Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

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, Maltab, Java, C++. Do budowania sieci wykorzystano biblioteki: YOLO11, (TensoRT? 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 is abstract. This one is a little too short as it should occupy the whole page.

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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

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

Spis treści

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-stych 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 1993r. badania nad m.in. rozpoznawaniem ręcznego pisma, czy sterowaniem ruchem robota.[1]

W ostatnich latach ogromny postęp w sztucznej inteligencji dokonał się za pośrednictwem tak zwanego głębokiego uczenia. Za protoplastę głębokich sieci można uznać zdefiniowany na początku lat 90-tych wielowarstwowy neocognitron Fukushimy, jednak prawdziwy rozwój zawdzięczamy profesorowi LeCun, który zdefiniował podstawową strukturę i algorytm uczący specjalizowanej sieci konwolucyjnej CNN. Aktualnie sieci CNN stanowią podstawową strukturę stosowaną na szeroką skalę w przetwarzaniu sygnałów i obrazów. W miedzyczasie powstało wiele odmian sieci będącej modyfikacją struktury podstawowej (R-CNN, AlexNET, GoogLeNet, ResNet, , U-Net, YOLO)[4]

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 11.

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.

Przykłady obliczeń Python i Matlab zaczerpnięto z [4]. **Pełen kod dostępny na github:** https://github.com/piotrHeinzelman/inz/tree/main/MixedProj w przypadku Matlab i Python wykorzystano dostępne funkcje, w przypadku Java zastosowano obliczenia wg. wzoru 26. Wyniki obliczeń są porównywalne, założono że zaproponowane implementacje są zgodne. Nie porównywano czasów przygotowania danych ani czytania plików.

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 oraz CNN, poprzez realizacje i testy własnego rozwiązania zwłaszcza z wykorzystaniem możliwości obliczeniowych karty graficznej.

1.2 Układ pracy

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. Opisano także kilka wybranych problemów ograniczających wydajność obliczeń w systemach cyfrowych.

W drugiej części przedstawiono wybrane cechy języków, a także ich wpływ na decyzję o wyborze rozwiązania.

W trzeciej części zaprezentowano zebrane informacje o językach oraz podsumowano zebrane dane i ich relacje. Wnioski na temat wyższości jednego rozwiązania nad drugim będą należeć od konkretnych twórców sieci i będą zależały od ich wymagań i możliwości.

```
1
2  -- INFO --- do napisania kod dla : ---
3
4  5) realizacja Klasyfikowanie pisma odrecznego sieci glebokie CNN
       realizacja sieci z wykorzytsniem bibliotek.
5  + a) Matlab
6  + b) Python tensorflow.keras
7  c) wlasna implementacja Java lub C++ libtorch
8
9  Cel podstawowy: porownanie wydajnosci implementacji. Zbadanie wydajnosci
       uczenia sieci.
10 Cele dodatkowe: potwierdzenie poprawnosci wlasnej implementacji w Java
       lub C++
11
12
```

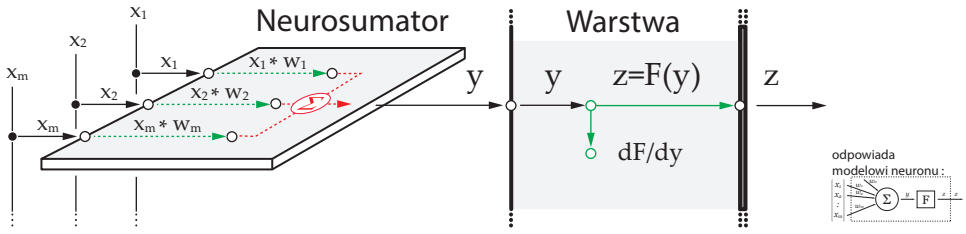
```
6) Rozpoznawanie twarzy sieci glebokie CNN
a) Matlab
b) Python tensorflow.keras
c) wlasna implementacja Java lub C++ libtorch

Cel podstawowy: porownanie wydajnosci implementacji.


7) Klasyfikacji obrazow z wykorzystaniem sieci glebokiej CNN
a) Matlab
b) Python tensorflow.keras
c) C++ libtorch

Cel podstawowy: porownanie wydajnosci implementacji i wydajnosci procesu
    uczenia nadzorowanego.

8) detekcja i segmentacji obiektow sieci glebokiej CNN
b) Python tensorflow.keras
c) C++ libtorch

Cel podstawowy: porownanie wydajnosci implementacji.
```

Rozdział 2

Perceptron wielowarstwowy - MLP

2.1 Propagacja sygnału

Model neuronu [1], rys. 1 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:

$$y = \sum_{i=0}^{m} x_i * w_i \tag{1}$$

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.

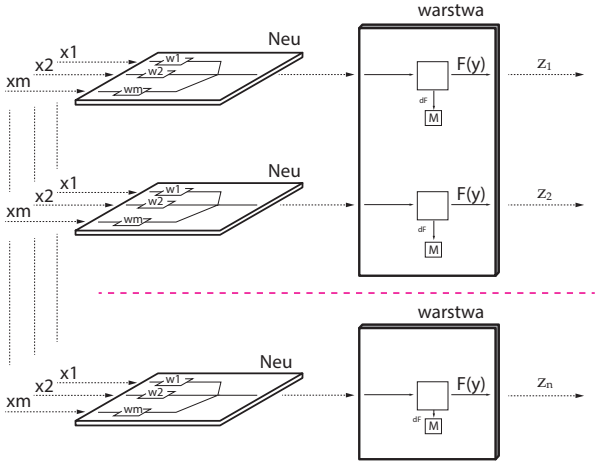
$$z = F(y) \tag{2}$$

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

Dodatkowo 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>=0 to z=0	jeśli y<0 to 1, jeśli y>=0 to 0
softmax	$z_i = \frac{e^{x_i}}{\sum e^{x_i}}$	dla prawidłowej klasy k: $y_k(1-y_k)$,
softmax		dla pozostałych klas: $-y_i * y_k$

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

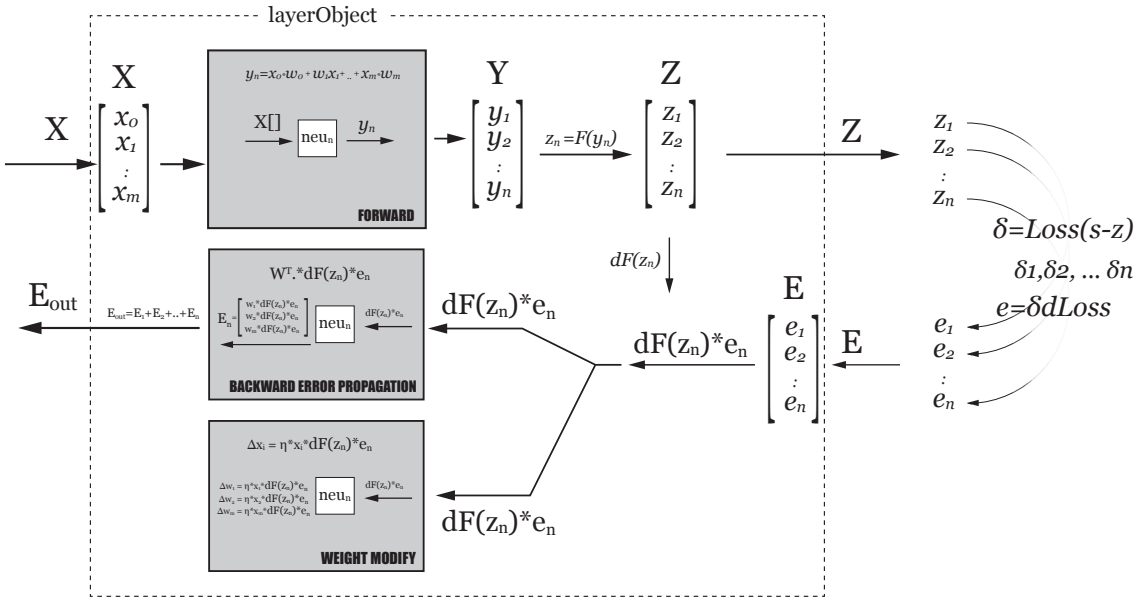


Rysunek 2. Przepływ sygnałów przez warstwę w czasie propagacji sygnału "w przód"

Odpowiedzią n neuronów zorganizowanych w warstwę jest zbiór wartości $[z_1, z_2, ..., z_n]$ (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

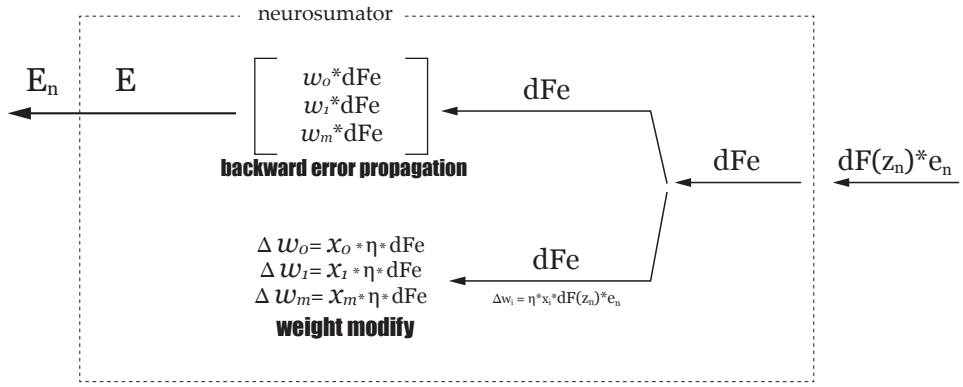


Rysunek 3. Przepływu sygnałów w obiekcie Warstwa

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

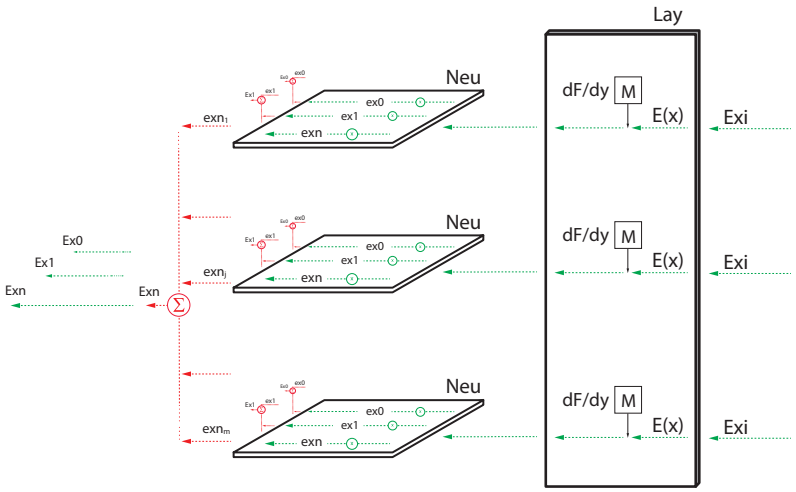
Dla warstwy wyjściowej logistycznej może to być różnica między oczekiwaną odpowiedzią a odpowiedzią uzyskaną. $[e_1 = s_1 - z_1, e_2 = s_2 - z_2, ..., e_n = s_n - z_n]$, lub w zapisie wektorowym:$E = S - Z$, lub gdzie S jest wielkością oczekiwaną.

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 $E_n = [w_0 * dFe, w_1 * dFe, ..., w_m * dFe]$. 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.

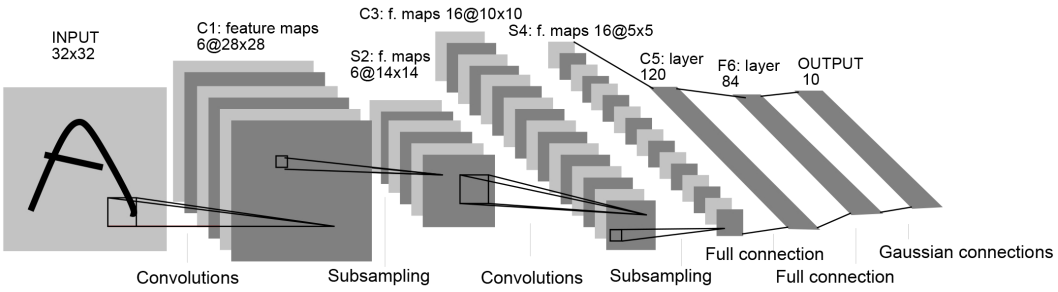


Rysunek 5. propagacja wstecz - uczenie sieci. (Operacje niezależne (mnożenie) oznaczone kolorem zielonym i operacje wymagające synchronizacji (dodawanie) oznaczone kolorem czerwonym)

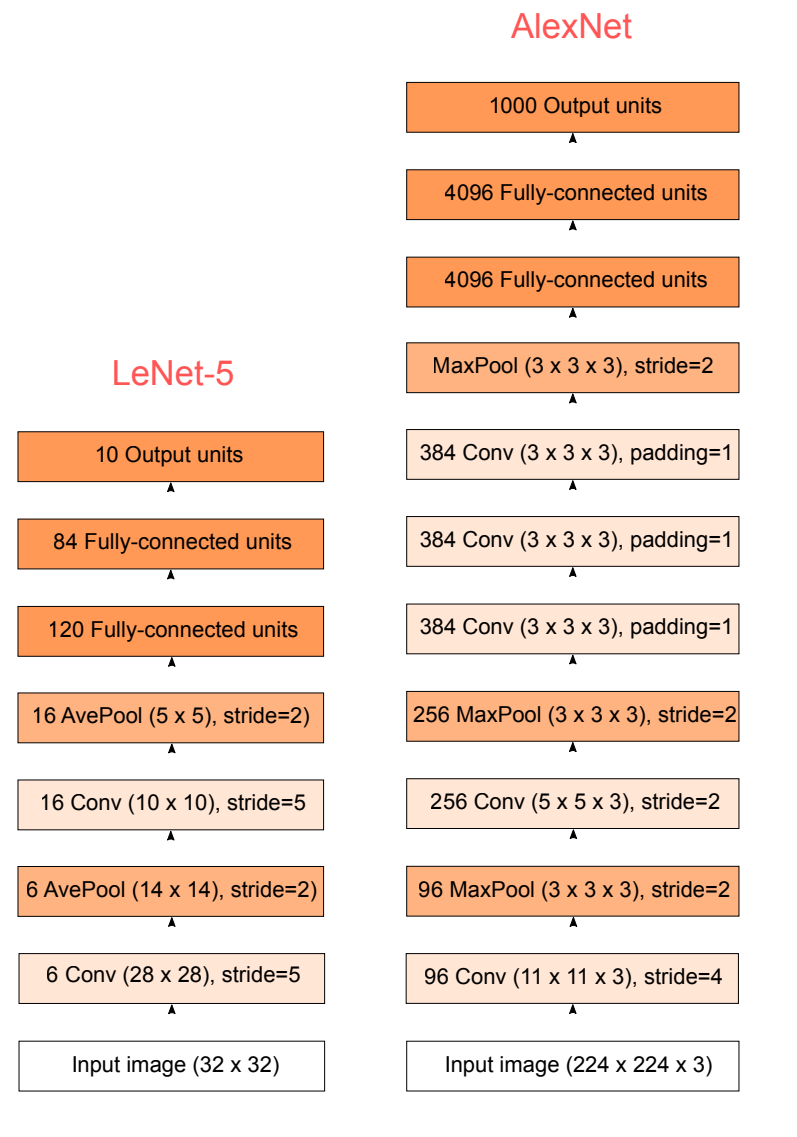
Rozdział 3

Sieć splotowa CNN

Cytat z [4]. 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 liczebności kanałów, reprezentujące cechy charakretystyczne dla przetwarzanego zbioru. Próbka 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.



Rysunek 6. Architektura LaNet-5 [6]

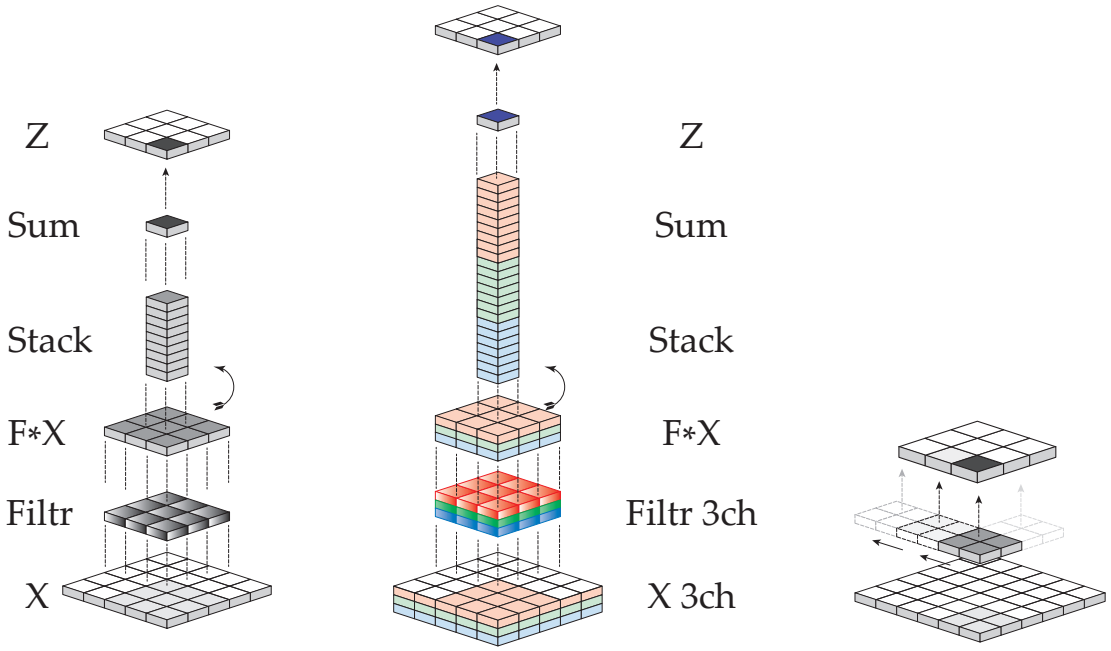


Rysunek 7. Porównanie struktury LaNet i AlexNet link

3.1 Operacja splotu

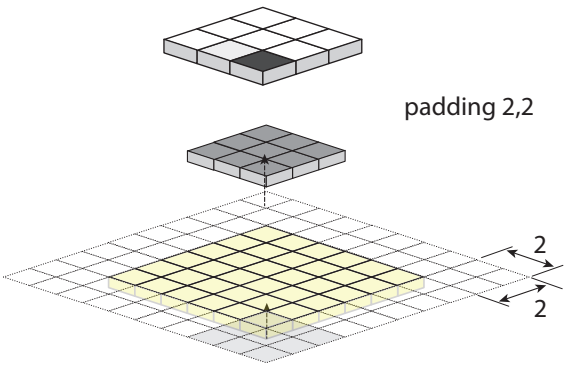
Jeśli wartości pikseli obrazu wejściowego X oznaczymy $X_{(i,j)}$ a wartości filtra F oznaczymy jako $F_{(m,n)}$ obraz wejściowy Y, $Y_{(o,p)}$ wówczas każdy piksel obrazu wyjściowego obliczymy:

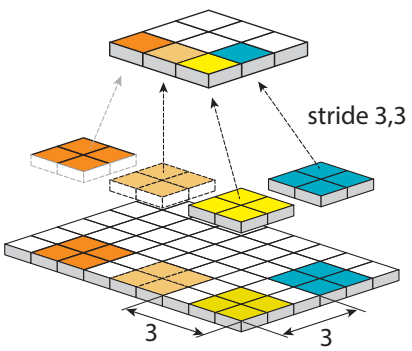
$$Y_{(O,P)} = \sum_{(m=1,n=1)}^{(M,N)} F_{(m,n)} * X_{(m+O,n+P)} \tag{3}$$



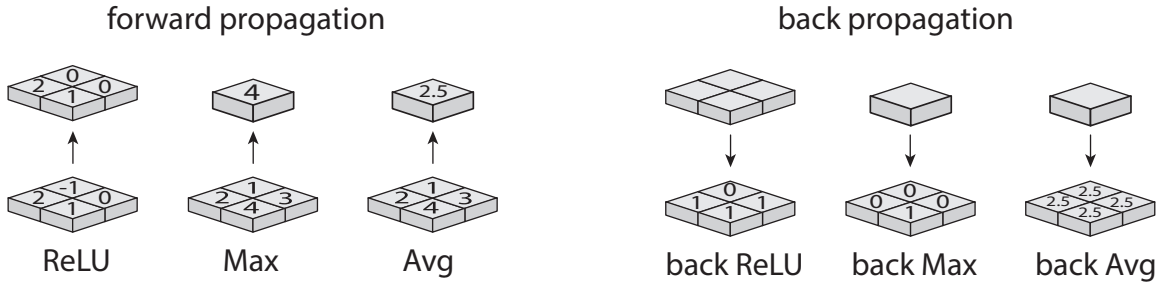
Rysunek 8. konwolucja 1 kanał 1 filtr, konwolucja 3 kanały 1 filtr, przesuwanie filtra nad obrazem

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





3.2 ReLU i warstwy redukujące wymiar



Rysunek 9. wynik działania warstw ReLU, Avg, Max przy przepływie wprost i wstecz.

Po operacji splotu można zastosować warstwę ReLU. Funkcja aktywacji ReLU = max{x, 0}. Zerowane są wartości ujemne, funkcja nie ma ciągłej pochodnej, przy obliczeniach przyjmowano że pochodna wynosi 1 gry 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 liczbę 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.

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

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

3.5 Propagacja błędu przez warstwę splotową

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

3.6 Yolo

YoLo schemat [2]

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

Rozdział 4

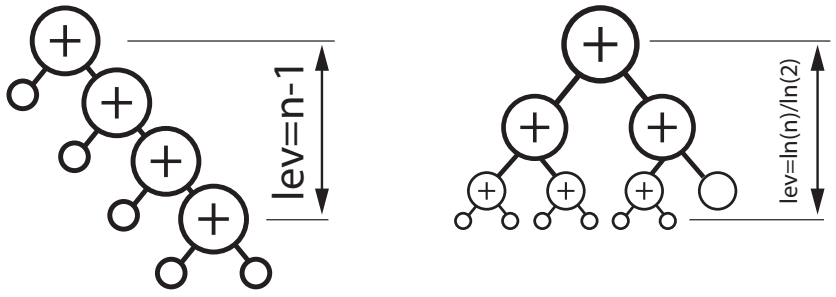
Problemy wydajnościowe

4.1 Synchronizacje sygnałów

Operacje niezależne - na rysunkach oznaczono kolorem zielonym - takie jak mnożenie wag przez wartości wejściowe można wykonywać równolegle, wykorzystując osobne wątki, procesy, rdzenie. Operacje sumowania oznaczono kolorem czerwonym - nie są operacjami niezależnymi, wszystkie składniki sumy muszą być dostępne w chwili wykonywania działania.

4.2 Dodawanie częściowo równoległe

Sumowanie a+b+c+d realizowane jest analogiczne do zdegenerowanego drzewa binarnego czyli (((a+b)+c)+d)... nie jest operacją równoległą. Przy dużej ilości składników można wymusić dodawanie jak w zwykłym drzewie binarnym przez dodanie nawiasów, wówczas działanie stanie się częściowo równoległe ((a+b) + (c+d) ...).



Rysunek 10. drzewo zdegenerowane i niezdegenerowane

Porównanie szybkości dodawania: (C++)
czas wykonania: **0.2478 [s]** $b = a1 + a2 + a3 + a4 + ... + a1024;$
czas wykonania: **0.0956 [s]** $b = ((..((a1 + a2) + (a3 + a4)) + ((a5 + a6)...$

4.3 Kodowanie

Z punktu widzenia matematyków różnica pomiędzy 1.0000000000000001 a 1.0 jest spora, ponieważ pomiędzy tymi wartościami mamy nieskończenie wiele liczb, natomiast przy obliczeniach, których dokonujemy używając maszyn cyfrowych musimy zważyć, czy zwiększać dokładność zwiększając ilość bitów zajmowanych przez liczby, jednocześnie zmniejszać szybkość operacji, czy wykorzystać mniej dokładne reprezentacje zyskując na szybkości pewnych operacji.

Najczęściej używaną reprezentacją liczb rzeczywistych jest opisana przez normę IEEE 754. Norma ta definiuje zapis liczby w 32, 64 liczb 128 bitach. W językach programowania dostępne jako klasy *Float*, *Double*.

- Matlab: przypisanie a=1.0000000000000001 // a=1;
- Python: print (.000600000000000000001) // 0.0006.
- Java: $0.1 + 0.2 = 0.30000000000000004$.

Do zastosowań specjalnych możemy użyć specjalnych formatów zapisu np. BigDecimal w Java o większej dokładności (większej liczbie cyfr znaczących).
Operowanie na typach specjalnych drastycznie zmniejsza wydajność operacji.
Float - 0.002 [sek.] / BigDecimal 0.042 [sek.]

Rozdział 5

Cechy języków

popularność, dostępność bibliotek, wygoda instalacji, wsparcie obliczeń na kartach graficznych, stopień trudności języka, koszt licencji

Rozdział 6

Jednowymiarowa regresja liniowa

6.1 Wprowadzenie

Przykłady obliczeń Python i Matlab zostały zaczerpnięte z [4]. **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 26. 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.

6.2 Prowadzenie badania

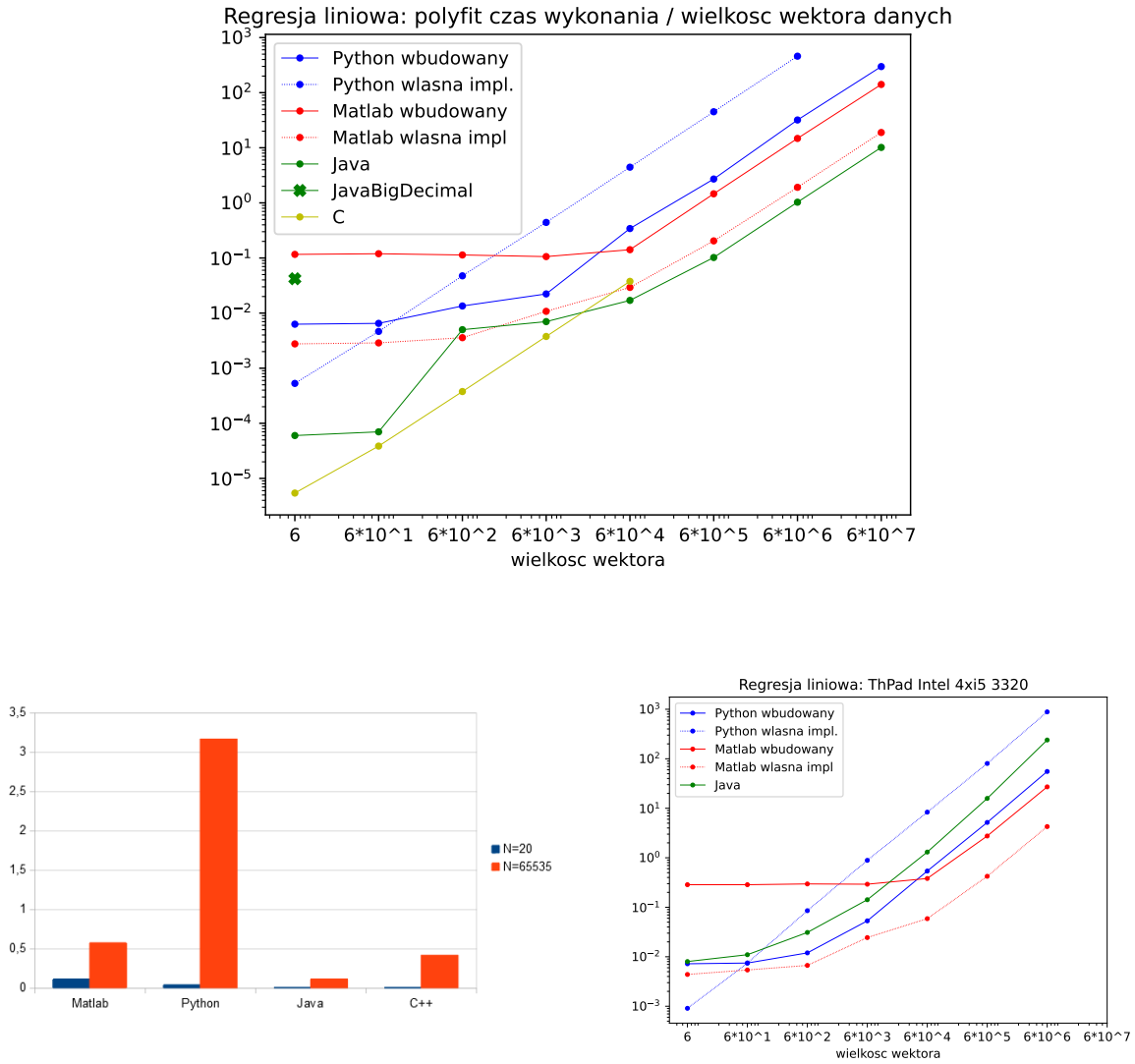
Programy w kolejnych językach uruchamiam poniższym kodem:

```
./matRun >> output.txt
./pyRun  >> output.txt
./javaRun >> output.txt

//matRun
echo "---- Matlab app start: ----"
/usr/local/MATLAB/R2024a/bin/matlab  -nodisplay -nosplash -nodesktop -batch 'run matlab.m'

//pyRun
echo "---- Python app start: ---- "
python main.py

// javaRun
javac Main.java
echo "---- Java app start: ----"
java Main
```

6.3 Kod realizujący obliczenia

```
//                  ---- MATLAB ----

    a = polyfit(x,y,1);

//                  ---- Python ----

    a = np.polyfit(x,y,1)

//                  ----  Java ----

     for (int i = 0; i < x.length; i++) {
            xsr += x[i];
            ysr += y[i];
        }
        xsr = xsr / x.length;
        ysr = ysr / y.length;

        for (int i = 0; i < x.length; i++) {
            sumTop += ((x[i] - xsr) * (y[i] - ysr));
            sumBottom += ((x[i] - xsr) * (x[i] - xsr));
        }
        w1 = sumTop / sumBottom;
        w0 = ysr - w1 * xsr;

//               ---- C++ ----

      for ( int i=0; i<len; i++ ){
         xsr +=  X[i];
         ysr +=  Y[i];
      }
   xsr=xsr / len; ysr=ysr / len;
   double sumTop=0.0;
   double sumBottom=0.0;

      for ( int i=0;i<len;i++ ){ //  xtmp = X[i]-sr ! ;
       sumTop    += ((X[i]-xsr)*(Y[i]-ysr));
      sumBottom += ((X[i]-xsr)*(X[i]-xsr));
      }
      w1 = sumTop / sumBottom;
      w0 = ysr -(w1 * xsr) ;

.
```

6.4 Uzyskane wyniki



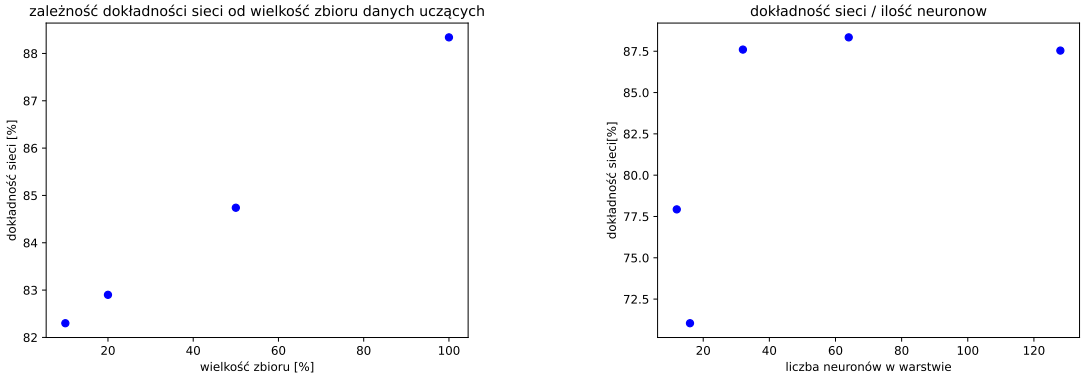
Rysunek 11. Porównanie czasów obliczania regresji liniowej

Rozdział 7

Klasyfikowanie pisma odręcznego MLP

7.1 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 zestawy 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)

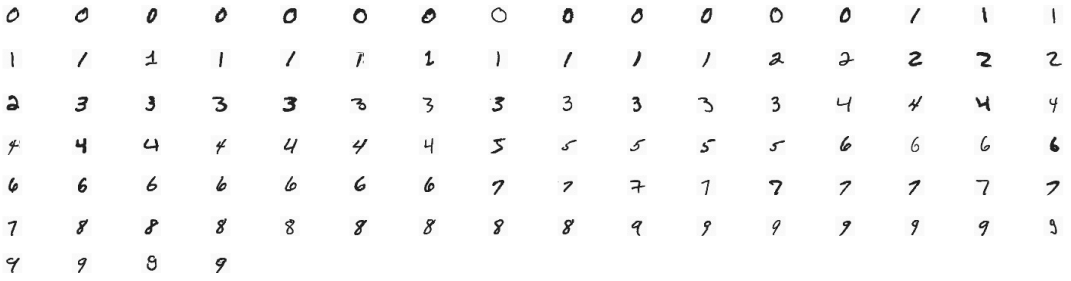


Rysunek 12. Szacowanie parametrów sieci

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.

7.2 Dane treningowe

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



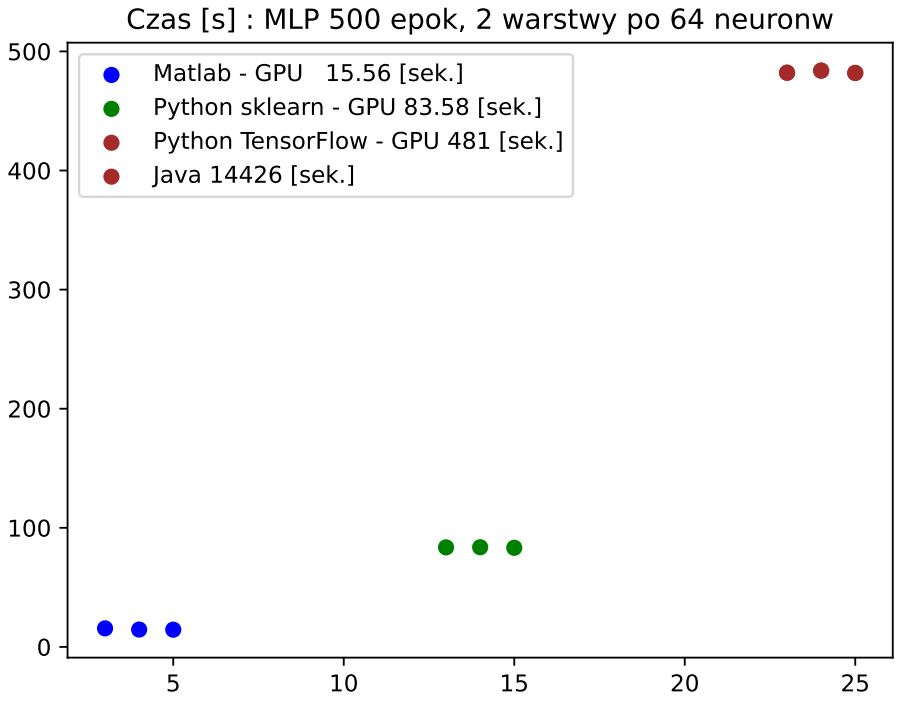
Rysunek 13. podgląd próbki wektorów uczących

poniżej kod ładujący dane, oraz metody narzędziowe:

```
1
2  public  void prepareData( int percent ){
3     trainYfile =  loadBin( "../data/" + "t10k-images-idx3-ubyte", 8, percent*600 ); //offset=8, size=percent*600  // OK
4     testYfile =  loadBin( "../data/" + "t10k-labels-idx1-ubyte",  8, percent*100 ); // offset=8, size=percent*100 // OK
5
6     trainY = new float[percent*600][];
7     testY  = new float[percent*100][];
8     //train Y
9     for (int i=0;i<percent*600;i++){
10       trainY[i] = new float[]{ 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
11       trainY[i][ trainYfile[i] ]=1.0f;
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[] vectorSubstSsubZ( 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  }
```

7.3 Uzyskane wyniki



7.4 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)  );
```

7.5 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:

```
// main.java
private float[][] testX; private float[][] testY; private float[][] trainX; private float[][] trainY;

private Layer layer1; private Layer layer2; private Layer layer3;

private Tools tools = new Tools();
int numOfEpoch=500;
float[] CSBin_data=new float[numOfEpoch];

public void prepare() {
    tools.prepareData( 50 );
    testX = tools.getTestX();
    testY = tools.getTestY();
    trainX = tools.getTrainX();
    trainY = tools.getTrainY();

    layer1=new Layer( LType.sigmod , 64 ,784 ); layer1.setName("Layer1"); // n neurons
    layer2=new Layer( LType.sigmod , 64 ,64 ); layer2.setName("Layer2"); // n neurons
    layer3=new Layer( LType.softmaxMultiClass , 10 ,64 ); layer3.setName("Layer3"); // n neurons
    layer1.rnd();  layer2.rnd();  layer3.rnd();
}

public void run() {

float Loss = 0.0f;
for (int epoch = 0; epoch < numOfEpoch; epoch++) {
    for ( int index = 0; index < trainX.length; index++ ) {
        layer1.setX( trainX[ index ] );
        layer1.nForward();
        layer2.setX( layer1.getZ() );
        layer2.nForward();
        layer3.setX( layer2.getZ() );
        layer3.nForward();

        float[] S_Z = tools.vectorSubstSsubZ( trainY[ ind_ex ], layer3.getZ() );
        layer3.nBackward( S_Z );
        layer2.nBackward( layer3.getEout() );
        layer1.nBackward( layer2.getEout() );
    }
}
        // check accuracy
int len = testX.length;
int accuracy = 0;
for (int i = 0; i < len; i++) {
    layer1.setX( testX[i] );
    layer1.nForward();
    layer2.setX( layer1.getZ() );
    layer2.nForward();
    layer3.setX( layer2.getZ() );
    layer3.nForward();

    int netClassId = tools.getIndexMaxFloat( layer3.getZ() );
    int fileClassId = tools.getIndexMaxFloat( testY[i] );
    if (fileClassId == netClassId) { accuracy++; }
    }
System.out.println(100.0f * accuracy / len + "%");
}}
```

```
// neuron.java
public class Layer {
    private String name;
    private  LType lType;
    private  Neuron[] neurons;
    private  float X[];
    private  float Y[];
    private  float Z[];
    private  float dFofZ[];
    private  float Eout[]; // S-Z for last

    public Layer( LType lType, int n, int m ) {// n - number of neurons & output size Y[n], Z[n]
        this.lType=lType;                      // m - number of inputs  = input  size X[m]
        this.neurons = new Neuron[n];
        for (int i=0; i<n; i++){
            this.neurons[i]=new Neuron(m, this);
        }
        X = new float[m];
        Y = new float[n];
        Z = new float[n];
        dFofZ = new float[n];
        Eout = new float[m];
    }

    public void setWmn( int n, int m, float wji ){ neurons[n].setWm( m, wji ); }

    public void rnd(){
        Random random=new Random();
        for ( Neuron neu : neurons ) {
            for ( int m=0; m<X.length; m++ ) {
                neu.setWm( m , (float)( -1.0f+2.0f*random.nextFloat() )  );
            }
        }
    }

    public float[] nForward() {
        switch (lType) {
            case sigmod:
            case sigmod_CrossEntropy_Binary:{
                for (int n = 0; n < neurons.length; n++) {
                    Y[n] = neurons[n].Forward(X);
                    Z[n] = F(Y[n]);
                    dFofZ[n] = dF(Z[n]);
                }
                for (int m=0;m<Eout.length;m++){ Eout[m]=0; }
                return Z;
            }
            case softmaxMultiClass: {
                int len = neurons.length;
                float sum = 0.0f;
                float max = Y[0] = neurons[0].Forward(X);
                for (int i=1;i<len;i++){
                    dFofZ[i] = 1;
                    Y[i]=neurons[i].Forward(X);
                    if (Y[i]>max) { max=Y[i]; }
                }
                for (int i = 0; i < len; i++) {
                    Y[i] = (float) Math.exp( Y[i]-max );
                    sum += Y[i];
                }
                for (int i = 0; i < len; i++) {
                    Z[i] = Y[i] / sum;
                }
                return Z;
            }
            default: { return Z; }
        }
    }

    public void nBackward( float[] Ein ){ // S-Z or Ein
        for ( int i=0;i<neurons.length;i++ ){ Eout[i]=0.0f; }
        for ( int n=0; n< neurons.length; n++ ){
            neurons[n].Backward( Ein[n] * dFofZ[n] );
        }
    }


```

```
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_dFIznI ) {
29         float[] X = parent.getX();
30         for ( int m=0; m<W.length; m++ ) {
31             parent.getEout()[m] += ( W[m] * en_x_dFIznI );
32             W[m] += mu * en_x_dFIznI * X[m];
33         }
34     }
35 }
```

7.6 Pyton -sklearn

```
import numpy as np
import time
import struct
import sklearn.neural_network as snn

def readFileX ( fileName , offset, percent, multi ):
    file=open( fileName, 'rb' )
    file.read( offset )
    data=np.fromfile( fileName, np.uint8, percent*100*784*multi, '',
   offset )
    data=data.reshape(percent*100*multi, 784)
    data=1-(data/128)
    file.close()
    return data

def readFileY ( fileName , offset, percent, multi ):
    file=open( fileName, 'rb' )
    file.read( offset )
    len=percent*100*multi
    data=np.fromfile( fileName, np.uint8, len, '', offset )
    out=[]
    for i in range ( len ):
        tmp=[0,0,0,0,0,0,0,0,0,0]
        tmp[ data[i]] = 1
        out.append( tmp )
    file.close()
    return out

percent=50
trainX = readFileX ('data/train-images-idx3-ubyte', 16, percent ,6 )
trainY = readFileY ('data/train-labels-idx1-ubyte', 8, percent, 6 )
testX = readFileX ('data/t10k-images-idx3-ubyte', 16, percent, 1  )
testY = readFileY ('data/t10k-labels-idx1-ubyte', 8, percent, 1 )

net = snn.MLPClassifier(hidden_layer_sizes=(64,64), random_state=1,
   activation='logistic', solver='sgd' )
net.fit( trainX, trainY )

start=time.time()
for i in range(1):
    for epo in range (500):
        net.partial_fit( trainX, trainY )
print("Time: ", time.time()-start, score: ",net.score(testX, testY))
```

7.7 Pyton -tensorflow

```
import tensorflow as tf
import numpy as np
import time
from tensorflow.keras.backend import clear_session
import matplotlib.pyplot as plt

physical_devices = tf.config.list_physical_devices('GPU')
if physical_devices:
   for gpu in physical_devices:
      tf.config.experimental.set_memory_growth(gpu, True)

# params
epochs = 500
percent = 50
num_classes = 10

def readFileX ( fileName , offset, percent, multi ):
    file=open( fileName, 'rb' )
    file.read( offset )
    data=np.fromfile( fileName, np.uint8, percent*100*784*multi, '',
   offset )
    data=data.reshape(percent*100*multi, 784)
    data=(data/255)
    file.close()
    return data

def readFileY ( fileName , offset, percent, multi ):
    file=open( fileName, 'rb' )
    file.read( offset )
    len=percent*100*multi
    data=np.fromfile( fileName, np.uint8, len, '', offset )
    file.close()
    return data

trainX = readFileX ('data/train-images-idx3-ubyte', 16, percent ,6 )
trainY = readFileY ('data/train-labels-idx1-ubyte', 8, percent, 6 )
testX = readFileX ('data/t10k-images-idx3-ubyte', 16, percent, 1  )
testY = readFileY ('data/t10k-labels-idx1-ubyte', 8, percent, 1 )

trainX = trainX.astype("float32") # / 255
testX = testX.astype("float32") # / 255
trainX = trainX.reshape(6*percent*100, 784).astype("float32") / 255
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)
```

7.8 C++ -on ? CUDA ?

Rozdział 8

Sieci głębokie

8.1 Wstęp

Cytat z [4]. 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 Fukushimy. Jednak prawdziny 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 osiągana przez sieć splotową już po 500 epokach uczenia sięga 98%.

8.2 Struktura sieci CNN

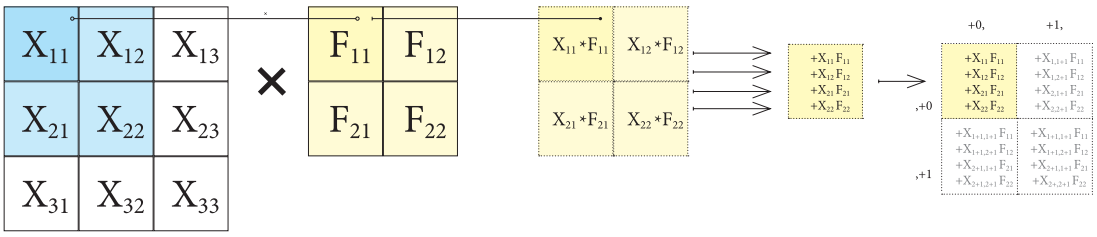
Cytat z [4]. Sieci CNN powstały jako narzędzie analizy i rozpoznawania obrazów wzorowane na sposobie działania naszych zmysłów. Wyeliminowały kłopotliwy proces manualnego opisu cech charakterystycznych obrazów, stanowiących atrybuty wejściowe dla końcowego etapu klasyfikatora. W tym rozwiązaniu sieć 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 charakretystyczne 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.

8.3 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 oznaczymy $X_{(i,j)}$ a wartości filtra F oznaczymy jako $F_{(m,n)}$ obraz wejściowy Y, $Y_{(o,p)}$ wówczas każdy piksel obrazu wyjściowego obliczymy:

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



8.4 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 klika, 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 ReLU $= \max\{x, 0\}$, na wyjściu z sieci SoftMax.

8.5 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.

8.6 Przetwarzanie wstecz

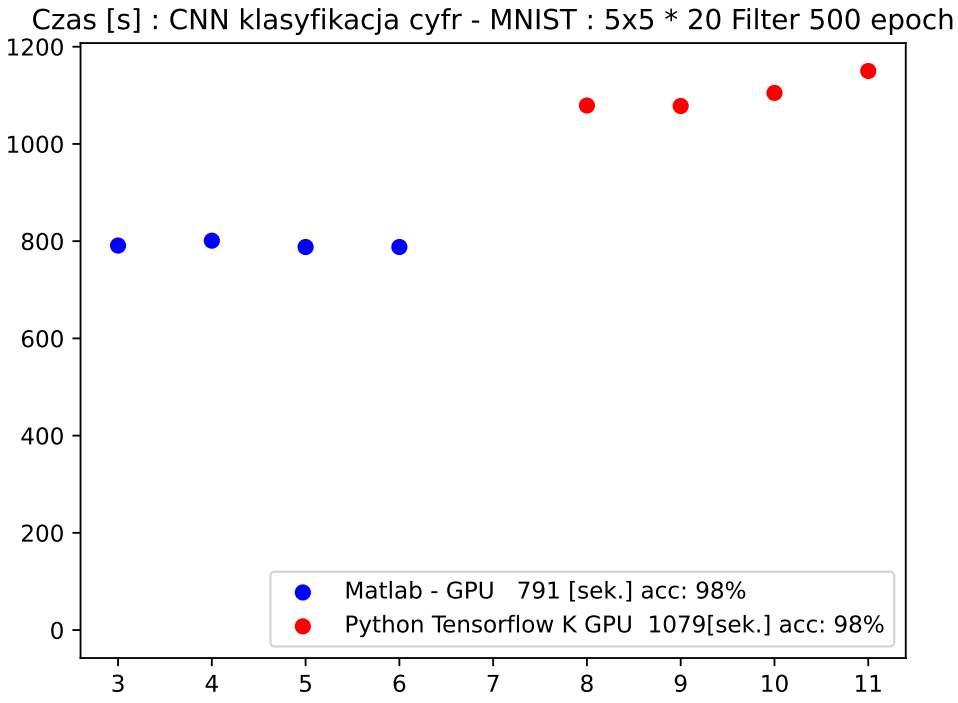
Wsteczna propagacja sygnału przez sieć softmax i MLP została omówiona w poprzednich rozdziałach. Sygnał błedu przekazywamy wstecz przez warstwę AVG będzie wynosił $1/N * N$ błędu warstwy n+1. Błąd przekazywany przez warstwę MAX będzie przekazywany niezmieniony elementowi który miał najwyższą wartość.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

Pochodna splotu:

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

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

8.7 Uzyskane wyniki



8.8 Kod realizujący obliczenia w Matlab

```
function accuracy = accuracyCheck( first, second )
    goals=0;
    s=size(first);
    s=s(2);
    for i=(1:s)
        val1=first(i);
        val2=second(i);
        if (val1==val2)
            goals=goals+1;
        end
    end
    accuracy = goals/s;
end


function index = indexOfMaxInVector( vec ) %
    val=vec(1);
    index=1;
    s=size(vec);
    s=s(1);
    for i = (2:s)
        if (val<vec(i))
            index=i;
            val=vec(i);
        end
    end
end

function aryOfInt = aryOfVectorToAryOfInt( aryOfVec )
    s = size( aryOfVec );
    h=s(1);
    s=s(2);
    aryOfInt=zeros(1,s);
    for i=1:s
      vec=aryOfVec(1:h,i);
      index = indexOfMaxInVector(vec);
      if index==10
          index=0;
      end
      aryOfInt(i)=index;
    end
end


```

```
function showx( arrayx , i )
    img0=arrayx(1:784,i);
    img0=img0*256;
    image(img0);

    img=zeros(28,28);
        for i=(1:28)
            row=img0((i-1)*28+1:(i)*28);
           img(i,1:28)=row;
        end
    image(img)
end

if ( 1==1 )
    percent=100;

    fileIMG=fopen( 'data/train-labels-idx1-ubyte','r');
    fileData=fread( fileIMG, 'uint8' );
    fclose(fileIMG);
    ytmp=fileData(9:8+percent*600)';
    ysize=percent*600;
    yyy=zeros(1,ysize);
    for i=(1:ysize)
        d=ytmp(i);
        d=d+1;
        yyy(i)=d ;
    end
    ytrain=categorical(yyy);

    fileData=1;

    fileIMG=fopen( 'data/t10k-labels-idx1-ubyte','r');
    fileData=fread( fileIMG, 'uint8' );
    fclose(fileIMG);

    ytmp=fileData(9:8+percent*100)';
    ysize=percent*100;
    yyy=zeros(1,ysize);
    for i=(1:ysize)
        d=ytmp(i);
        d=d+1;
        yyy(i)=d ;
    end
    ytest=categorical(yyy);

    fileIMG=fopen( 'data/train-images-idx3-ubyte','r');
```

```
    fileData=fread( fileIMG, 'uint8' );
    fclose(fileIMG);
    tmp=fileData(17:16+percent*784*600);

    for i=1:percent*600
        col=tmp(1+(i-1)*784:i*784);
        row=col';
        ary=zeros(28,28);
        for j=1:28
            for k=1:28
                val=row(k+((j-1)*28));
                xtrain(j,k,1,i)=val; %/255
            end
        end
    end
    fileData=1;

    fileIMG=fopen( 'data/t10k-images-idx3-ubyte','r');
    fileData=fread( fileIMG, 'uint8' );
    fclose(fileIMG);
    tmp=fileData(17:16+percent*784*100);

    for i=1:percent*100
        col=tmp(1+(i-1)*784:i*784);
        row=col';
        ary=zeros(28,28);
        for j=1:28
            for k=1:28
                val=row(k+((j-1)*28));
                xtest(j,k,1,i)=val; %/255
            end
        end
    end
    fileData=1;
end

input = imageInputLayer([28 28 1]);  % 28x28px 1 channel
conv = convolution2dLayer(5, 20); % 10 filter, 5x5
relu = reluLayer;                    %reLU
maxPooling2dLayer(2,Stride=2);
fc = fullyConnectedLayer(10);
sm = softmaxLayer;
co = classificationLayer;

epochs=500;

```

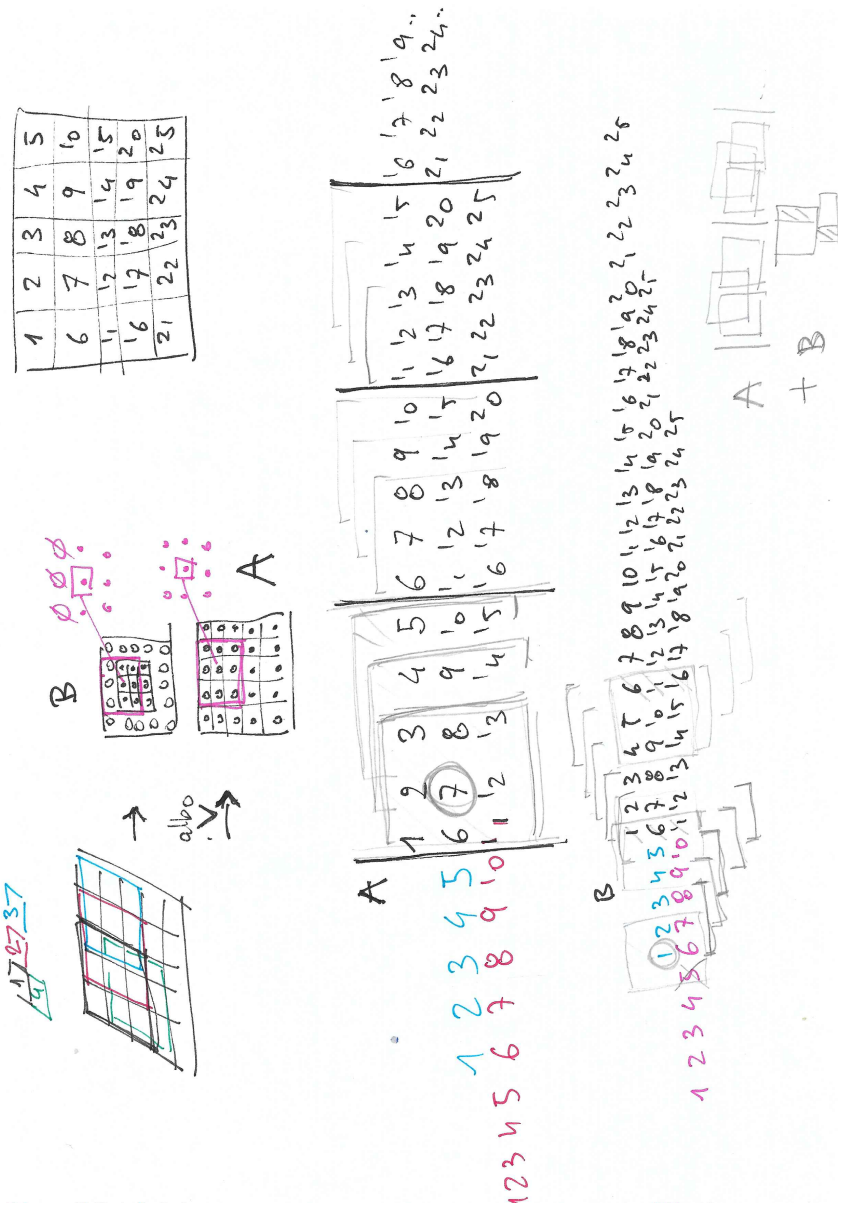
```
layers = [ input
    conv
    relu
    fc
    sm
    co];

options=trainingOptions('adam', 'MaxEpochs',epochs, '
   ExecutionEnvironment','gpu','ValidationPatience',10 , 'Verbose',0);

ST = datetime('now');
    netTransfer = trainNetwork( xtrain, ytrain, layers, options);

ED = datetime('now');
D = duration( ED-ST );

weights_first=netTransfer.Layers(2,1).Weights(:,:,1,1);
predictedLabels = classify(netTransfer, xtest);
accuracy = accuracyCheck( predictedLabels', ytest );

fprintf ('m[]=%f\n' , seconds(D)  );
```

8.9 CNN

```
https://www.mathworks.com/matlabcentral/fileexchange/74760-image-
   classification-using-cnn-with-multi-input-cnn?s_tid=
   srchtitle_support_results_1_CNN
```

8.10 Python TensFlow

// derative of convolution 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://en.wikipedia.org/wiki/Convolution


A
B
albo
A + B

Rozdział 9

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 LaTeXa:

- <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!

Bibliografia

[1] Józef Korbicz Andrzej Obuchowicz, D. U., *Sztuczne sieci neuronowe: podstawy i zastosowania*. Akademicka Oficyna wydawnicza PLJ, 1994, ISBN: 83-7101-197-0.

[2] Mieczysława Muraszkiewicza i Roberta Nowaka, praca zbiorowa pod redakcją, *Sztuczna inteligencja dla inżynierów*. Oficyna Wydawnicza Politechniki Warszawskiej, 2023, ISBN: 978-83-8156-584-4.

[3] R., W., *Metody programowania nieliniowego*. Warszawa: WNT, 1986.

[4] Stanisław Osowski, R. S., *Matematyczne modele uczenia maszynowego w językach MATLAB i PYTHON*. Oficyna Wydawnicza Politechniki Warszawskiej, 2023, ISBN: 978-83-8156-597-4.

[5] Stuart Russell, P. N., *Artificial Intelligence: A Modern Aproach, 4th Edition*, Grażyński, T. A., red. Pearson Education, Inc., Polish language by Helion S.A. 2023, 2023, ISBN: 978-83-283-7773-8.

[6] Y. Bengio - Université de Montréal, Y. L. .-. N. Y. U., *Convolutional Networks for Images, Speech, and Time-Series*, 1997.

Spis rysunków

Spis tablic

Spis załączników

Załącznik 1

Uogólniona reguła delty

Rozważmy sieć jednowarstwową z elementami przetwarzającymi o nieliniowej, lecz niemalejącej i różniczkowalnej funkcji aktywacji F wówczas zmianę wag przy prezentacji μ-tego wzorca można opisać równaniem:

$$\Delta w_{ji} = -\eta \frac{\partial \xi}{\partial w_{ji}} = -\eta \frac{\partial \xi}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} = -\eta \frac{\partial \xi}{\partial z_j} \frac{\partial z_j}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}}, \tag{10}$$

przy czym:

$$\frac{\partial \xi}{\partial z_j} = (s - z_j), \text{ z def. } ((\frac{1}{2}(x-y)^2)' = (x-y), \tag{11}$$

$$\frac{\partial z_j}{\partial y_j} = F'(y_j), \tag{12}$$

$$\frac{\partial y_j}{\partial w_{ji}} = x_i; \tag{13}$$

stąd ostatecznie wzór przyjmuje postać:

$$\Delta w_{ji} = \eta F'(y)(S - z_j)x_i = \eta F'(y)(S - F(y))x_i = \eta F'(y)(s_1, s_2 \dots s_m - F(x_1 * w_1, x_2 * w_2 \dots))x_i \tag{14}$$

oraz dla warstwy ukrytej:

$$\Delta w_{ji} = \eta F'(y_j) \sum_{i=1}^{n_{m+1}} F'(y_j)(s^{m+1} - z_j^{m+1}) w_{ji} \tag{15}$$

Załącznik 2

Algorytm propagacji wstecznej

Algorytm ten [1], podaje on przepis na zmianę wag w_{ij} dowolnych połączeń elementów przetwarzających rozmieszczonych w sąsiednich warstwach sieci jednokierunkowej. Jest on oparty na minimalizacji sumy kwadratów błędów uczenia z wykorzystaniem optymalizacyjnej metody największego spadku [3]. Dzięki zastosowaniu specyficznego sposobu propagowania błędów uczenia sieci powstałych na wyjściu, tzn. przesyłania ich do warstwy wyjściowej od wejściowej, algorytm propagacji wstecznej stał się jednym z najskuteczniejszych algorytmów uczenia sieci. Rozważamy sieć jednowarstwową o liniowych elementach przetwarzających. Załóżmy, że mamy P-elementowy zbiór wzorców. Przy prezentacji μ-tego wzorca możemy zdefiniować błąd:

$$\delta_j^\mu = s_j^\mu - z_j^\mu = s^\mu - y_j^\mu = s^\mu - \sum_{i=0}^{m} w_{ij} x_i^\mu, \tag{16}$$

gdzie s_j^μ, y_j^μ oznaczają odpowiednio oczekiwane i aktualne wartości wyjścia j-tego elementu oraz ważoną sumę wejść wyznaczoną w jego sumatorze przy prezentacji μ-tego wzorca. x_i^μ i-ta składowa μ-tego wektora wejściowego, w_{ji} - oznacza wagę połączenia pomiędzy j-tym elementem warstwy wyjściowej a i-tym elementem warstwy wejściowej. m-liczba wejść.
Jako miarę błędu sieci ξ wprowadzimy sumę po wszystkich wzorcach błędów powstałych przy prezentacji każdego z nich:

$$\xi = \sum_{\mu=0}^{P} \xi_\mu = \frac{1}{2} \sum_{\mu=1}^{P} \sum_{j=1}^{n} (s^\mu - y^\mu)^2, \tag{17}$$

gdzie

$$\xi_\mu = \frac{1}{2} \sum_{j=1}^{n} (s^\mu - y^\mu)^2, \tag{18}$$

Problem uczenia sieci to zagadnienie minimalizacji funkcji błędu ξ. Jedną z najprostszych metod minimalizacji jest gradientowa metoda największego spadku [3]. Jest to metoda iteracyjna, która poszukuje kolejnego lepszego punktu w kierunku przeciwnym do gradientu funkcji celu w danym punkcie. Stosując powyższą metodę do uczenia sieci, zmiana Δw_{ji} wagi połączenia winna spełniać relację:

$$\Delta w_{ji} = -\eta \frac{\partial \xi}{\partial w_{ji}} = -\eta \sum_{\mu=1}^{P} \frac{\partial \xi_\mu}{\partial w_{ji}} = -\eta \sum_{\mu=1}^{P} \frac{\partial \xi_\mu}{\partial z_j^\mu} \frac{\partial z_j^\mu}{\partial w_{ji}} \tag{19}$$

gdzie η oznacza współczynnik proporcjonalności. W przypadku elementów liniowych mamy:

$$\frac{\partial \xi_\mu}{\partial z_j^\mu} = -(s_j^\mu - z_j^\mu) = -\delta_j^\mu, \tag{20}$$

$$\frac{\partial z_j^\mu}{\partial w_{ji}} = \frac{\partial y_j^\mu}{\partial w_{ji}} = x_i^\mu \tag{21}$$

stąd otrzymujemy:

$$\Delta w_{ji} = \eta \sum_{\mu=1}^{P} \delta_j^\mu x_j^\mu \tag{22}$$

ostatecznie pełną regułę zapiszemy:

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}, \tag{23}$$

Konsekwentna realizacja metody największego spadku wymaga dokonywania zmian wag dopiero po zaprezentowaniu sieci pełnego zbioru wzorców. W praktyce stosuje się jednak zmiany wag po każdej prezentacji wzorca zgodnie ze wzorem:

$$\Delta^\mu w_{ji} = -\eta \frac{\partial \xi_\mu}{\partial w_{ji}} = \eta \delta_j^\mu x_i^\mu, \tag{24}$$

Załącznik 3

Jednowymiarowa regresja liniowa

[5] Funkcja liniowa jednej zmiennej to funkcja w postaci $y = w_1x + w_0$; współczynniki w_0 i w_1 możemy traktować jak wagi, i możemy je traktować łącznie jako wektor $\mathbf{W} =< w_0, w_1 >$ a samo przekształcenie można utożsamić z iloczynem skalarnym $y = \mathbf{W} * < 1, x >$. Zadanie dopasowania najlepszej hipotezy hw wiążącej te dwie wielkości nosi nazwę regresji liniowej. Matematycznie dopasowanie to sprowadza się do znalezienia wektora W minimalizującego funkcję straty, zgodnie z teorią Gaussa jako miarę tej straty przyjmuje się sumę miar dla wszystkich przykładów:

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, hw(x_j)) = \sum_{j=1}^{N} L_2(y_j - hw(x_j))^2 = \sum_{j=1}^{N} L_2(y_j - (w_1x + w_0))^2,$$

Naszym celem jest znalezienie optymalnego wektora W

$$\mathbf{W} = \text{argmin } Loss(h_w)$$

Gdy funkcja ciągła osiąga minimum w danym punkcie, pierwsze pochodne cząstkowe po argumentach tej funkcji zerują się w tym punkcie; w kontekście regresji liniowej nasza funkcja $Loss(h_w)$ jest funkcją dwu zmiennych: w_0 i w_1, których wartości w punkcie minimum określone są przez układ równań:

$$\begin{cases} \frac{\partial}{\partial w_0} \sum_{(} y_j - (w_1x + w_0))^2 = 0, \\ \frac{\partial}{\partial w_1} \sum (y_j - (w_1x + w_0))^2 = 0, \end{cases}$$

Rozwiązaniem takiego układu są wartości:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(x_j^2) - (\sum x_j)^2}, w_0 = \frac{\sum y_j - w_1(\sum x_j)}{N}, \tag{25}$$

Dla dużych N[5] musimy użyć następującej, równoważnej postaci rzeczonych wzorów:

$$w_1 = \frac{\sum(x_j - \overline{x})(y_j - \overline{y})}{\sum(x_j - \overline{x})^2}, w_0 = \overline{y} - w_1\overline{x}, \tag{26}$$

gdzie $\overline{x}$ i $\overline{y}$ są średnimi arytmetycznymi:

$$\overline{x} = \frac{\sum x_j}{N}, \overline{y} = \frac{\sum y_j}{N}, \tag{27}$$