/t10k-labels-idx1-ubyte', 8, percent, 1 )
34
35 net = snn.MLPClassifier(hidden_layer_sizes=(64,64), random_state=1,
36     activation='logistic', solver='sgd' )
37 net.fit( trainX, trainY )
38
39 start=time.time()
40 for i in range(1):
41     for epo in range (500):
42         net.partial_fit( trainX, trainY )
43 print("Time: ", time.time()-start, score: ",net.score(testX, testY))
```


8.7 Python -tensorflow

```
1 import tensorflow as tf
2 import numpy as np
3 import time
4 from tensorflow.keras.backend import clear_session
5 import matplotlib.pyplot as plt
6
7 physical_devices = tf.config.list_physical_devices('GPU')
8 if physical_devices:
9     for gpu in physical_devices:
10         tf.config.experimental.set_memory_growth(gpu, True)
11
12 # params
13 epochs = 500
14 percent = 50
15 num_classes = 10
16
17 def readFileX ( fileName , offset , percent , multi ) :
18     file=open( fileName , 'rb' )
19     file.read( offset )
20     data=np.fromfile( fileName , np.uint8 , percent*100*784*multi , '' ,
21         offset )
22     data=data.reshape(percent*100*multi , 784)
23     data=(data/255)
24     file.close()
25     return data
26
27 def readFileY ( fileName , offset , percent , multi ) :
28     file=open( fileName , 'rb' )
29     file.read( offset )
30     len=percent*100*multi
31     data=np.fromfile( fileName , np.uint8 , len , '' , offset )
32     file.close()
33     return data
34
35 trainX = readFileX ( 'data/train-images-idx3-ubyte' , 16 , percent , 6 )
36 trainY = readFileY ( 'data/train-labels-idx1-ubyte' , 8 , percent , 6 )
37 testX = readFileX ( 'data/t10k-images-idx3-ubyte' , 16 , percent , 1 )
38 testY = readFileY ( 'data/t10k-labels-idx1-ubyte' , 8 , percent , 1 )
39
40 trainX = trainX.astype("float32") # / 255
41 testX = testX.astype("float32") # / 255
42 trainX = trainX.reshape(6*percent*100 , 784).astype("float32") / 255
43 testX = testX.reshape(1*percent*100 , 784).astype("float32") / 255
```

```
44 model = tf.keras.models.Sequential([
45     tf.keras.layers.Input(shape=(784,)),
46     tf.keras.layers.Dense(64, activation='sigmoid'),
47     tf.keras.layers.Dense(64, activation='sigmoid'),
48     tf.keras.layers.Dropout(0.2),
49     tf.keras.layers.Dense(10, activation='softmax')
50 ])
51
52 model.compile(optimizer='adam',
53     loss='sparse_categorical_crossentropy',
54     metrics=['accuracy'])
55
56 start=time.time()
57
58 model.fit(trainX, trainY, epochs=epochs, verbose=0)
59
60 end=time.time()
61 d=end-start
62
63 clear_session()
64 print("# Python Tensorflow Time: " , d)
65
66 # model.evaluate(testX, testY)
```

8.8 C++ -on ? CUDA ?

Rozdział 9

Sieci głębokie

9.1 Wstęp

Cytat z [ossowski2023]. Ogromny postęp w dziedzinie sztucznej inteligencji dokonał się za pośrednictwem tzw. głębokiego uczenia. Głębokie uczenie dotyczy głównie wielowarstwowych sieci neuronowych, które pełnią jednocześnie funkcję generatora cech diagnostycznych dla analizowanego procesu oraz tworzą ostateczny wynik klasyfikacji bądź regresji. Uzyskuje się w ten sposób doskonałe narzędzie do bezinterwencyjnej metody generacji cech, która pozwala na poprawę dokładności działania systemu. Za protoplastę można uznać zdefiniowany na początku lat dziewięćdziesiątych wielowarstwowy neocognitron Fukushima. Jednak prawdziwy rozwój zawdzięczamy jednak profesorowi LeCun, który zdefiniował podstawową strukturę i algorytm uczący specjalizowanej sieci konwolucyjnej (zwanej splotową). Jest to sieć wielowarstwowa, której angielska nazwa Convolution Neural Network która tradycyjnie jest skracana do CNN. Aktualnie CNN stanowi podstawową strukturę stosowaną w przetwarzaniu obrazów i sygnałów. Opisana w poprzedniej części sieć MLP w klasyfikowaniu ręcznie pisanych cyfr osiągała dokładność około 88%, nawet przy długim trenowaniu. Sieć konwolucyjna inaczej splotowa dużo lepiej radzi sobie z wyszukiwaniem wzorców w obrazach. Dokładność osiągnięta przez sieć splotową już po 500 epokach uczenia sięga 98%.

9.2 Struktura sieci CNN

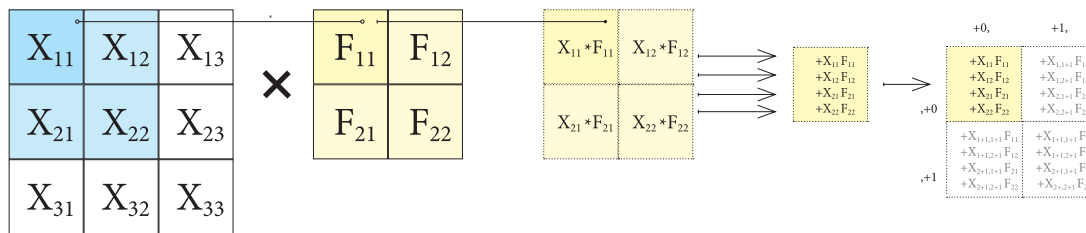
Cytat z [ossowski2023]. Sieci CNN powstały jako narzędzie analizy i rozpoznawania obrazów wzorowane na sposobie działania naszych zmysłów. Wyeliminowały kłopotliwy proces manualnego opisu cech charakterystycznych obrazów, stanowiących atrybuty wejściowe dla końcowego etapu klasyfikatora. W tym rozwiązaniu sieć sama odpowiada za generację cech. Poszczególne warstwy sieci CNN przetwarzają obrazy z warstwy poprzedzającej, poszukując prymitywnych cech (np. grup pikseli o podobnym stopniu szarości, krawędzie przecinające się itp.). Kolejne warstwy ukryte generują pewne uogólnienia cech z warstw poprzedzających organizowanych w formie obrazów (macierzy liczb). Obrazy te w niczym nie przypominają oryginałów, ale reprezentują cechy charakterystyczne

dla nich, wyrażone w formie intensywności pikseli. W efekcie kolejnego przetwarzania obrazów z poprzedniej warstwy ostatnia warstwa konwolucyjna generuje obrazy o stosunkowo niewielkich wymiarach ale o dużej liczebności, reprezentujące cechy charakterystyczne dla przetwarzanego zbioru. Obrazy te w poszczególnych warstwach są reprezentowane przez tensory (struktury 3-wymiarowe, których szerokość i wysokość odpowiada wymiarom obrazu, a głębokość jest równa liczbie obrazów). Elementy tensora ostatniej warstwy są przetwarzane na postać wektorową (1-wymiarowa tablica liczb) w procesie wypłaszczania, a następnie przekazywane na wejście układu klasyfikującego - sieci MFC.

9.3 Podstawowe operacje w sieciach CNN

Tworzenie obrazu wyjściowego z obrazu wejściowego i filtra polega na wykonaniu operacji splotu $X * F$ dwu kwadratowych macierzy liczb. Jeśli wartości pikseli obrazu wejściowego X oznaczmy $X_{(i,j)}$ a wartości filtra F oznaczmy jako $F_{(m,n)}$ obraz wejściowy Y , $Y_{(o,p)}$ wówczas każdy piksel obrazu wyjściowego obliczymy:

$$Y_{(O,P)} = \sum_{(m=1,n=1)}^{(M,N)} F_{(m,n)} * X_{(m+O,n+P)} \quad (7)$$



9.4 Inne rodzaje warstw

Obrazy uzyskane po operacji splotu są przetwarzane przez warstwę redukującą rozmiar, będzie to albo warstwa AVG - obliczenie średniej wartości sąsiednich pól, albo warstwa MAX przypisująca wartość maksymalną sąsiednich pól. Cykli splotu i redukcji obrazów może być kilka, po nich występuje warstwa łącząca - zmieniająca sekwencję obrazów w jeden ciąg wartości, które tworzą wektor wejściowy dla warstw FullConnected. Warstwą wyjściową jest warstwa klasyfikująca SoftMax. W sieciach CNN najczęściej stosowana jest funkcja aktywacji $\text{ReLU} = \max\{x, 0\}$, na wyjściu z sieci SoftMax.

9.5 Przetwarzanie wprost

Obraz wejściowy w skali szarości, lub obraz kolorowy rozbity na 3 kanały RGB zostaje przetworzony przez pierwszą grupę warstw filtrów, następnie warstwę redukującą. Wyjście z warstw redukujących podawane jest do sieci MLP, z której wyjście przetwarzane jest przez warstwę SoftMax.

9.6 Przetwarzanie wstecz

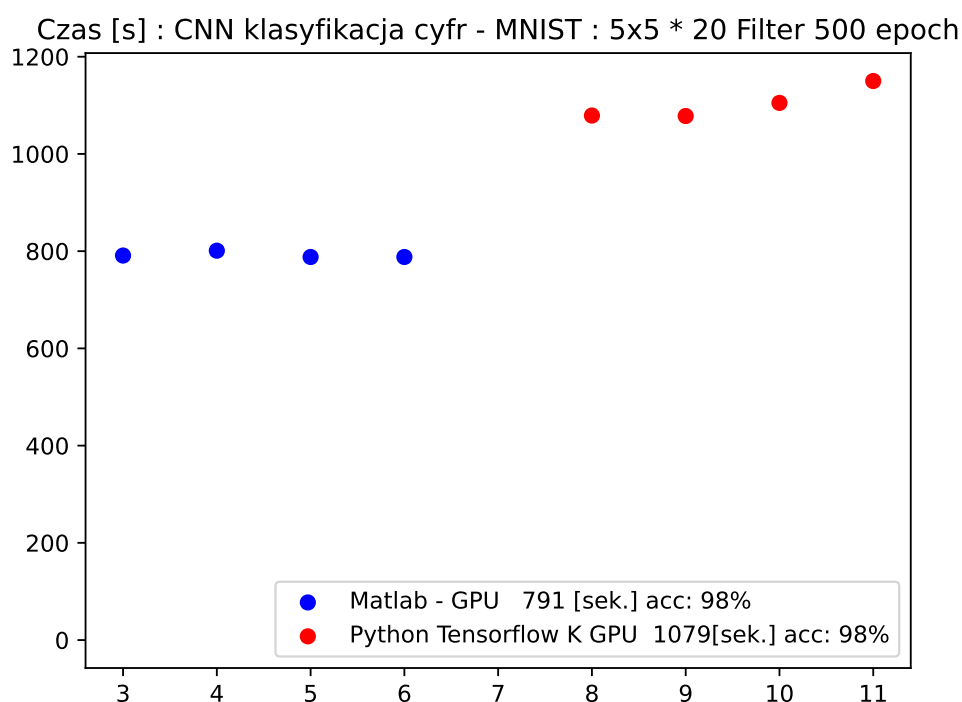
Wsteczna propagacja sygnału przez sieć softmax i MLP została omówiona w poprzednich rozdziałach. Sygnał błędu przekazywany wstecz przez warstwę AVG będzie wynosił $1/N * N$ błędu warstwy $n+1$. Błąd przekazywany przez warstwę MAX będzie przekazywany niezmieniony elementowi który miał najwyższą wartość. Do pozostałe elementów będzie przekazywana wartość 0. Zatem wagi tych potomne tych elementów nie będą aktualizowane.

Pochodna splotu:

$$\frac{\partial L}{\partial F} = Conv(Input.X, Loss.Gradient \frac{\partial L}{\partial O}) \quad (8)$$

$$\frac{\partial L}{\partial F} = FullConv(180Rotated.filter.F, Loss.Gradient \frac{\partial L}{\partial O}) \quad (9)$$

9.7 Uzyskane wyniki



9.8 Kod realizujący obliczenia w Matlab

```
1 function accuracy = accuracyCheck( first, second )
2     goals=0;
3     s=size(first);
4     s=s(2);
5     for i=(1:s)
6         val1=first(i);
7         val2=second(i);
8         if (val1==val2)
9             goals=goals+1;
10        end
11    end
12    accuracy = goals/s;
13 end
14
15
16 function index = indexOfMaxInVector( vec ) %
17     val=vec(1);
18     index=1;
19     s=size(vec);
20     s=s(1);
21     for i = (2:s)
22         if (val<vec(i))
23             index=i;
24             val=vec(i);
25         end
26     end
27 end
28
29 function aryOfInt = aryOfVectorToAryOfInt( aryOfVec )
30     s = size( aryOfVec );
31     h=s(1);
32     s=s(2);
33     aryOfInt=zeros(1,s);
34     for i=1:s
35         vec=aryOfVec(1:h,i);
36         index = indexOfMaxInVector(vec);
37         if index==10
38             index=0;
39         end
40         aryOfInt(i)=index;
41     end
42 end
43
44
```



```

45 function showx( arrayx , i )
46     img0=arrayx(1:784,i);
47     img0=img0*256;
48     image(img0);
49
50     img=zeros(28,28);
51     for i=(1:28)
52         row=img0((i-1)*28+1:(i)*28);
53         img(i,1:28)=row;
54     end
55     image(img)
56 end
57
58 if ( 1==1 )
59     percent=100;
60
61     fileIMG=fopen( 'data/train-labels-idx1-ubyte','r');
62     fileData=fread( fileIMG, 'uint8' );
63     fclose(fileIMG);
64     ytmp=fileData(9:8+percent*600)';
65     ysize=percent*600;
66     yyy=zeros(1,ysize);
67     for i=(1:ysize)
68         d=ytmp(i);
69         d=d+1;
70         yyy(i)=d ;
71     end
72     ytrain=categorical(yyy);
73
74     fileData=1;
75
76     fileIMG=fopen( 'data/t10k-labels-idx1-ubyte','r');
77     fileData=fread( fileIMG, 'uint8' );
78     fclose(fileIMG);
79
80     ytmp=fileData(9:8+percent*100)';
81     ysize=percent*100;
82     yyy=zeros(1,ysize);
83     for i=(1:ysize)
84         d=ytmp(i);
85         d=d+1;
86         yyy(i)=d ;
87     end
88     ytest=categorical(yyy);
89
90     fileIMG=fopen( 'data/train-images-idx3-ubyte','r');

```

```

91     fileData=fread( fileIMG, 'uint8' );
92     fclose(fileIMG);
93     tmp=fileData(17:16+percent*784*600);
94
95     for i=1:percent*600
96         col=tmp(1+(i-1)*784:i*784);
97         row=col';
98         ary=zeros(28,28);
99         for j=1:28
100             for k=1:28
101                 val=row(k+((j-1)*28));
102                 xtrain(j,k,1,i)=val; %/255
103             end
104         end
105     end
106     fileData=1;
107
108     fileIMG=fopen( 'data/t10k-images-idx3-ubyte','r');
109     fileData=fread( fileIMG, 'uint8' );
110     fclose(fileIMG);
111     tmp=fileData(17:16+percent*784*100);
112
113     for i=1:percent*100
114         col=tmp(1+(i-1)*784:i*784);
115         row=col';
116         ary=zeros(28,28);
117         for j=1:28
118             for k=1:28
119                 val=row(k+((j-1)*28));
120                 xtest(j,k,1,i)=val; %/255
121             end
122         end
123     end
124     fileData=1;
125 end
126
127 input = imageInputLayer([28 28 1]); % 28x28px 1 channel
128 conv = convolution2dLayer(5, 20); % 10 filter, 5x5
129 relu = reluLayer; %ReLU
130 maxPooling2dLayer(2,Stride=2);
131 fc = fullyConnectedLayer(10);
132 sm = softmaxLayer;
133 co = classificationLayer;
134
135 epochs=500;
136

```

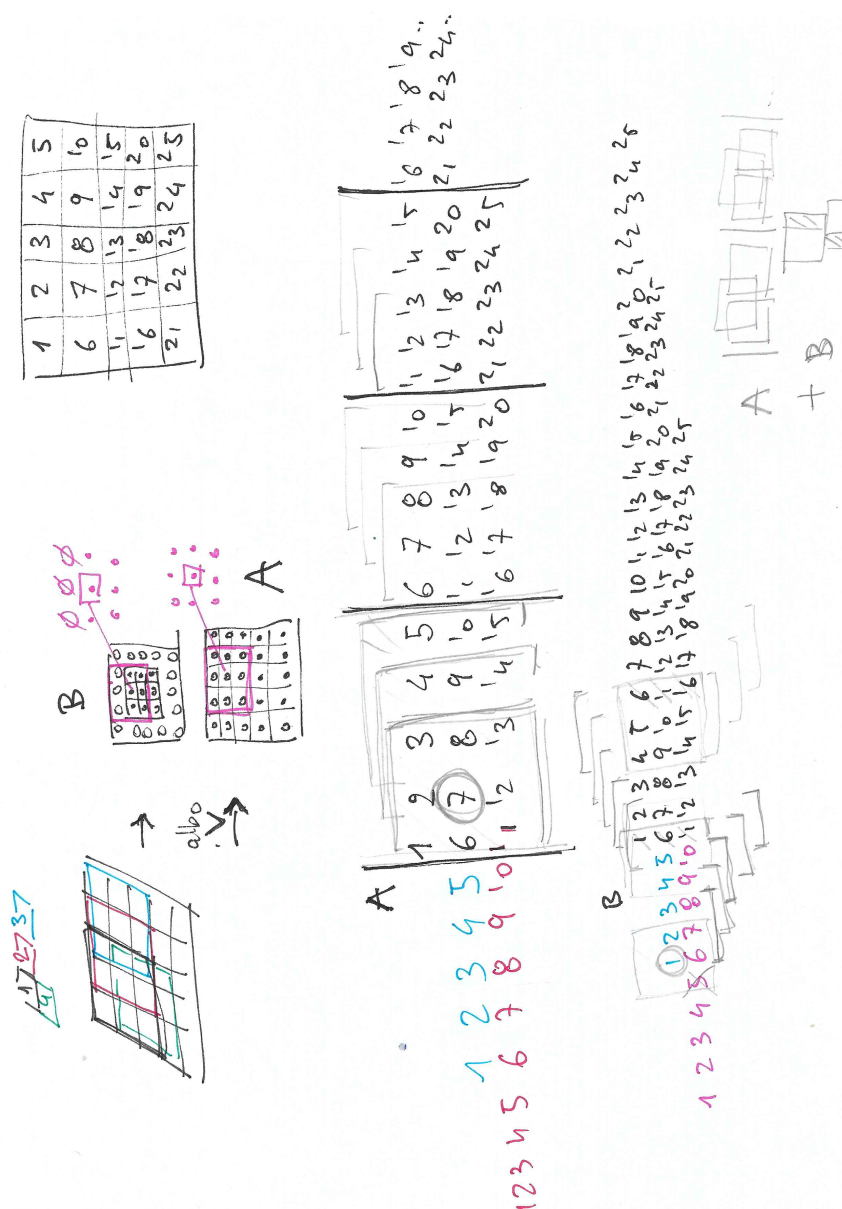
```
137 layers = [ input
138             conv
139             relu
140             fc
141             sm
142             co];
143
144 options=trainingOptions('adam', 'MaxEpochs',epochs, '
    ExecutionEnvironment','gpu','ValidationPatience',10 , 'Verbose',0);
145
146 ST = datetime('now');
147     netTransfer = trainNetwork( xtrain, ytrain, layers, options);
148
149 ED = datetime('now');
150 D = duration( ED-ST );
151
152 weights_first=netTransfer.Layers(2,1).Weights(:,:,1,1);
153 predictedLabels = classify(netTransfer, xtest);
154 accuracy = accuracyCheck( predictedLabels', ytest );
155
156 fprintf ('m[]=%f\n' , seconds(D) );
```

9.9 CNN

```
1 https://www.mathworks.com/matlabcentral/fileexchange/74760-image-
    classification-using-cnn-with-multi-input-cnn?s\_tid=
    srchtitle\_support\_results\_1\_CNN
```

9.10 Python TensorFlow

// derative of convolution [https://www.physicsforums.com/threads/ how-do-you-derive-the-derivative-of-a-convolution.403002/](https://www.physicsforums.com/threads/how-do-you-derive-the-derivative-of-a-convolution.403002/) [https://dsp.stackexchange.com/questions/46746/ how-to-evaluate-derivative-of-convolution-integral](https://dsp.stackexchange.com/questions/46746/how-to-evaluate-derivative-of-convolution-integral) <https://en.wikipedia.org/wiki/Convolution>



Rozdział 10

Podsumowanie

- Teoria: wiemy jak ma działać, ale jednak nie działa.
- Praktyka: działa, ale nie wiemy – dlaczego?
- Łączymy teorię z praktyką: nic nie działa i nie wiemy, dlaczego.

Więcej informacji na temat \LaTeX :

- <https://www.overleaf.com/learn> – przystępny tutorial na stronie Overleaf,
- <https://www.latex-project.org/> – strona domowa projektu,
- <https://www.tug.org/begin.html> – dobry zbiór odnośników do innych materiałów.

Powodzenia!

Spis rysunków

1	Model sztucznego neuronu - propagacja sygnału.	11
2	Przepływ sygnałów przez warstwę w czasie propagacji sygnału "w przód"	12
3	propagacja wstecz - uczenie sieci. (Operacje niezależne (mnożenie) oznaczone kolorem zielonym i operacje wymagające synchronizacji (dodawanie) oznaczone kolorem czerwonym)	13
4	Przepływu sygnałów wstecz i proces uczenia neuronu	14
5	Architektura LaNet-5 [YannCnn]	15
6	Porównanie struktury LaNet i AlexNet link	16
7	a) konwolucja 1 kanał 1 filtr, b) konwolucja 3 kanały 1 filtr, c) przesuwanie filtra nad obrazem	17
8	wynik działania warstw ReLU, Avg, Max przy przepływie wprost i wstecz.	18
9	Rejestry AVX, AVX2 i AVX512, operacja równoległego mnożenia AVX	22
10	Sumator wielokanałowy	23
11	Operacje niezależne i zależne sum	23
12	drzewo zdegenerowane i niezdegenerowane	24
13	Porównanie czasów obliczania regresji liniowej	31
14	Szacowanie parametrów sieci	33
15	podgląd próbki wektorów uczących	34