# Wstęp do sztucznej inteligencji
# Sprawozdanie 5

Jan Kwiatkowski 299374
Dawid Ruciński 304212

Maj 2023

## 1 Treść zadania

W ramach piątych ćwiczeń będą Państwo musieli zaproponować architekturę, zaimplementować, wytrenować i przeprowadzić walidację sieci neuronowej do klasyfikacji ręcznie pisanych cyfr. Zbiór danych do użycia: MNIST - http://yann.lecun.com/exdb/mnist/. (zbiór ten dostępny jest też w ramach sklearn, co może być dla Państwa prostsze w pracy https://scikit-learn.org/0.19/datasets/mldata.html).

Należałoby uzyskane rezultaty porównać dla różnej liczby neuronów ukrytych i różnej liczby warstw (sprawdzić tylko kilka przypadków z uwagi na czas nauki sieci). Warto sprawdzić też wpływ współczynnika uczenia na pracę modelu. W raporcie warto opisać wygląd przykładowej sieci i przeprowadzić wstępną analizę zbioru danych. Pokazać jakie przykłady ze zbioru MNIST zostały sklasyfikowane poprawnie a jakie nie (dosłownie kilka obrazów np. że cyfry 1 i 7 są błędne albo że 2 jest wykrywane poprawnie).

## 2 Analiza zbioru danych

Badanym zbiorem jest MNIST - kolekcja monochromatycznych obrazów 28x28 przedstawiających pisane cyfry wraz z etykietami, co to są za cyfry. Obrazów jest 70 000, potem zostały one podzielone w zbiory testowy i uczący.

Dystrybucja poszczególnych klas w zbiorze prezentuje się następująco:

Tabela 1: Dystrybuca klas

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 6903 | 7877 | 6990 | 7141 | 6824 | 6313 | 6876 | 7293 | 6825 | 6958 |

## 3 Implementacja

Celem rozwiązania jest klasyfikacja zbioru obrazów przedstawiających ręcznie pisane cyfry o wymiarach 28x28 pikseli. Etykietami są cyfry przedstawione na obrazach.

Została zaimplementowana sieć neuronowa o architekture MLP (multilayer perceptron). Pierwotna implementacja zakłada wykorzystanie trzech warstw: wejściowej o rozmiarze 784 neuronów (28x28), ukrytej o rozmiarze 128 neuronów, a także wyjściowej o rozmiarze 10 neuronów. W przypadku dwóch pierwszych warstw funkcją aktywacji jest funkcja sigmoidalna dana wzorem:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

, zaś w przypadku ostatniej warstwy jest to funkcja softmax:

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2}$$

W ten sposób na wyjściu jest otrzymywany wektor liczb o rozmiarze takim, jak liczba neuronów warstwy wyjściowej. Ponieważ składowe tego wektora są unormowane, dają one w rezultacie zestaw liczb sumujących się do 1. Finalnie, wybierana jest etykieta reprezentująca najwyższą wartość.

Warstwy są w pełni połączone (fully-connected) między sobą. Oznacza to, że każdy poszczególny neuron z danej warstwy ma połączenie ze wszystkimi neuronami. Każde z nich jest scharakteryzowane w postaci jego wagi. Jest to możliwa do zmiany wartość liczbowa, przez którą mnoży się dane wejscie neuronu. W ten sposób można modyfikować odpowiedź sieci na zadane pobudzenie.

# 4 Trenowanie modelu

## 4.1 Parametry

- zbiór do trenowania,
- zbiór z danymi do przewidzenia,
- współczynnik uczenia,
- liczba i wielkość warstw,
- liczba epok.

## 4.2 Uczenie sieci

Algorytm składa się z dwóch głównych etapów: kroku w przód (metoda forward) i kroku wstecz (metoda backpropagation).

Metoda train wykonuje iteracje przez określoną liczbę epok. W każdej epoce, wykonuje metodę forward i następnie metodę wstecznej propagacji gradientu. Obliczane są metryki (accuracy oraz mse) na podstawie danych testowych, które są drukowane po każdej iteracji.

### 4.2.1 Metoda forward

Na początku, dla danego wejścia X, obliczane są wartości aktywacji dla każdej warstwy ukrytej i dla warstwy wyjściowej. Wyniki te są przechowywane w self.activations.

Dla każdej warstwy ukrytej, obliczane są wartości ‘z‘ poprzez pomnożenie aktywacji z poprzedniej warstwy przez odpowiednie wagi, a następnie dodanie biasu. Wyniki te przechowywane są w liscie ‘z_values‘ Funkcja sigmoidalna jest stosowana do obliczenia aktywacji dla danej warstwy.

Na końcu, dla warstwy wyjściowej, obliczane są wartości ‘z_out‘ poprzez pomnożenie aktywacji ostatniej warstwy przez wagi warstwy wyjściowej, a następnie dodanie biasu. Funkcja softmax jest stosowana do uzyskania prawdopodobieństw przynależności do klas.

### 4.2.2 Metoda backpropagation

Na podstawie wartości wyjściowych i oczekiwanych wartości ‘y‘, obliczane są błędy dla warstwy wyjściowej. Błędy te są mierzone za pomocą różnicy pomiędzy przewidywanymi a oczekiwanymi wartościami.

Następnie, błędy są propagowane wstecz przez warstwy ukryte. Dla każdej warstwy, błąd jest obliczany jako iloczyn błędu z poprzedniej warstwy i pochodna funkcji sigmoidalnej z wartości ‘z‘. To przekazywanie błędów pozwala na aktualizację wag w celu minimalizacji błędu.

Ostatecznie, wagi i biasy są aktualizowane na podstawie obliczonych gradientów i współczynnika uczenia (learning_rate) za pomocą wzorów:

- waga -= learning_rate * gradient
- bias -= learning_rate * gradient_bias

# 5 Eksperymenty

Eksperymenty przeprowadzono dla poniższych ustawień:

- różnych wartości współczynnika uczenia 0,1 ; 0,2 ; 0,3 ; 0,4,
- różnych ilości warstw: [784,128,10], [784,256,64,10]
- zbiór trenujący i testowy podzielony w proporcji 85% do 15%,
- 1000 epok,
- pomiar accuracy oraz mse po każdej epoce.

Tabela 2: Porównanie wyników dla sieci 3 warstwowej po 1000 epokach

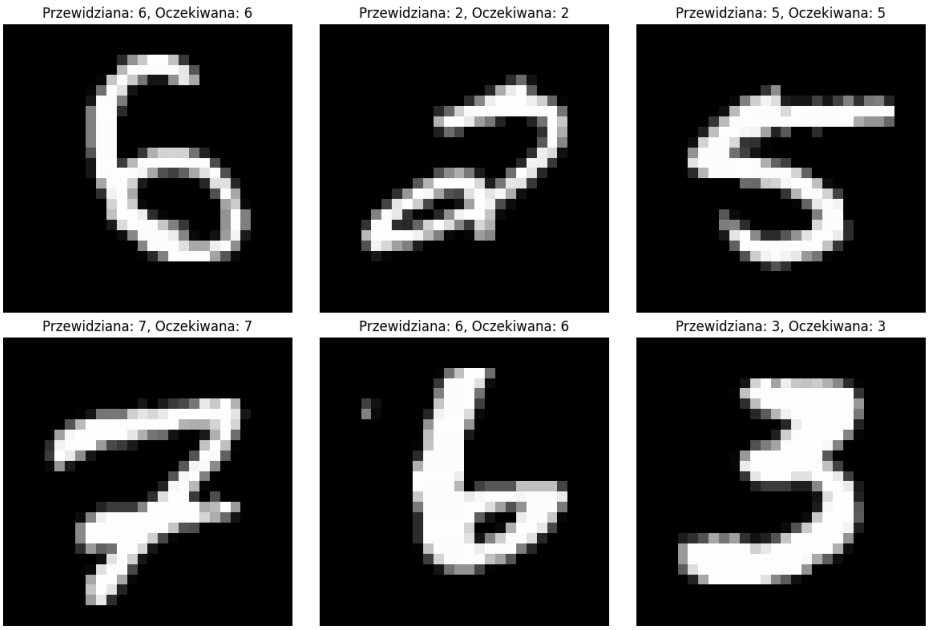
| | Eksperyment 1 | | Eksperyment 2 | |
|---|---|---|---|---|
| Współczynnik uczenia | Accuracy | Error | Accuracy | Error |
| 0,1 | 0,825 | 2,840 | 0,829 | 2.893 |
| 0,2 | 0,869 | 2,224 | 0,868 | 2.213 |
| 0,3 | 0,887 | 1,942 | 0,890 | 1.922 |
| 0,4 | 0,878 | 2,060 | 0,901 | 1,668 |

Tabela 4: Porównanie wyników dla sieci 3 warstwowej po 1000 epokach

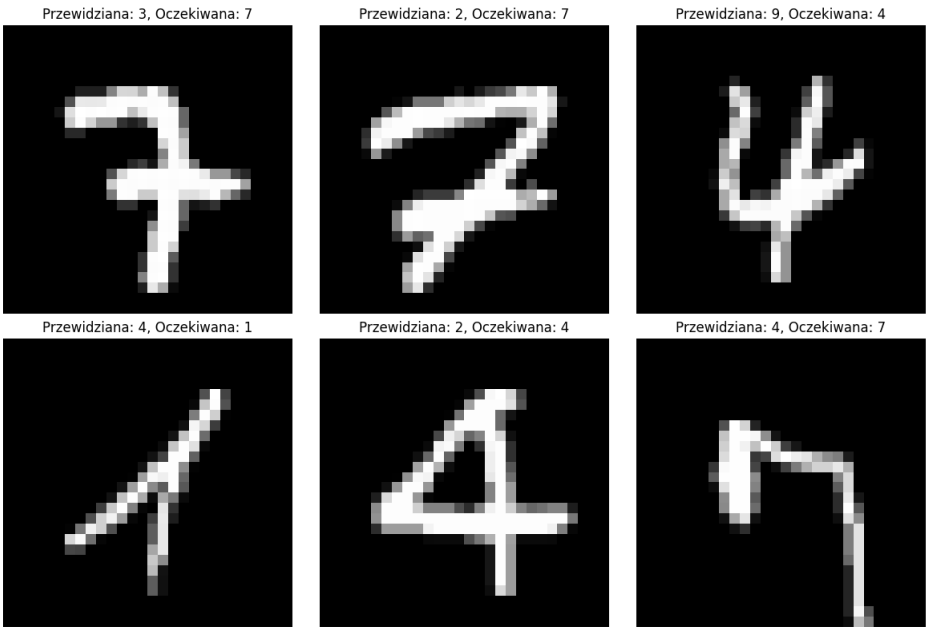
| Współczynnik uczenia | Neurony w warstwie ukrytej | Accuracy | Error |
|---|---|---|---|
| 0,1 | 64 | 0,817 | 3,021 |
| 0,2 | 64 | 0,868 | 2,122 |
| 0,3 | 64 | 0,887 | 1,850 |
| 0,4 | 64 | 0,893 | 1,792 |
| 0,1 | 128 | 0,831 | 2,784 |
| 0,2 | 128 | 0,873 | 2,119 |
| 0,3 | 128 | 0,892 | 1,762 |
| 0,4 | 128 | 0,904 | 1,592 |
| 0,1 | 256 | 0,835 | 2,787 |
| 0,2 | 256 | 0,871 | 2,119 |
| 0,3 | 256 | 0,890 | 1,765 |
| 0,4 | 256 | 0,902 | 1,532 |

Tabela 3: Porównanie wyników dla sieci 4 warstwowej po 1000 epokach

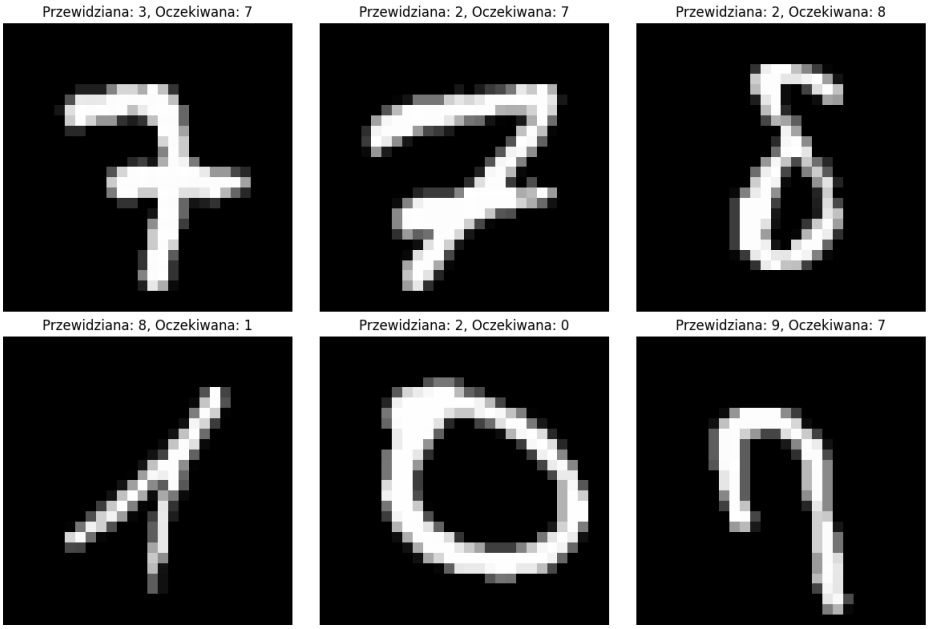
| | Eksperyment 1 | |
|---|---|---|
| Współczynnik uczenia | Accuracy | Error |
| 0,1 | 0,765 | 3,711 |
| 0,2 | 0,845 | 2,471 |
| 0,3 | 0,878 | 1,946 |
| 0,4 | 0,895 | 1,661 |



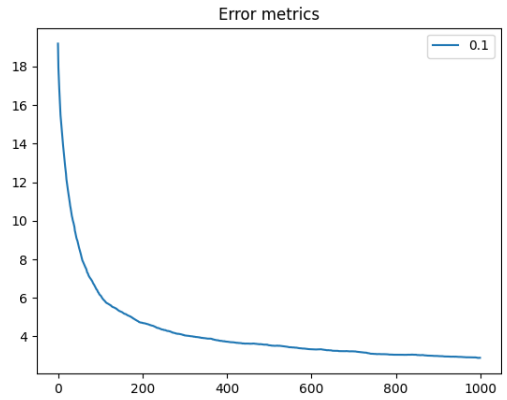

Rys. 1: Przykładowe poprawnie sklasyfikowane obrazy dla lr=0,2



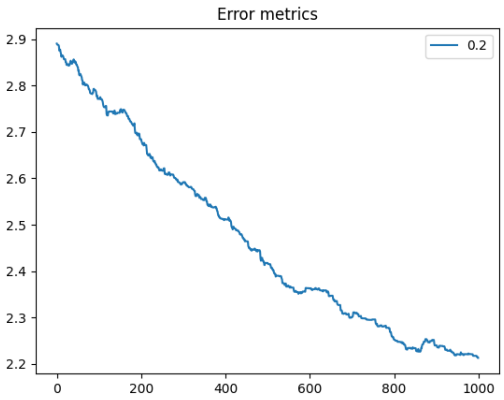

Rys. 2: Przykładowe błędnie sklasyfikowane obrazy dla lr=0,2



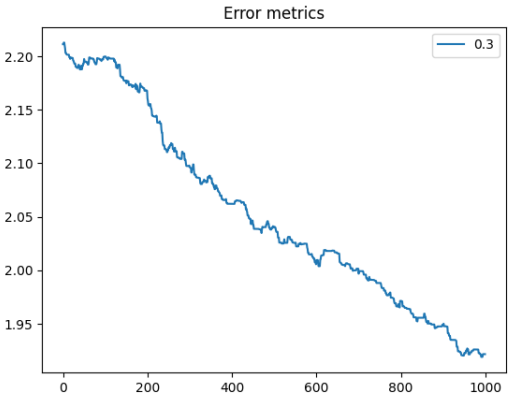

Rys. 3: Przykładowe błędnie sklasyfikowane obrazy dla lr=0,4



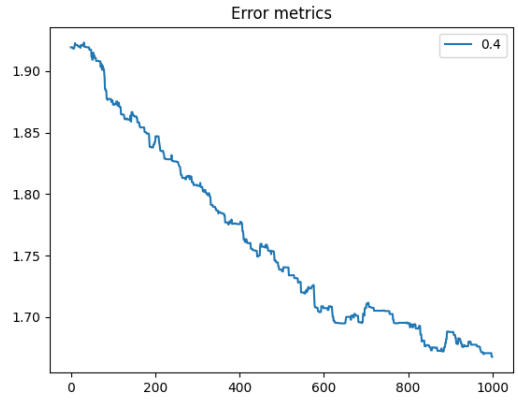

Rys. 4: MSE dla lr=0,1 i 3 warstw sieci



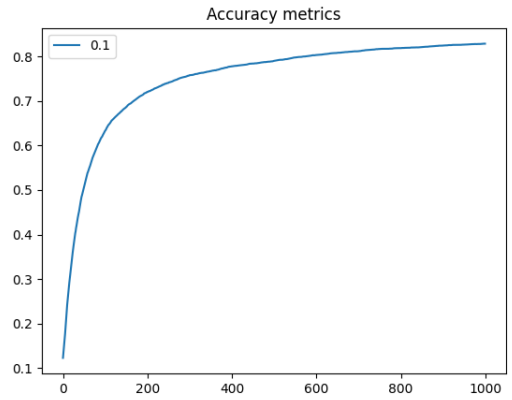

Rys. 5: MSE dla lr=0,2 i 3 warstw sieci



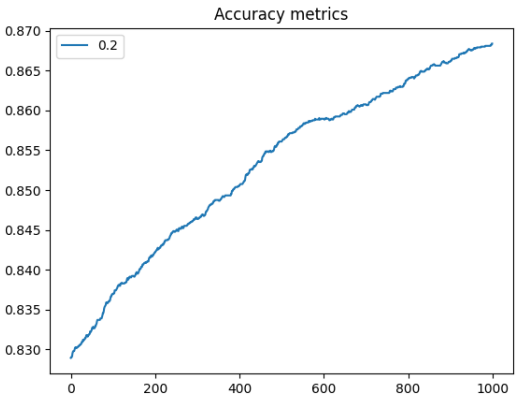

Rys. 6: MSE dla lr=0,3 i 3 warstw sieci



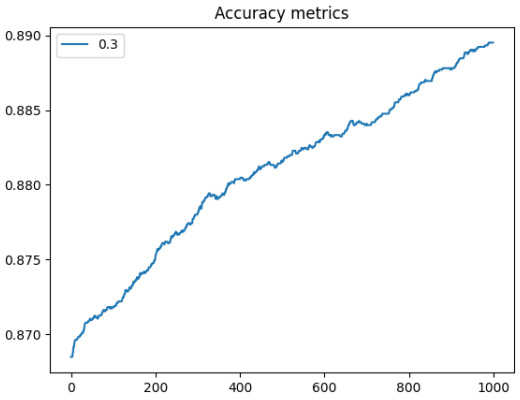

Rys. 7: MSE dla lr=0,4 i 3 warstw sieci



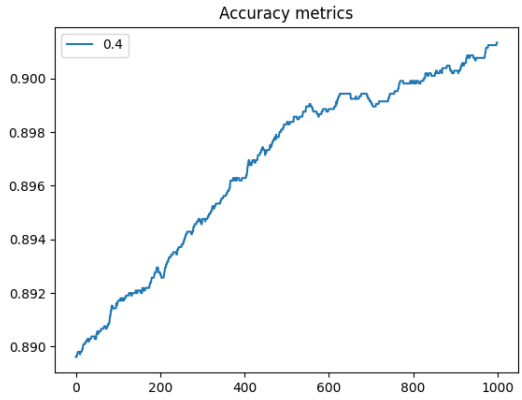

Rys. 8: Accuracy dla lr=0,1 i 3 warstw sieci



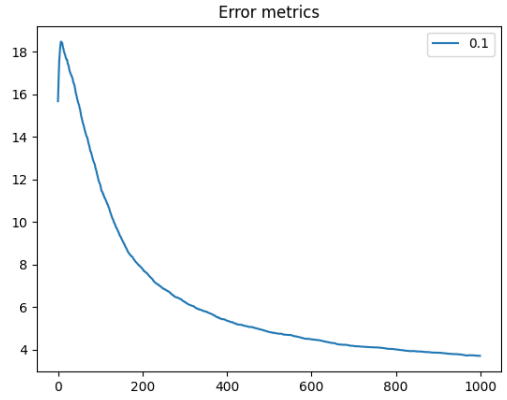

Rys. 9: Accuracy dla lr=0,2 i 3 warstw sieci



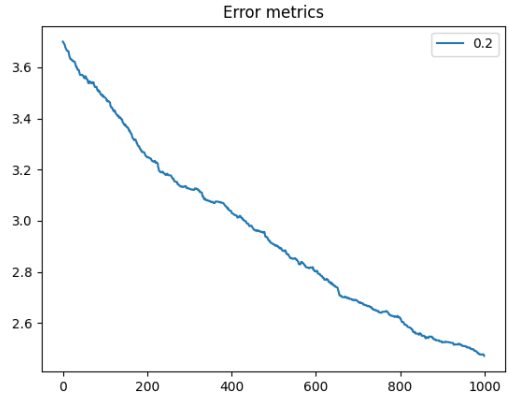

Rys. 10: Accuracy dla lr=0,3 i 3 warstw sieci



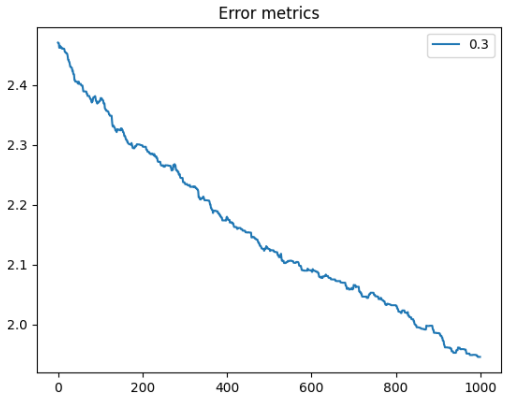

Rys. 11: Accuracy dla lr=0,4 i 3 warstw sieci



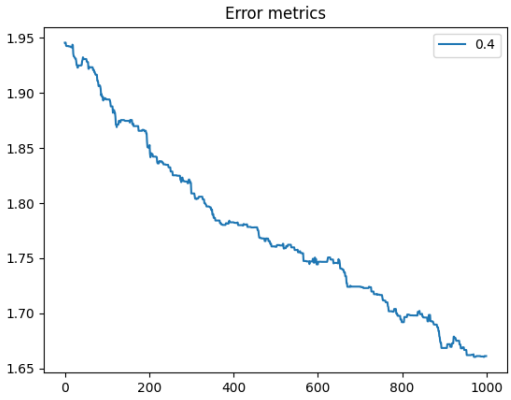

Rys. 12: MSE dla lr=0,1 i 4 warstw sieci



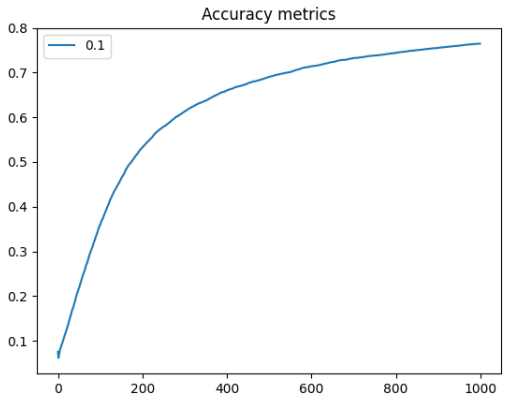

Rys. 13: MSE dla lr=0,2 i 4 warstw sieci



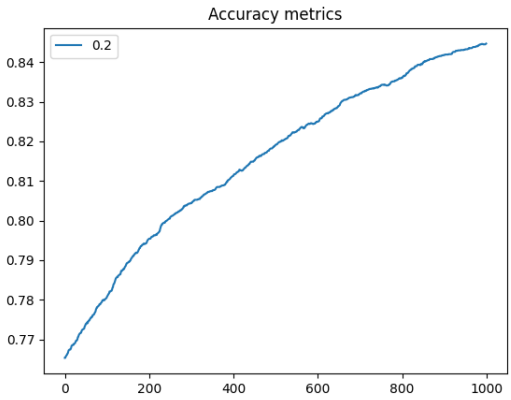

Rys. 14: MSE dla lr=0,3 i 4 warstw sieci



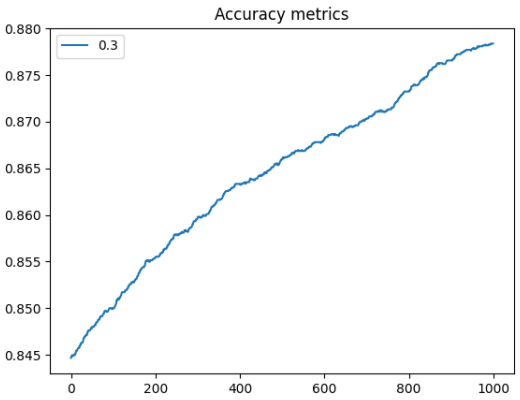

Rys. 15: MSE dla lr=0,4 i 4 warstw sieci



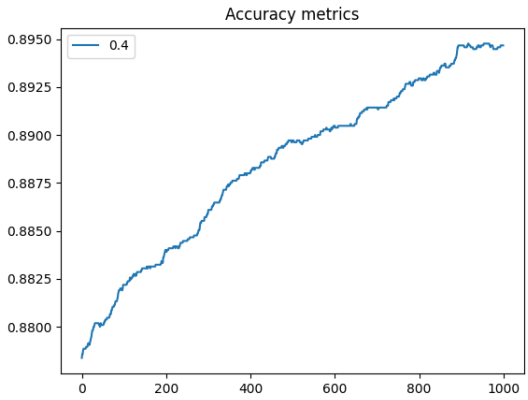

Rys. 16: Accuracy dla lr=0,1 i 4 warstw sieci




Rys. 17: Accuracy dla lr=0,2 i 4 warstw sieci




Rys. 18: Accuracy dla lr=0,3 i 4 warstw sieci




Rys. 19: Accuracy dla lr=0,4 i 4 warstw sieci

# 6 Wnioski

Wraz ze wzrostem współczynnika uczenia uczenie zachodzi szybciej co widać na przebiegach dla obu architektur sieci (Rysunki 8-11 oraz 16 - 19) . Jednakże, dla wysokich wartości przebieg uczenia jest mniej stabilny. W naszej ocenie najlepiej sprawdziła się sieć o trzech warstwach i 128 neuronach w warstwie ukrytej, jednakże różnice między wynikami są niewielkie.

W kwestii oceny wizualnej, niektóre obrazy są nieczytelne lub przypominają inne, niekompletne cyfry. Najczęściej dochodziło do pomyłek między 4 a 1, prawdopodobnie z powodu występowania ukośnej kreski, a także 3 i 7 z powodu występowania trzech poziomych kresek (Rysunek 2 oraz 3). Dochodziło też do pomyłek przy rozpoznawaniu cyfr posiadających łuki lub pętle, szczególnie, gdy były one niekompletne.

Porównując naszą implementację z siecią z biblioteki tensorflow (plik ‘tensorflowNN‘), można stwierdzić, że mimo uzyskania względnie wysokiej precyzji i małego błądu na wyjściu w stworzonej przez nas sieci gotowa implementacja działa dużo skuteczniej (accuracy na poziomie 0,9995 po 1000 epokach). Nasz program wykonuje się jednak szybciej.

# 7 Podział prac

- Dawid: Ogólny zarys projektu (sieci), eksperymenty i zbieranie wyników do pliku,
- Jan: Rysowanie wykresów i obrazów, metoda backpropagation,
- Wspólnie: Sprawozdanie i końcowe dostosowanie algorytmu.