Dominik Kędzior,

Inżynieria Obliczeniowa, gr. I

Nr albumu: 293094

## SPRAWOZDANIE 2

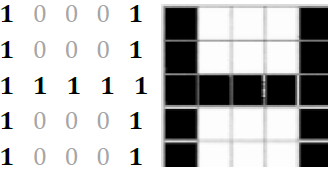
## Budowa i działanie sieci neuronowej.

Celem ćwiczenia było poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

Sieć neuronowa jest to ogólna nazwa struktur matematycznych i ich programowych lub sprzętowych modeli, realizujących obliczenia lub przetwarzanie sygnałów poprzez rzędy elementów zwanych sztucznymi neuronami, wykonujących pewną podstawową operacje na swoim wejściu.

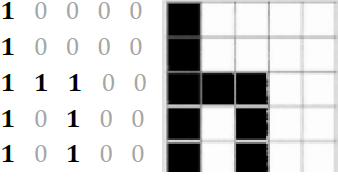
Do wykonania zadania wykorzystałem dwa rodzaje sieci neuronowych, które miały zostać nauczone rozpoznawania różnic pomiędzy dużymi oraz małymi literami. Na początku wygenerowałem dane uczące, które zawierały po dziesięć dużych oraz małych liter alfabetu w dwuwymiarowej tablicy 5x5 oraz dane testowe. Następnie za pomocą programu napisanego przy urzyciu języka Python oraz jego biblioteki neurolab wygenerowałem wcześniej wspomniane dwie jednowarstwowe sieci neuronowe, które później były uczone oraz testowane.

Umieszczanie liter w tablicy polegało na zinterpretowaniu danego pola przez sieć. Tablica składa się z 25 pól, które odpowiednio są czarne - 1, lub białe - 0. Czarne pole oznaczało, że w danym miejscu występował fragment litery, np. dla dużej i małej litery H wygląda to następująco:



Następnie dla litery H wygenerowana interpretacja jest zapisywana w postaci ciągu zer i jedynek:

**H = [1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1]**

****

Oraz dla litery h wygenerowana interpretacja jest zapisywana w postaci ciągu zer i jedynek:

**h = [1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 0 1 0 0];**

W analogiczny sposób powstała interpretacje liter A, a, B, b, C, c, D, d, E, e, F, f, I, i, K, k, L, l

które zostały zapisane i zmodyfikowane w pliku Dane.py.

### Opis wykorzystanych metod i zmiennych

**net.newp(...):**

Funkcja newp(start\_data, output\_count) tworzy jednowarstwową sieć neuronową, przyjmuje odpowiednio argumenty:

– start\_data – wartości minimalne i maksymalne dla elementów wejściowych sieci dla funkcji tworzących sieć neuronową, składa się z 25 par, które odpowiednio oznaczają wartość maksymalną - 1 oraz minimalną - 0,

– output\_count – liczba elementów wektora wyjściowego sieci, jego wartość jest równa 1 (przechowuje informacje czy dana litera jest duża, czy też nie). Można zatem traktować tę liczbę jako ilość perceptonów naszej sieci.

**net.newff(...)**

Funkcja newff(start\_data, output\_count) tworzy prostą sieć neuronową, pobiera dane wejściowe, a następnie zwraca perceptron. Jej argumentami podobnie jak w funkcji newlin są start\_data i output\_count.

**net.train(...)**

Funkcja train(input,target,epochs,show,lr) uczeniy (trenuje) sieci z wykorzystaniem danych wejściowych i wyjściowych, przyjmuje argumenty:

input – Dane uczące wejściowe

target – Dane uczące wynikowe

epochs – liczba epok

show – przedział wyświetlania na ekranie

lr – współczynnik uczenia

**train\_gd** – uczy sieć wykorzystując metodę największego spodku gradientu.

**net** - – zmienna, do której będzie przypisywana nowa tworzona sieć neutronowa,

**input** – dane uczące sieci neuronowej (litery alfabetu zapisane kolumnowo),

**target** – zmienna przechowujące dane wyjściowe odpowiadające danym uczącym (gdzie 1 odpowiada dużej literze, a 0 małej),

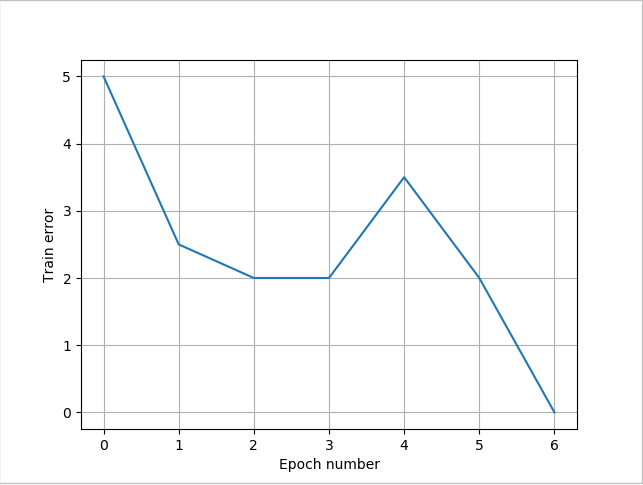
*biblioteka numpy – operacje na macierzach*

*biblioteka pyplot – przygotowanie wykresów*

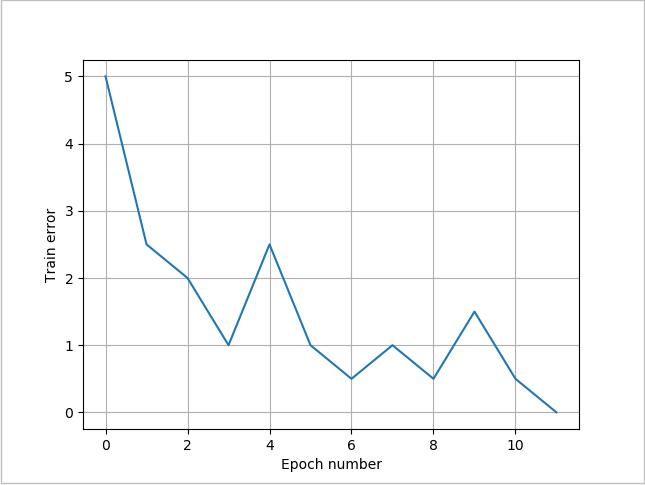
## Kilka przykładowych wyników

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Lr = 0.1 | | Lr = 0.01 | | Lr = 0.001 | |
|  | newp | newff | newp | newff | newp | newff |
| ‘A’ | 1.0 | 0.96292 | 1.0 | 0.94204 | 1.0 | 0.88769 |
| ‘b’ | 0.0 | 0.04131 | 0.0 | 0.07442 | 0.0 | 0.02136 |
| ‘C’ | 1.0 | 0.94154 | 1.0 | 0.88140 | 1.0 | 0.92922 |
| ‘d’ | 0.0 | 0.38561 | 0.0 | 0.07546 | 0.0 | 0.11352 |

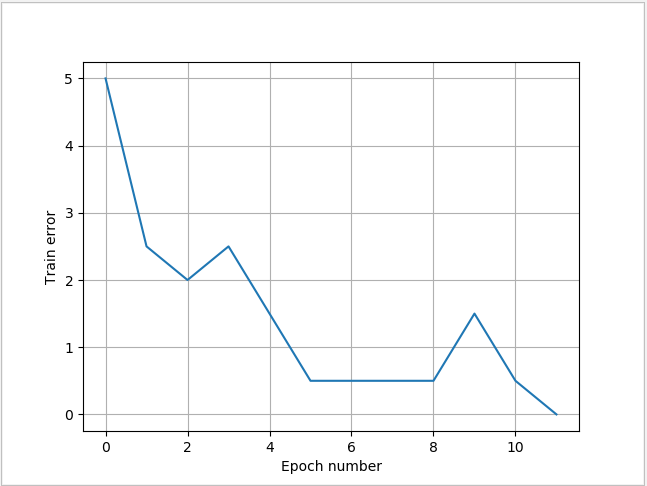
## Kilka Przykładowych wykresów zależności błędu od liczby epok



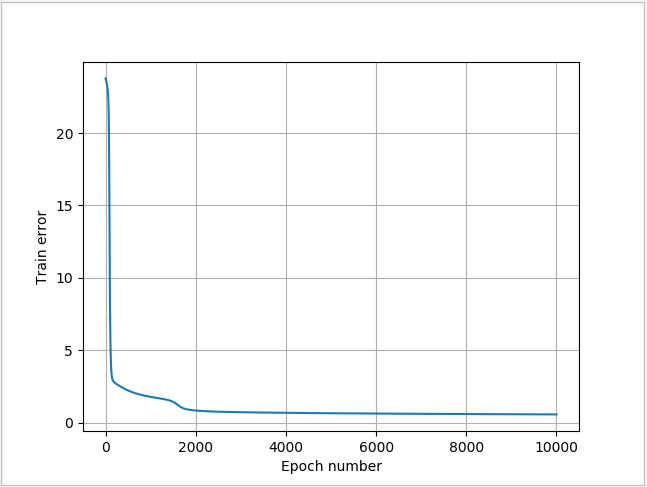
Rysunek 1. newp(epoch = 1000, lr =0.1)



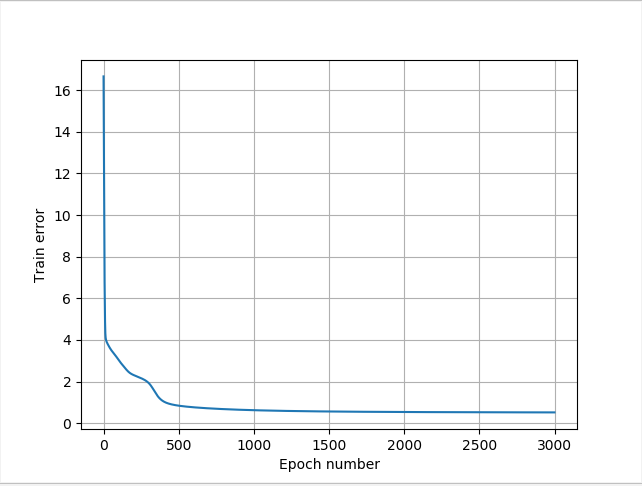
Rysunek 2. newp(epochs = 10000, lr = 0.01)



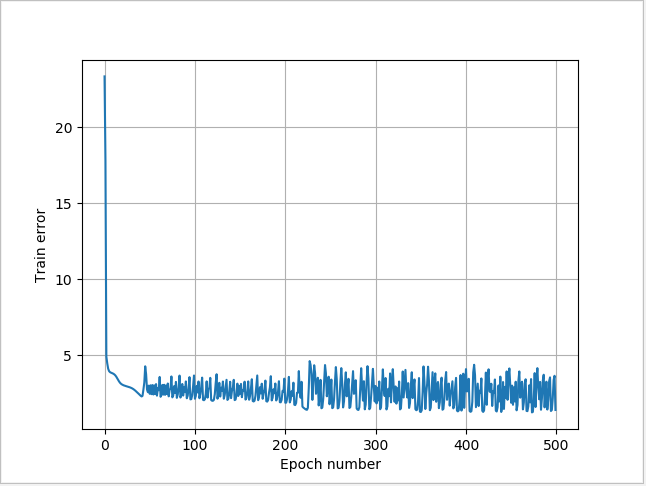
Rysunek 3. newp(epochs = 500, lr = 0.001)



Rysunek 4. newff(epochs = 10000, lr = 0.001)



Rysunek 5. newff(epochs = 3000, lr = 0.01)



Rysunek 6. newff(epochs =500, lr = 0.1)

## Wnioski

– Taela pokazuje, że współczynnik uczenia miał wpływ na dokładność przy rozpoznawaniu wielkości badanych liter - im mniejsza wartość współczynnika, tym mniejsza trafność w rozpoznawaniu wielkości liter (ale również skutkowało to mniejszą ilością potrzebnych epok do uczenia sieci). Co prawda różnice widać tylko w przypadku pierwszej metody - druga wypisywała konkretne wartości.

– Funkcja newp dawała lepsze efekty w porównaniu do newlin - do uczenia sieci potrzebowała zdecydowanie mniejszą liczbę epok (co miało wpływ na długość działania programu, im mniejsza liczba epok, tym program działał szybciej). Ponadto analizując powyższą tabelę można zauważyć, że newp posiadała wartości równe 1 lub 0, natomiast funkcja newff dawała zdecydowanie mniej klarowne wyniki (takie wyniki mogły zwiększyć ilość występowania błędów).

- Obydwie metody nie generowały błędów jeśli test danej litery zawierał się w ciągu danych uczących. W momencie wyboru litery spoza przedziału danych uczących algorytmy zaczęły je źle interpretować, zazwyczaj odwrotnie dużą literę jako małą albo myliły podobne do siebie litery (jeżeli np. podano literę „B”, gdzie ta litera została wcześniej wykreślona z danych uczących program interpretował ją jako „D”).

– Wysoki współczynnik uczenia znacznie przyspiesza tempo uczenia sieci, ale należy brać pod uwagę, że w tym przypadku dana metoda może stracić na precyzji działania, co skutkuje większą ilością generowanych błędów.

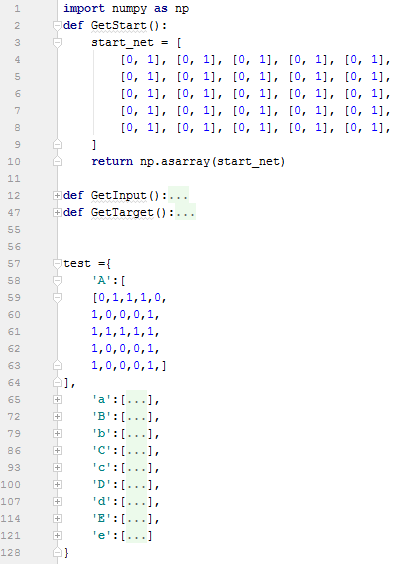
## Skrócony kod programu:



Rysunek 7. Small\_or\_capital\_letter\_using\_mewp.py



Rysunek 8. small\_or\_capital\_letter\_using\_newff.py



Rysunek 9. Data.py