Wprowadzenie do sztucznej inteligencji
Laboratorium 5: Sztuczne Sieci Neuronowe

Szymon Skarzyński, Łukasz Szarejko

grudzień 2021

1 Zadanie

1.1 Ogólny opis

W ramach piątych ćwiczeń będą Państwo musieli zaproponować architekturę, zaimplementować, wytrenować i przeprowadzić walidację sieci neuronowej do klasyfikacji ręcznie pisanych cyfr.

Link do zbioru danych MNIST:
http://yann.lecun.com/exdb/mnist/

2 Implementacja

2.1 Użyte oprogramowanie

- Python 3.9.7
- Numpy 1.21.3
- idx2numpy 1.2.3
- Matplotlib 3.4.3
- Tensorflow 2.7.0
- Seaborn 0.11.2
- Sklearn 1.0.1

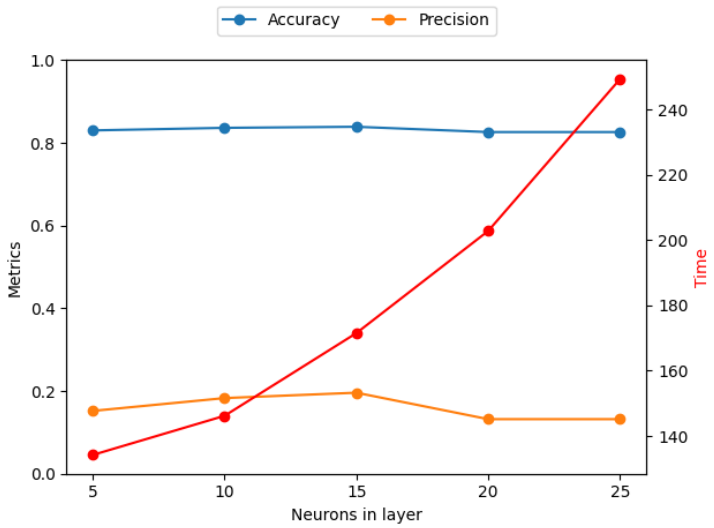
2.2 Utworzone skrypty

- data_loader.py - ładowanie i normalizacja danych
- neuron.py - implementacja neuronu
- layer.py - implementacja warstwy
- neural_network.py - implementacja sztucznej sieci neuronowej
- testing.py - testowanie sieci i generowanie wykresów
- example.py - wyświetlanie przykładowego działania sieci
- playground.py - porównanie z biblioteką Tensorflow

3 Analiza

3.1 Liczba neuronów ukrytych

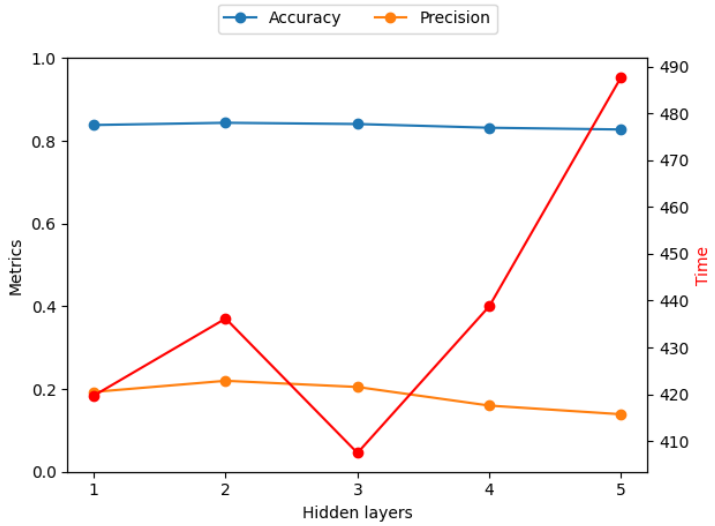
Na wstępie przebadamy jaki wpływ ma ilość neuronów w warstwach ukrytych na efektywność modelu. Założyliśmy wstępnie, że testy przeprowadzimy dla 2 warstw ukrytych, 10 epok oraz współczynnika uczenia równego 0.1. W dalszych punktach wartości będziemy stopniowo dobierali zgodnie z uzyskanymi wynikami. W związku z czasem trwania testów, wykorzystujemy tylko 10% zbioru trenującego i testowego. Warto też zaznaczyć, że sieć w warstwie wyjścia wybiera wyjście zgodnie z algorytmem softmax.



Jak widzimy na powyższym wykresie, model najlepiej spisuje się dla 15 neuronów w warstwach ukrytych, dlatego do dalszych badań będziemy brali właśnie tę wartość.

3.2 Liczba warstw ukrytych

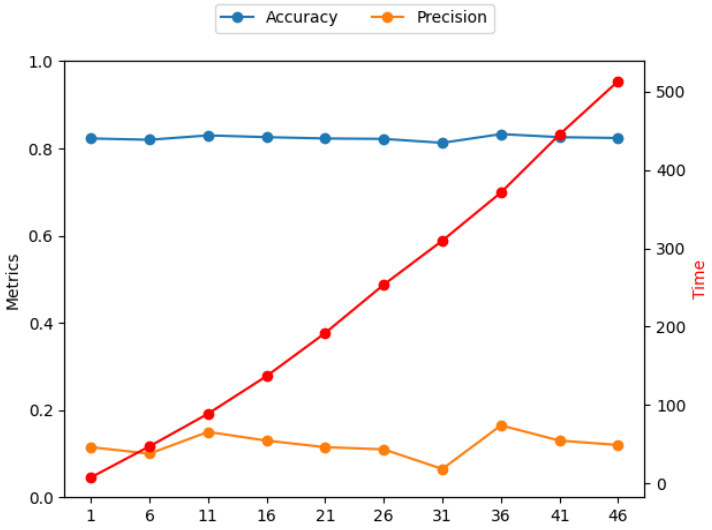
Przeprowadzimy teraz testy badające wpływ liczby warstw ukrytych na sprawność modelu.



Na powyższym wykresie widzimy, że czas wykonania dla 3 epok spadł diametralnie. Metryki jednak wskazują, że najlepszym rozwiązaniem jest wzięcie 2 warstw ukrytych. Możliwe, że tak krótki czas wykonania dla 3 warstw jest kompletnie przypadkowy, dlatego nie będzie on naszym wyznacznikiem, a w następnych punktach nasza sieć będzie miała 2 warstwy ukryte. Testowanie dla większej ilości warstw ukrytych zajęłoby zbyt dużo czasu, dlatego tego nie robimy.

3.3 Liczba epok

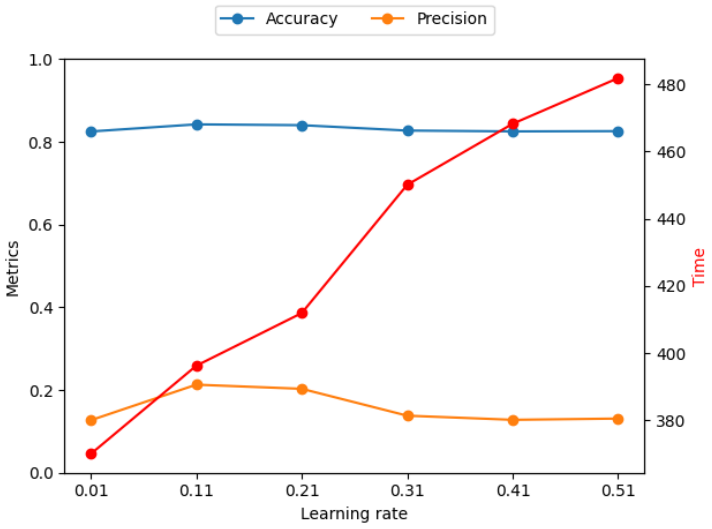
Ważnym elementem do przepadania jest też liczba epok, w których nasz model się dostraja. Poniższy wykres przedstawia wpływ ilości epok na sprawność modelu. Niestety, przez długi czas trwania testów, nie byliśmy w stanie w rozsądnym czasie przetestować sprawności dla większej liczby epok przy 10% danych, więc postanowiliśmy w tym teście zmniejszyć ilość danych do 2%.



Jak można zauważyć optymalną liczbą epok są okolice 11. W dalszych punktach wrócimy więc do liczby 10.

3.4 Współczynnik uczenia

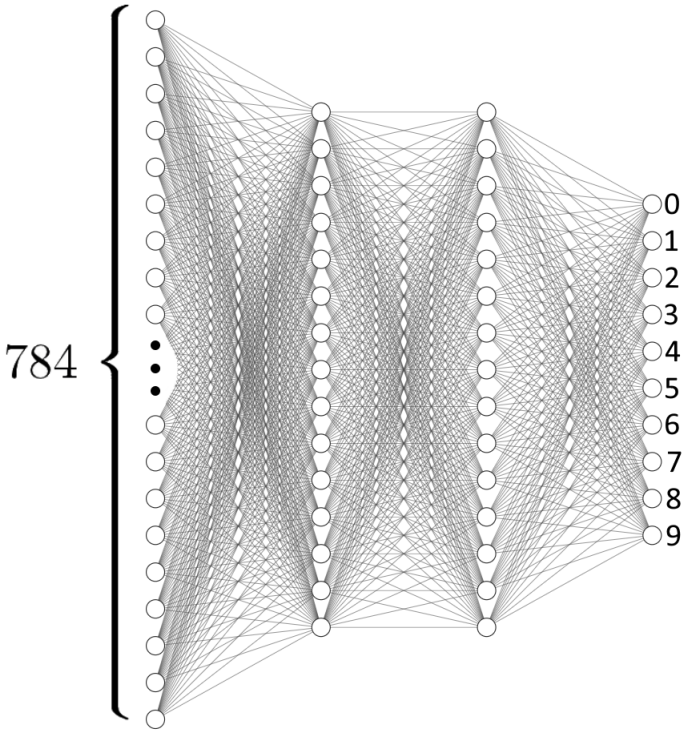
Ostatnim parametrem wymagającym dostrojenia jest współczynnik uczenia. Tutaj szczególnie ważna była górna granica testowania, gdyż przy odpowiednio dużym współczynniku, funkcja exp z biblioteki math rzucała błąd overflow, gdyż wynik operacji przekraczał możliwości floata.

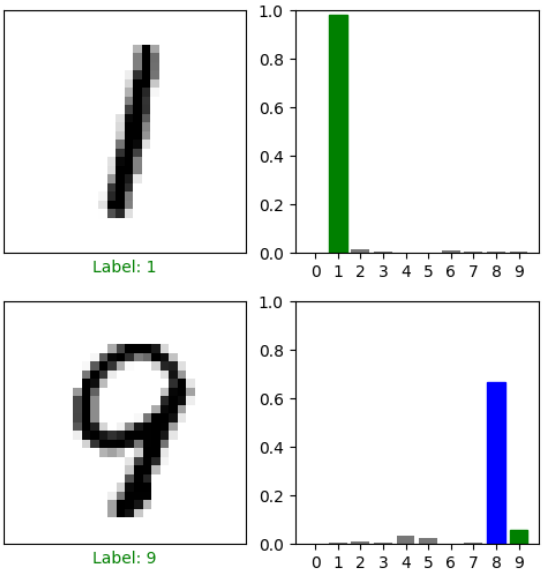


Jak możemy zauważyć model spisuje się lepiej dla niskich współczynników uczenia z zakresu 0.1 do 0.2. W związku z tym, że czas jest korzystniejszy dla dolnej granicy tego przedziału, do dalszej analizy weźmiemy właśnie 0.1.

4 Wygląd przykładowej sieci oraz przykłady sklasyfikowania

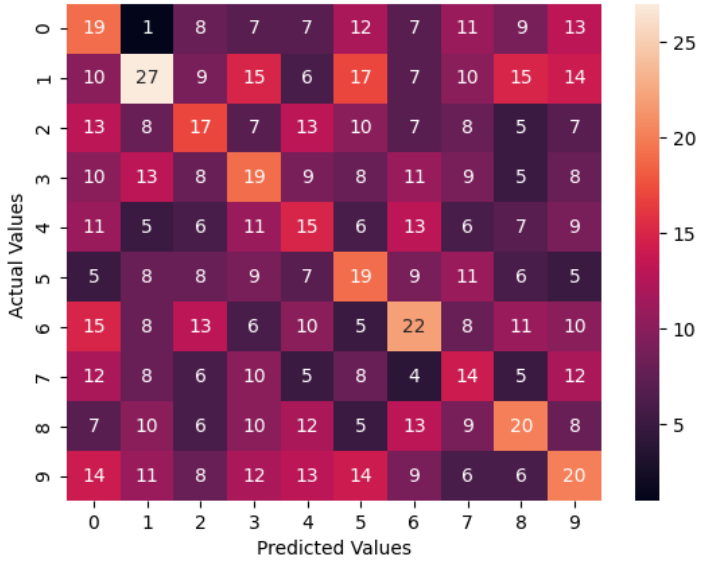
Poniżej zamieszczamy poglądowy rysunek naszej sieci oraz przykładowe wykonania, gdzie cyfra została sklasyfikowana błędnie oraz tam, gdzie poprawnie.





Zielony kolor przedstawia wartość prawdziwą, natomiast niebieski to co wybrał. Jeśli wybór pokrywa się z rzeczywistością, słupek niebieski pokryje nam się z zielonym, dlatego na wykresie z "1" nie ma słupka niebieskiego. Przedstawiony przyklad prezentuje przykładowe poprawne oraz błędne działanie sieci.

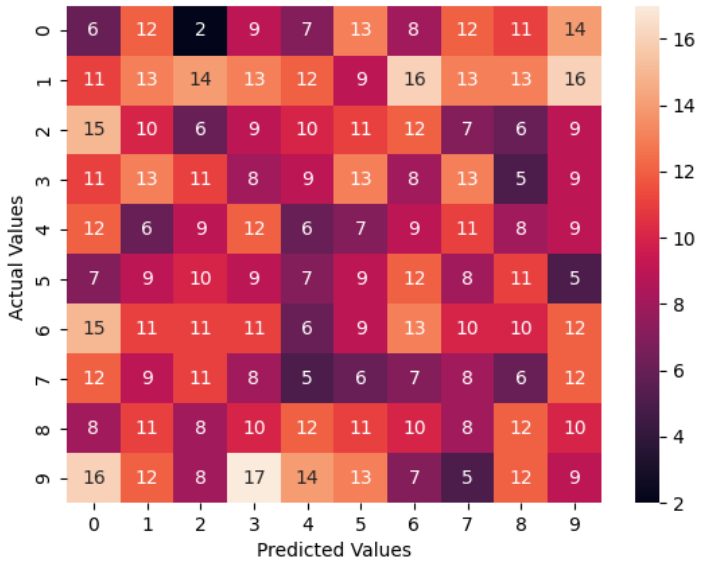
Dla ostatecznych parametrów modelu otrzymujemy taką oto macierz błędów:



Accuracy: 0.838
Precision: 0.192

5 Zainicjowanie modelu z wagami zerowymi

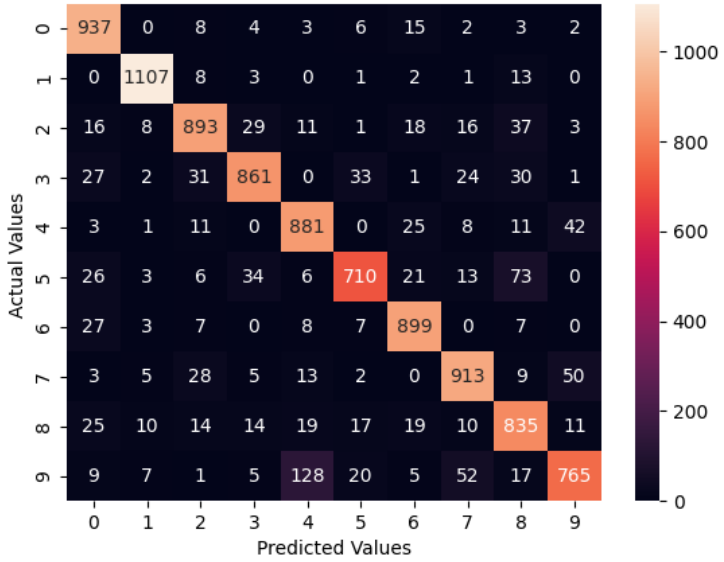
Przepadamy dodatkowo jak zachowa się model z dostrojonymi parametrami, ale przy wagach zainicjowanych wstępnie na wektor samych 0.



Accuracy: 0.818
Precision: 0.09
Jak widzimy, przez inicjalizowanie wagami zerowymi, nasza sieć ma jeszcze niższą skuteczność. Jest to spowodowane problemem zaniku gradientu, który występuje dla sieci z funkcją sigmoidalną.
Funkcja ReLU sprawdza się w takim przypadku o wiele lepie, dlatego współcześnie nie używa się raczej funkcji sigmoidalnej a funkcji ReLU lub podobnych.

6 Porównanie z Tensorflow

W celu sprawdzenia naszej sieci, porównamy wyniki jakie uzyskujemy z wynikamy uzyskanymi przez model z biblioteki tensorflow.



Accuracy: 0.88
Precision: 0.88
Jak mogliśmy się domyślić, model z biblioteki Tensorflow spisuje się o wiele lepiej. Przede wszystkim widać różnicę w precyzji modelu, jak i czasie, w którym model trenujemy i testujemy. Cały ten proces przebiegł zaledwie w kilka sekund dla dużego zbioru danych, gdzie nasza zaimplementowana sieć potrzebuje kilku minut na zaledwie 10% danych.