Podstawy sieci neuronowych

Sprawozdanie z wariantu I

Autor: Bartłomiej Gościniak 264190

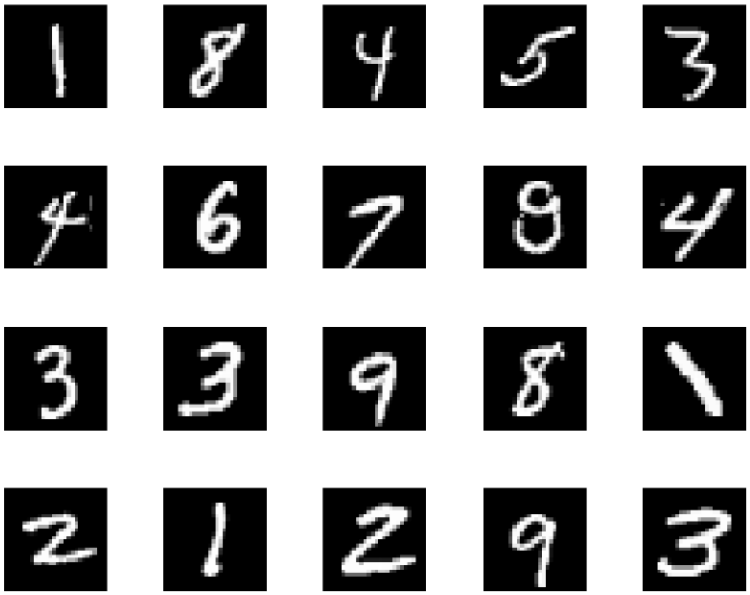
Grupa: Wtorek 17:05

Spis treści

1. Klasyfikacja

1.1 Opis zadania

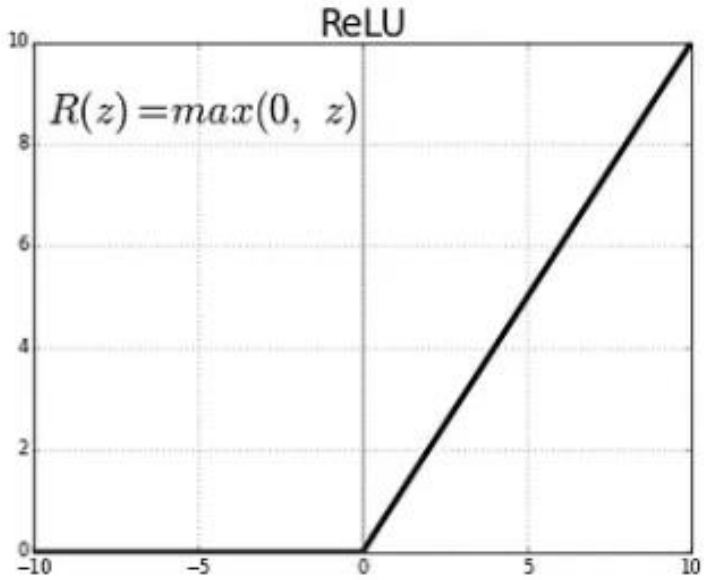
Zadanie polegało na stworzeniu sieci neuronowej będącej w stanie klasyfikować liczby pisane odręcznie z bazy MINST. Sieć została napisana w Pythonie wykorzystując bibliotekę NumPy.



Rys. 1 Przykład liczb z bazy MINST

1.2 Budowa sieci

Do wykonania zadania wykorzystano dwu-warstwową sieć, gdzie obie warstwy mają dziesięć neuronów. Na wejściu jest plik csv w którym każda linijka ma początku podpis jaką to liczbą a następnie 784 liczby odpowiadające kolorowi danego pikselu (0 – czarny, 255 – biały). Same dane wczytywane są z wykorzystaniem biblioteki Pandas. Samych liczb jest 60 tysięcy, tysiąc z nich jest wybrane do zestawu testowego a reszta jest wykorzystywana jako zestaw uczący. Same dane po wczytaniu są losowo ustawiane. Pierwsza warstwa sieci korzysta z funkcji aktywacji ReLU, druga korzysta z funkcji Softmax która pozwala na zamianę liczb na prawdopodobieństwa jaka to jest liczba. Na wyjściu jest 10 elementowy wektor wskazujący prawdopodobieństwo jaka to jest cyfra według sieci.



Rys. 2 Wykres funkcji ReLU

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Rys. 3 Wzór funkcji softmax

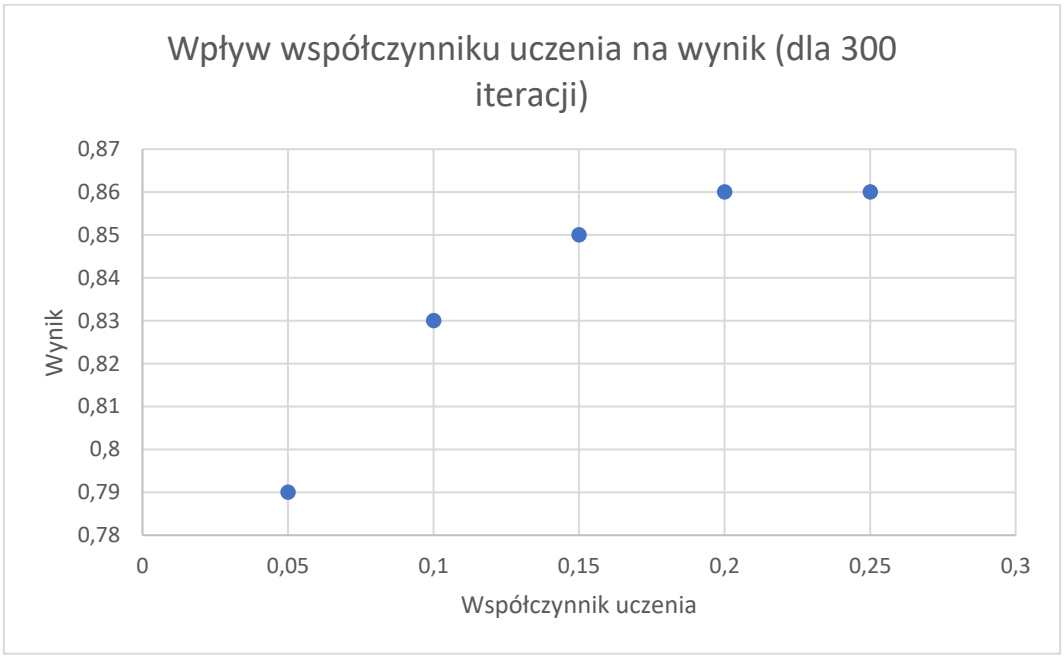
1.3 Testy sieci

W ramach testów sieci zmieniano liczbę iteracji oraz zmieniano współczynnik uczenia. Wypróbowano także kilka innych funkcji aktywacyjnych w pierwszej warstwie.

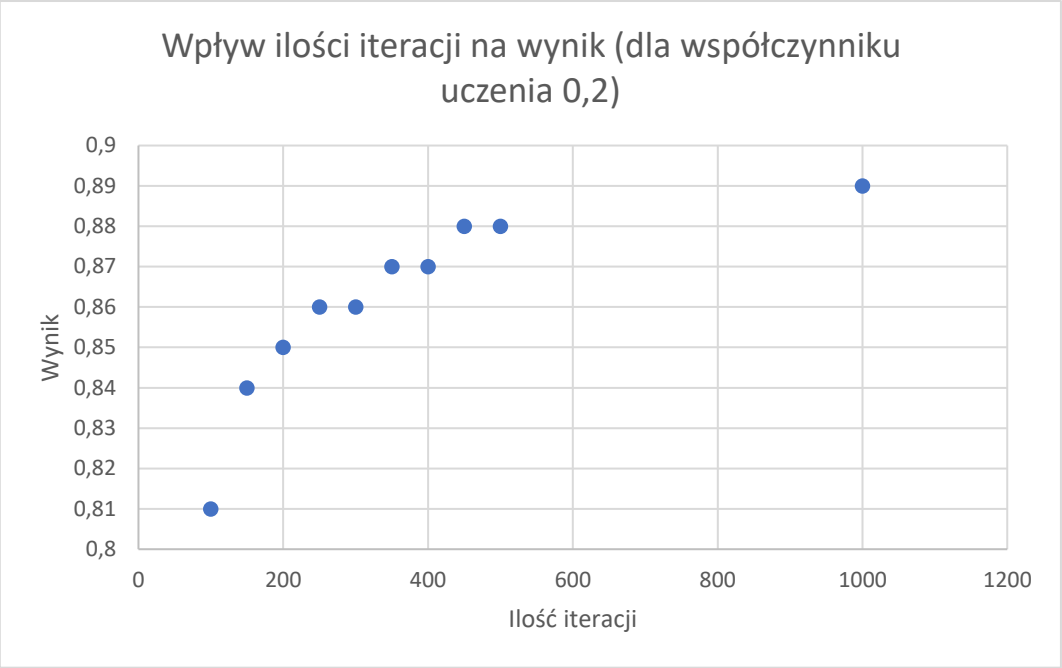
		współczynnik uczenia								
		0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
liczba iteracji	100	0,37	0,76	0,76	0,8	0,83	0,83	0,83	0,85	0,72
	200	0,77	0,83	0,86	0,85	0,87	0,88	0,87	0,87	0,88
	300	0,81	0,85	0,87	0,89	0,89	0,89	0,9	0,9	0,89
	400	0,8	0,88	0,88	0,86	0,89	0,9	0,91	0,91	0,91
	500	0,84	0,88	0,88	0,9	0,9	0,91	0,91	0,92	0,91
	600	0,86	0,88	0,9	0,91	0,9	0,91	0,91	0,92	0,91
	700	0,87	0,89	0,91	0,91	0,9	0,92	0,91	0,93	0,92
	800	0,85	0,89	0,91	0,91	0,92	0,92	0,92	0,93	0,92
	900	0,87	0,9	0,91	0,92	0,92	0,92	0,92	0,92	0,94
	1000	0,88	0,89	0,91	0,92	0,92	0,92	0,86	0,93	0,93

Rys. 4 Wyniki dla ReLU w zależności od wybranych parametrów

Testy przeprowadzono także wykorzystując funkcję liniową



Rys. 5 Wykres wpływu współczynnika na wynik dla funkcji liniowej



Rys. 6 Wykres wpływu ilości na iteracji na wynik dla funkcji liniowej

Jak widać powyżej, sieć neuronowa posiada zbliżone wyniki dla obu funkcji aktywacyjnych. Jednak funkcja ReLU osiągnęło trochę wyższe wyniki co może być spowodowane inicjacją parametrów albo lepszym działaniem funkcji w tym problemie.

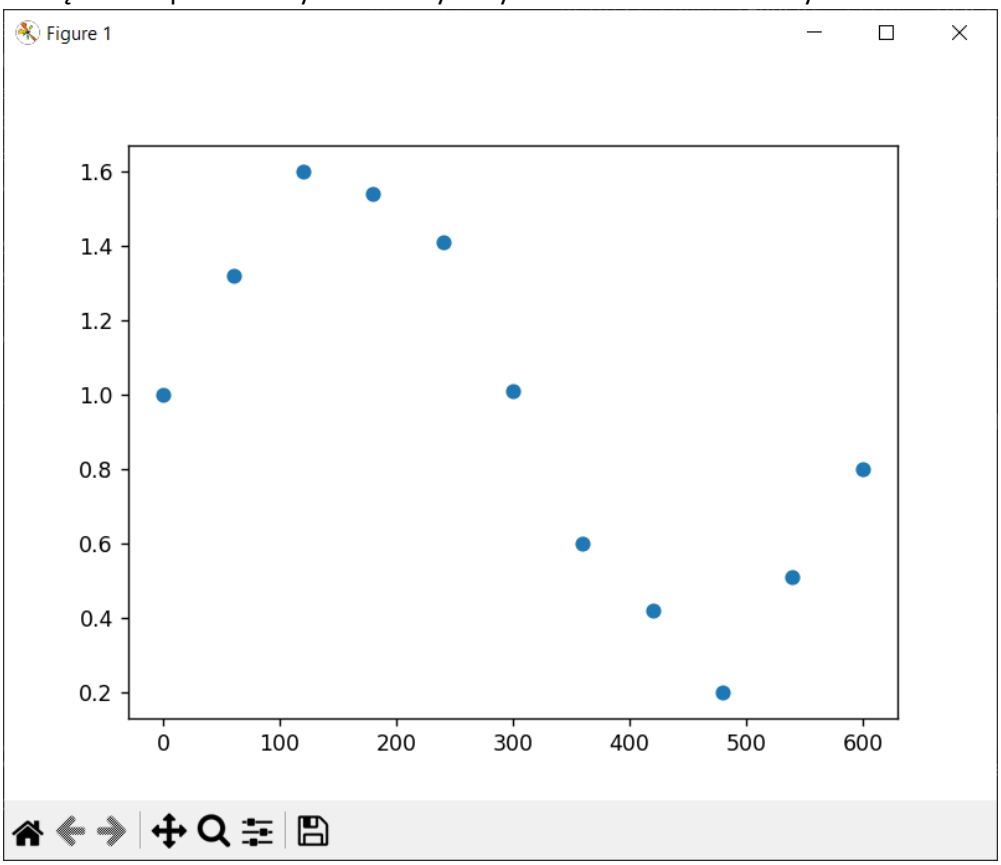
1.4 Wnioski

Sieć osiągała skuteczność w okolicach 85% niezależnie od wybranej funkcji aktywacyjnej. Ważne dla skutecznego działania sieci jest dobór odpowiednich parametrów (liczby iteracji oraz współczynnik uczenia), inaczej sieć będzie osiągała bardzo niskie wyniki. Ważne jest też posiadanie dobrego zestawu testowego, takiego którego sieć nie wykorzystywała w czasie procesu uczenia, żeby sprawdzić czy rzeczywiście potrafi poprawnie klasyfikować cyfry których jeszcze nie „widziała”.

2. Aproksymacja

2.1 Opis zadania

Zadanie polegały na aproksymacji funkcji która przedstawiała pomiar poziomu wody w morzu co godzinę. Sieć napisano w Pythonie z wykorzystaniem biblioteki NumPy.



Rys. 7 Funkcja pomiaru poziomu wody w morzu

2.2 Budowa sieci

Do aproksymacji wykorzystano dwu-warstwową sieć, po 5 neuronów w każdej warstwie. Na wejściu jest wektor z wszystkimi punktami dla których chcemy aproksymować wartość funkcji, a na wyjściu jest wektor zawierający wartość funkcji dla każdego elementu w wektorze wejściowym. Dla obu warstw wykorzystywano funkcję aktywacyjną ReLU.

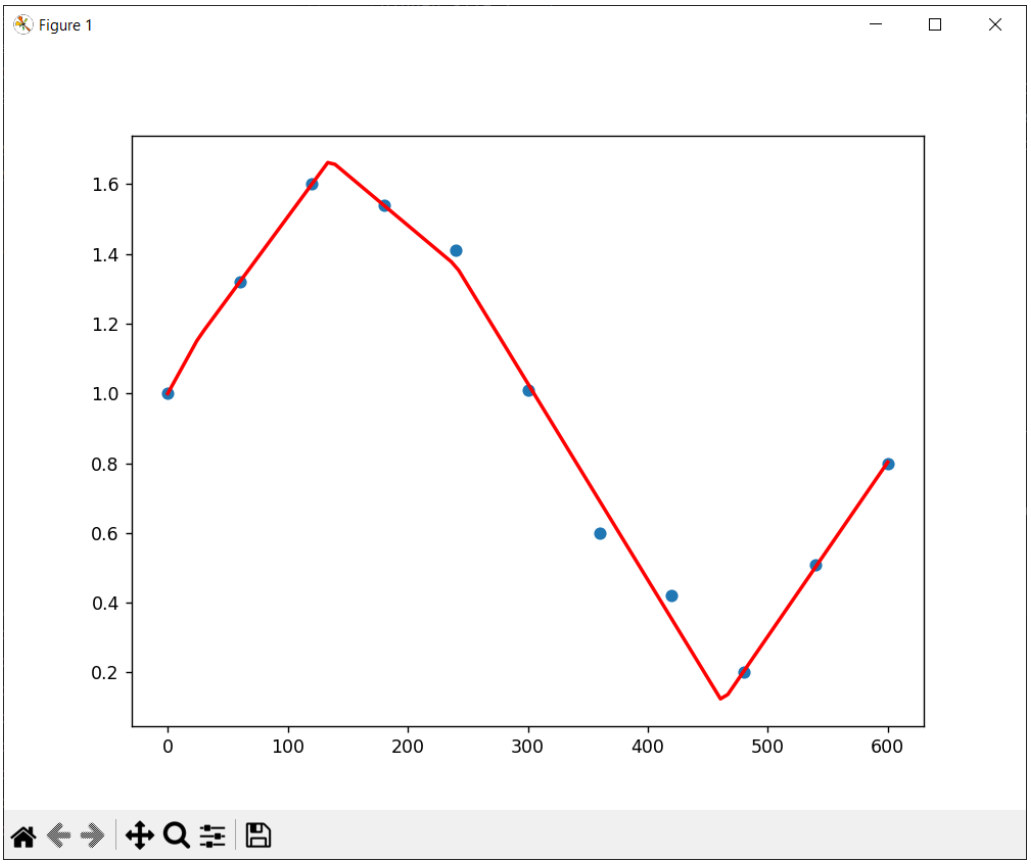
```
x_data = np.array([0, 60, 120, 180, 240, 300, 360, 420, 480, 540, 600], dtype="float64")
y_data = np.array([1.0, 1.32, 1.6, 1.54, 1.41, 1.01, 0.6, 0.42, 0.2, 0.51, 0.8], dtype="float64")

skala_x = np.max(x_data)
skala_y = np.max(y_data)

x_data /= skala_x
y_data /= skala_y
```

Rys. 8 Dane wykorzystane do nauczenia sieci

Wartości x są podane w minutach żeby łatwiej było wykonać następne testy w których należało aproksymować wartości w odstępach co 6 minut. Same dane są też skalowane żeby poprawnie wykonać testy dla funkcji sigmoidalnej która ma zakres do 1 a jak można zobaczyć dane, y wykracza poza tą wartość. W tym celu podzielono każdą liczbę przez największą wartość dzięki czemu wszystkie wartości nie przekraczają jedynki.

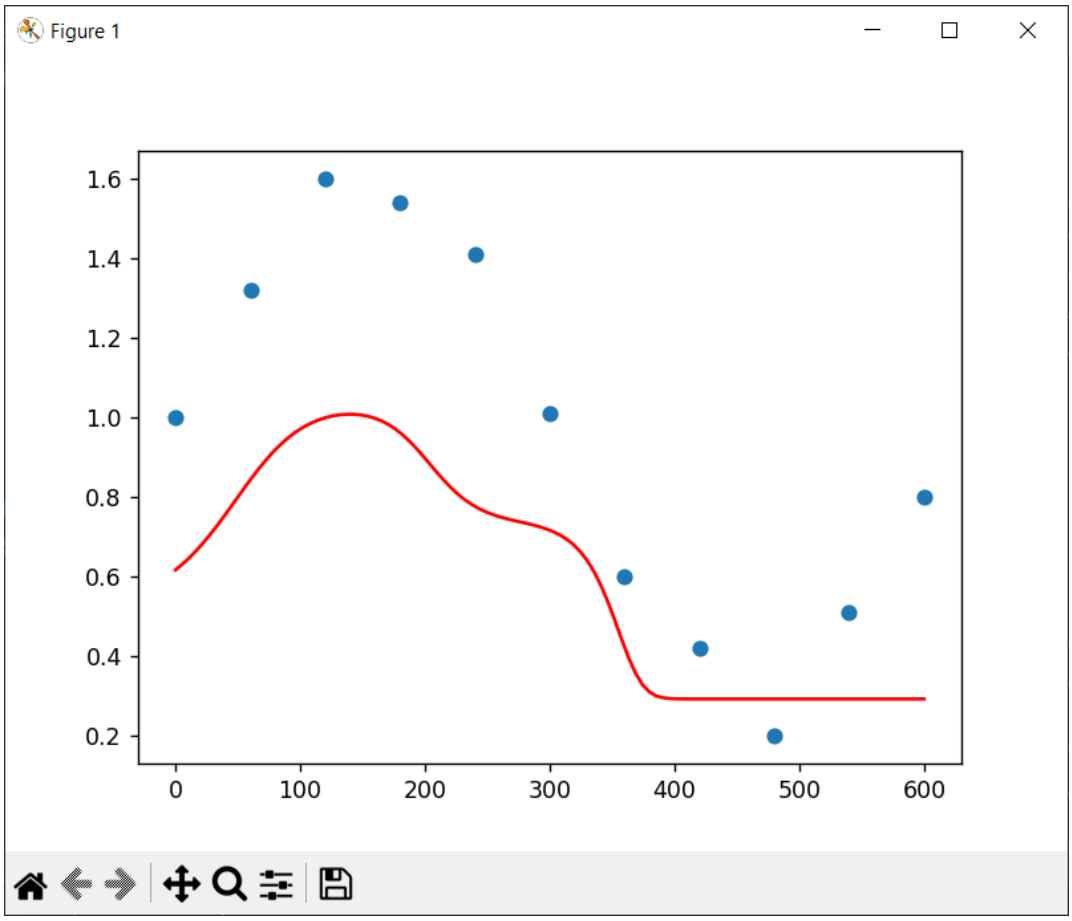
2.3 Testy

Do testów sieci wykorzystano poza funkcją ReLU wykorzystano funkcję sigmoidalną w celu złagodzenia przebiegu funkcji, niestety wyniki były gorsze niż z wykorzystaniem funkcji ReLU.



Rys. 9 Aproksymacja funkcji z wykorzystaniem ReLU

Jak widać na powyższym wykresie, sieć całkiem dobrze aproksymuje z wykorzystaniem ReLU oraz odpowiednim doborem parametrów. Funkcja jest ostra co jest spowodowane charakterystyką funkcji aktywacyjnej. W celu poprawy aproksymacji spróbowano użyć innych funkcji aktywacji, sigmoidalnej unipolarnej oraz progowej unipolarnej.



Rys. 10 Aproksymacja z wykorzystaniem funkcji sigmoidalnej

Jak widać na powyższym wykresie, wykorzystując funkcję sigmoidalną, aproksymacja jest łagodniejsza ale jest o wiele mniej dokładna. Podobne wyniki uzyskano wykorzystując funkcję progową unipolarną.

2.4 Wnioski

Jak można zobaczyć w poprzednim punkcie najlepsza aproksymacja została wykonana z wykorzystaniem funkcji ReLU. Działanie pozostałych funkcji aktywacji jest o wiele gorsze, mimo skalowania danych. Może to być spowodowane błędami w implementacji sieci, podawanie na wejściu jednego rozmiaru a następnie w ramach testów podawanie wektora innego rozmiaru też mógłby powodować problemy. Sama sieć do poprawnego działa i nauczenia się potrzebowała bardzo dużo iteracji, około pół miliona, co też może wskazywać pewne błędy w implementacji.