Wstęp do sztucznej inteligencji

Ćwiczenie 7

Sieci neuronowe

Grupa 101

Rafał Kuśmierz

2 czerwiec 2021

1. Wstęp

Rozwiązanie zadania laboratoryjnego napisane zostało w języku Python w środowisku VS Code.

2. Kod źródłowy

W folderze projektu znajdują się pliki źródłowe:

- "load_data.py" – skrypt czytający zamieszczone pliki z danymi trenującymi i testującymi
- "neuron.py" – moduł zawierający implementację klas warstw sieci neuronowej
- "network.py" - główna implementacja sieci neuronowej
- "test.py" - skrypt realizujący zadanie laboratoryjne, trenujący sieć i sprawdzający jej jakość. Generuje wykres porównujący dokładność przewidywań modelu w porównaniu z danymi na których się uczył. Przedstawia również praktyczne zastosowanie perceptronu.

Zastosowaliśmy najprostszą funkcję aktywacyjną - Rectified Linear Units **(ReLu**), która zwraca wartość 0 dla każdego argumentu mniejszego niż 0, oraz wartość argumentu dla każdego argumentu większego niż 0.:

$$f(X) = max(0, X)$$

W trenowaniu modelu pomaga pętla wstecznej propagacji poprzez dostosowanie wag w warstwie w celu zmniejszenia strat. W propagacji wstecznej aktualizacja wag odbywa się przy użyciu gradientów propagacji wstecznej, w naszym przypadku **SGD – stochastic gradient descent.**

Zamiast inicjalizacji wag neuronów losowymi, małymi wartościami, użyliśmy sposobu **inicjalizacji znormalizowanej** (**inicjalizacją Xaviera**) według rozkładu normalnego z wartością oczekiwaną *0* i odchyleniu standardowym równym [*2 / liczba neuronów wejściowych + liczba neuronów wyjściowych)*]$^{1/2}$.

Wariancja rozkładu:

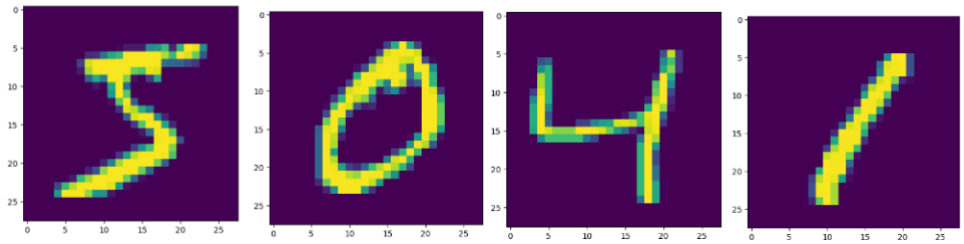
$$\mathrm{Var}(W_i) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

2.1. Uruchamianie:

Program uruchamiany poprzez uruchomienie programu “test.py” Wykonuje on 5000 cykli trenujących perceptron i tworzy wykres dokładności dopasowania co 20 cykli treningu. Następnie pokazuje testowe zastosowanie wyuczonego modelu, wyświetlając obrazy, na podstawie których określa jaka liczba jest na nich zaprezentowana.

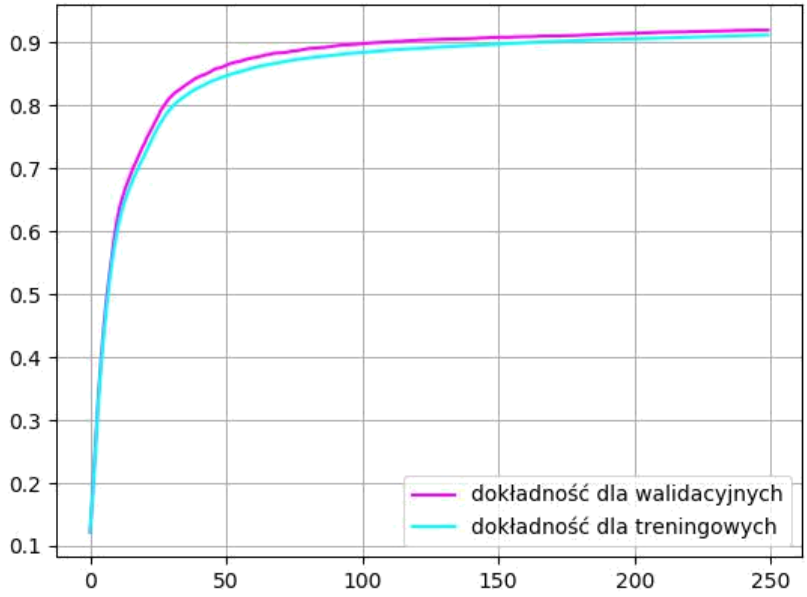
3. Badanie

Przykładowe zdjęcia które sieć uczy się rozpoznawać jako liczby:

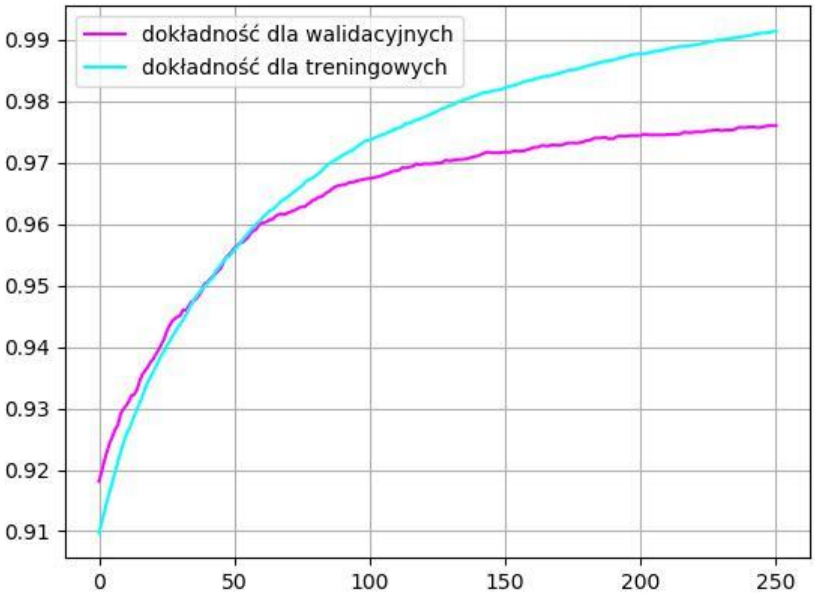


Poniżej przedstawiamy dwa wykresy. Pierwszy przedstawia stosunek skutecznych odczytów wartości z obrazków po kolejnych treningach - zaczynając od sprawdzenia przed jakimkolwiek treningiem, kończąc na 250. Po 250 treningach odczytywanie ma dokładność na poziomie około 92%. Drugi wykres przedstawia dokładność po większej liczbie treningów, od 250 do 5250. Podzieliliśmy wykresy dla podniesienia poziomu ich czytelności. Po ponad 5000 treningów skuteczność klasyfikatora wynosi już około 98%.

Badanie przeprowadzane było zarówno dla danych treningowych, jak i danych walidacyjnych. Na podstawie wykresów można zaobserwować, że sieć nie uczy się na pamięć, uczy się natomiast wymaganych zależności. Dzięki temu z każdym kolejnym treningiem lepiej rozpoznaje liczby na obrazkach. Na skuteczność nie ma natomiast wpływu, czy odczytywane obrazki były już wcześniej “widziane” przez sieć. Skuteczność prezentuje się na podobnym poziomie zarówno dla obrazków, które były wcześniej widziane, jak i tych, które nie były. Po pewnej ilości treningów to nawet na danych “nieznanych wcześniej” prezentowana była wyższa dokładność.



W drugim wykresie dla osi X każda jednostka odpowiada 20 treningom, dodatkowo analiza rozpoczyna się już po 250 przeprowadzonych treningach. Dlatego wykres rozpoczyna się od wartości, na której kończy się poprzedni. Dzięki temu otrzymaliśmy lepszą przejrzystość wykresu.



Poniższy zrzut ekranu konsoli prezentuje liczbę prawidłowych dopasowań wartości liczby do zdjęcia tej liczby dla wszystkich udostępnionych nam danych testowych - łącznie 10 000 obrazków.

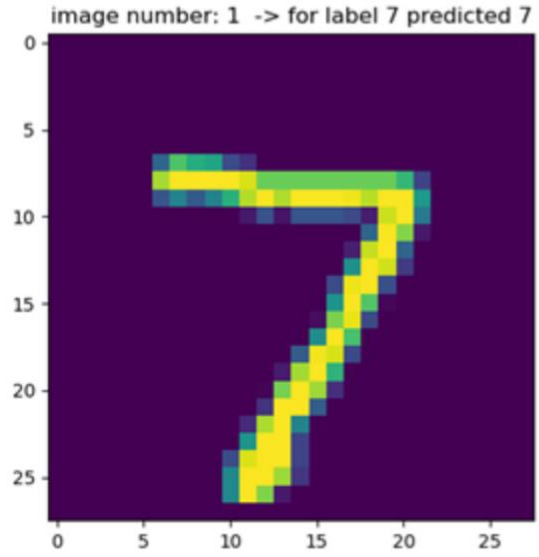
```
ilosc udanych prob -> 9774 na 10 000
```

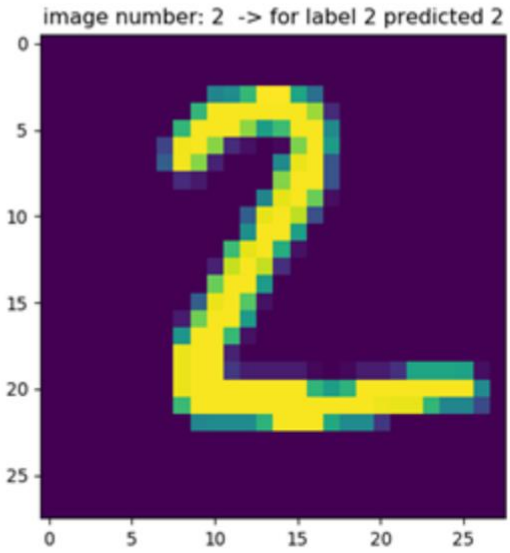
Jak widać perceptron osiągnął dokładność 97,74%.

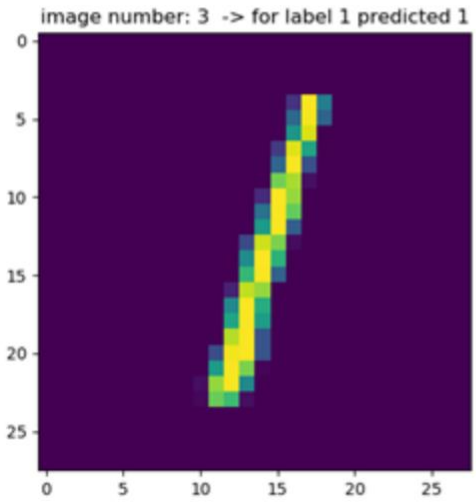
Poniżej prezentujemy użycie perceptronu wielowarstwowego.

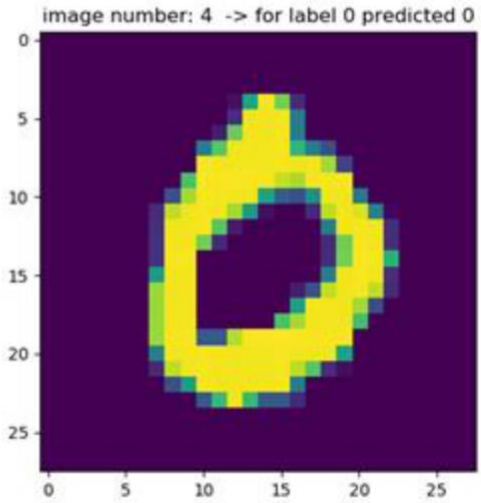
Przy użyciu poniższego fragmentu kodu wyświetlone zostały obrazki testowe (pierwsze 18), liczby jakie są na nich zapisane oraz wyniki odczytu przez nasz wytrenowany klasyfikator.

```
for i in range(18):
    pre = net.predict(network, test_images[i])
    label = test_labels[i]
    plt.title('image number: %d  -> for label %d predicted %d' % (i+1, label, pre))
    plt.imshow(test_images[i].reshape([28, 28]))
    plt.show()
```









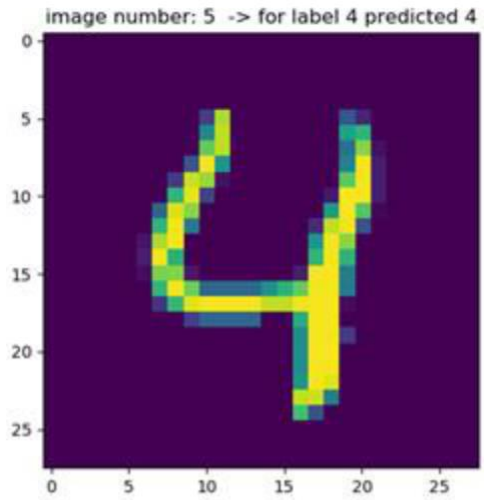


image number: 5 -> for label 4 predicted 4

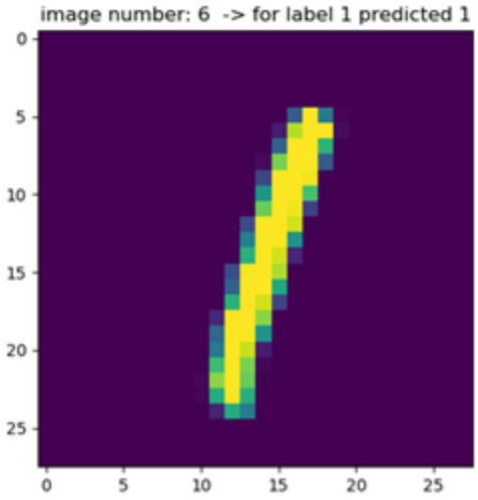


image number: 6 -> for label 1 predicted 1

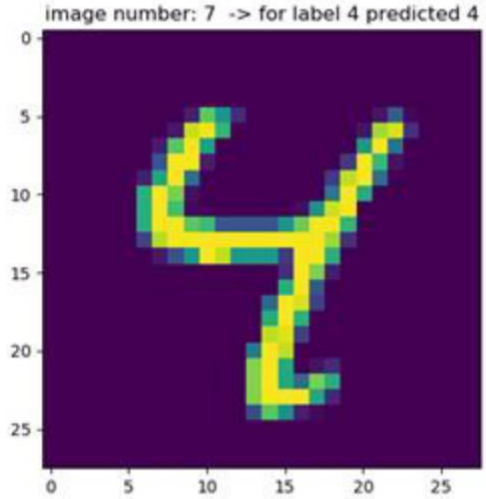


image number: 7 -> for label 4 predicted 4

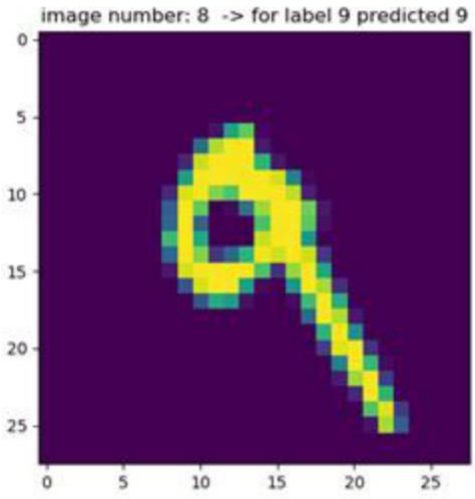


image number: 8 -> for label 9 predicted 9

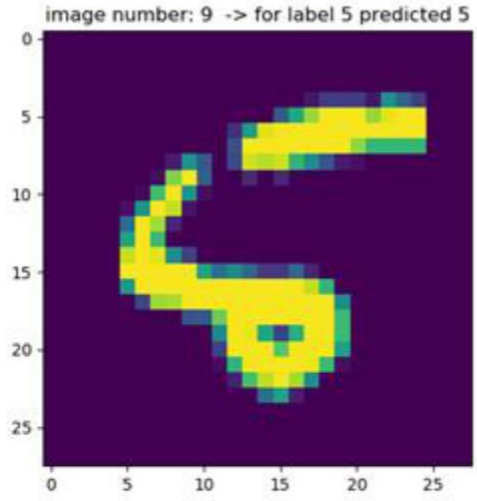


image number: 9 -> for label 5 predicted 5

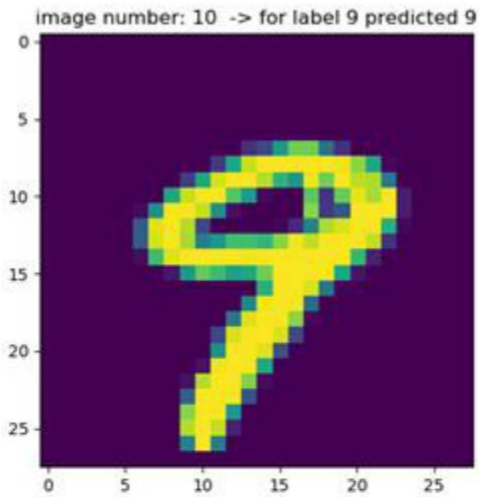


image number: 10 -> for label 9 predicted 9

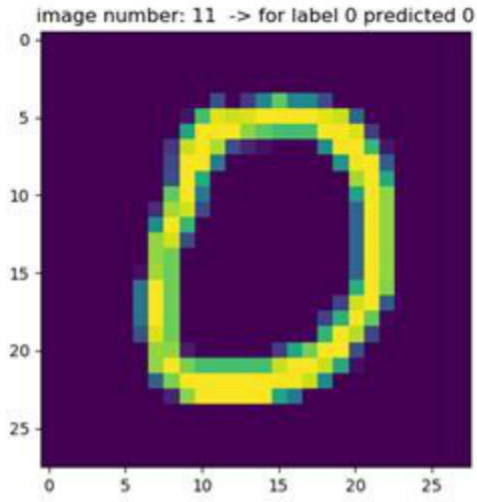


image number: 11 -> for label 0 predicted 0

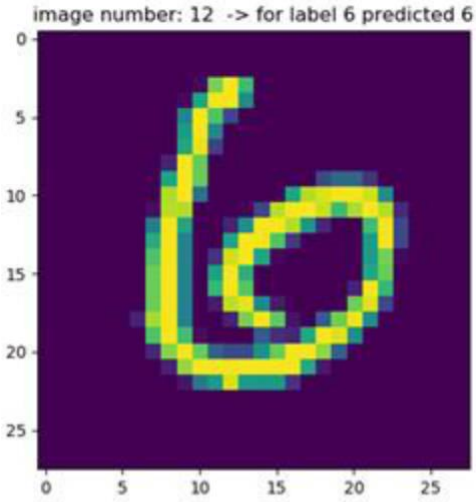


image number: 12 -> for label 6 predicted 6

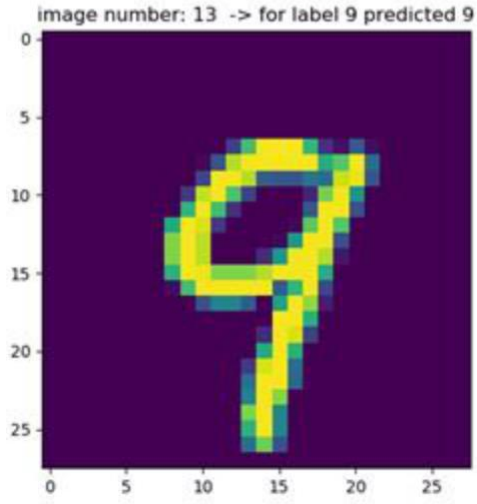


image number: 13 -> for label 9 predicted 9

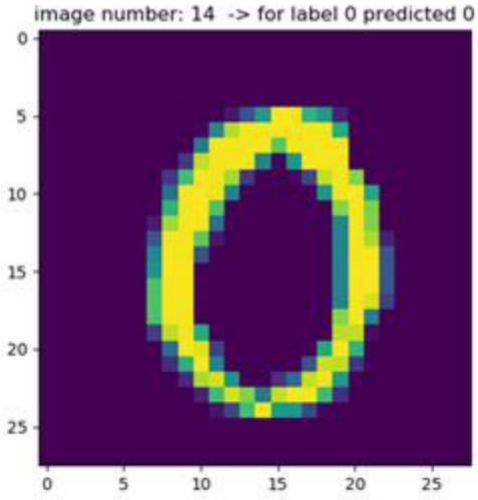


image number: 14 -> for label 0 predicted 0

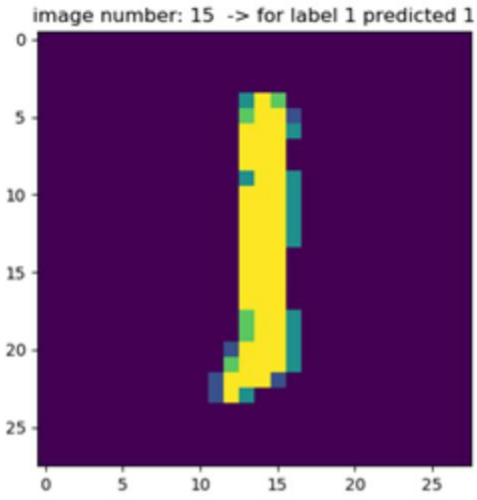


image number: 15 -> for label 1 predicted 1

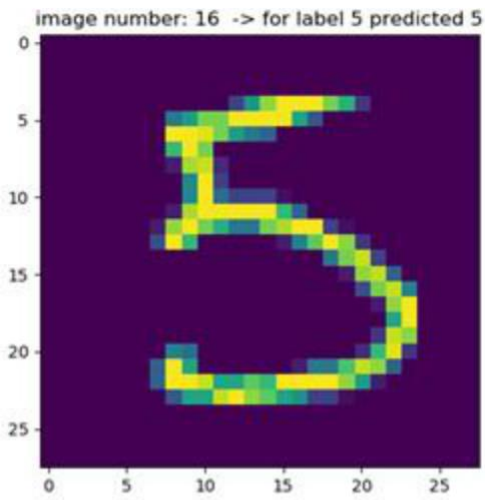


image number: 16 -> for label 5 predicted 5

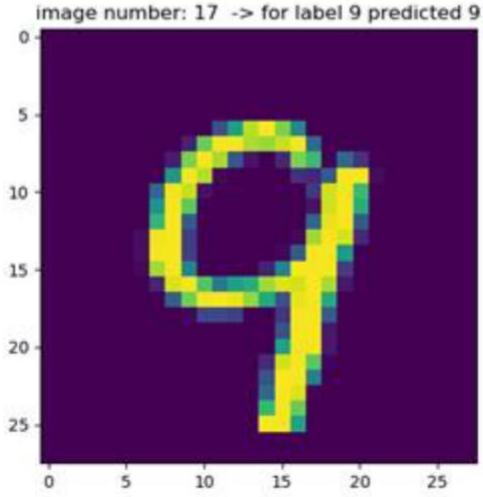


image number: 17 -> for label 9 predicted 9

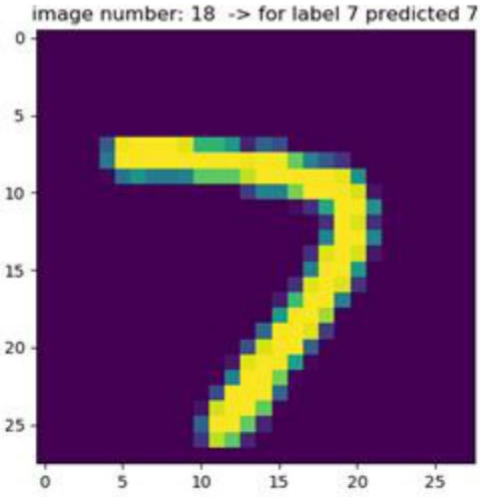


image number: 18 -> for label 7 predicted 7