Politechnika Wrocławska

Politechnika Wrocławska

Sieci neuronowe
Raport z projektu 1

Author:
Łukasz Sokołowski,
Aleksander Pajorski

22 stycznia 2024

Spis treści

Spis rysunków

1 Zadanie 1

1.1 Opis zadania

Zadanie pierwsze polegało na zaprojektowaniu wielowarstwowej sieci neuronowej rozwiązującej prosty problem klasyfikacji polegający na zidentyfikowaniu odręcznie napisanej cyfry na zdjęciu w skali szarości o wymiarach 28px x 28px. Do tego zadania wykorzystano obrazy pochodzące ze zbiorów MNIST, stworzonych z myślą o tworzeniu i projektowaniu klasyfikujacych sieci neuronowych.

1.2 Dane

Tak jak wspomniano wcześniej, dane do projektowania oraz późniejszego testowania sieci zostały zaciągnięte ze zbiorów MNIST. Zdjęcia z odręczenie narysowanymi cyframi w skali szarości reprezentowane są przez jeden wiersz w pliku o rozszerzeniu *.csv*, gdzie pierwszy znak opisuje cyfrę przedstawioną na zdjęciu, a każde kolejne 28 wartości z zakresu od 0 do 255 (wartości te są skalowane tak aby przyjmowały wartości (0, 1), zanim zostaną przekazane do sieci) reprezentują kolejny rząd pixeli zdjęcia. Każdy wiersz znaków w tym pliku z osobna reprezentuje jedno przykładowe zdjęcie. Zestawy danych prezentuje się następujaco:

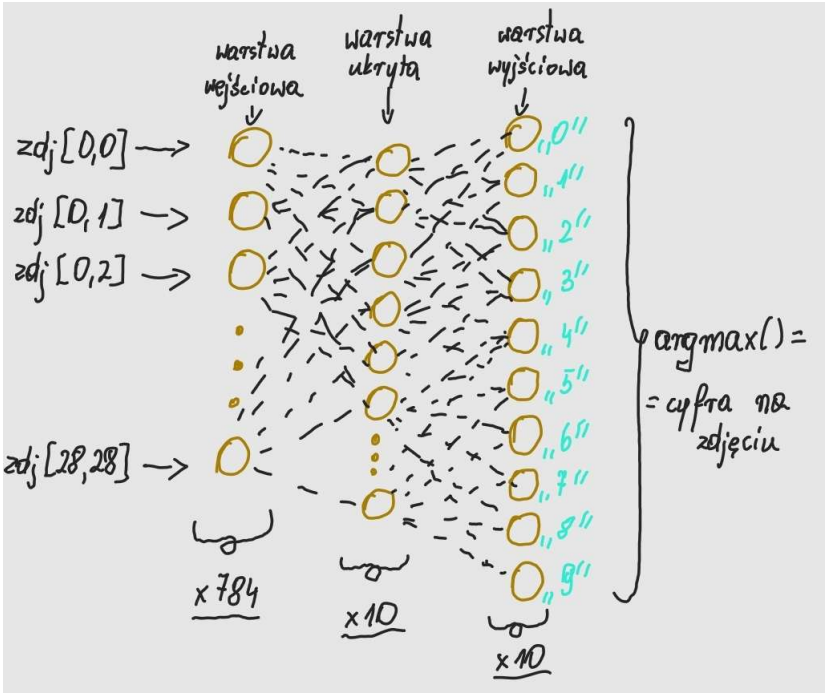
- Zestaw danych uczących, które jednocześnie w trakcie uczenia są danymi walidującymi - 40 tysięcy przykładowych zdjęć,
- Zestaw danych testowych - 10 tysięcy przykładowych zdjęć.

Zestaw danych uczących oraz zestaw danych testowych to dwa osobne zbiory,
w których kolejność przykładów jest losowo zmieniana,
przed każdym uruchomieniem uczenia się sieci oraz jej testowania.

Zaimplemenotwana sieć neuronowa w trakcie uczenia się, co określoną liczbę epok, sprawdza w ilu przypadkach propagacji do przodu klasyfikacja sieci zgadza się z opisem jaką liczbę przedstawia dane zdjęcie. Wynik w postaci ułamka wyświetlany jest na bieżąco w konsoli dzięki czemu można nadzorować proces uczenia się sieci.

1.3 Etap projektowania sieci

Do poprawnego zaprojektowania sieci należało w pierwszej kolejności ustalić ilość neuronów w warstwie wejściowej. Sieć przyjmuje obrazy o rozmiarze 28x28 pikseli, co daje 784 wejścia (jeden piksel na jedno wejście). Przy rozmiarze warstw ukrytych zdecydowano się na jedną warstwą ukrytą, z 10 neuronami. W problemie klasyfikacji, gdzie rozpoznajemy cyfry od 0 do 9, użyto 10 neuronów w warstwie wyjściowej tak, że każdy odpowiada konkretnej cyfrze.
Funkcje aktywacji należy dobrać odpowiednio dla problemu, jaki sieć ma rozwiązać. W tym przypadku wykorzystano funkcję aktywacji $ReLU$ dla warstw ukrytych i funkcję *softmax* dla warstwy wyjściowej. Wybór funkcji *softmax* jako funkcję aktywacji warstwy wyjściowej sprawia, że sieć na wyjściowych neuronach zwraca wartość prawdopodobieństwa z jaką uważa, że dana liczba znajduje się na zdjęciu.



Rysunek 1: Początkowy szkic projektowanej sieci.

1.4 Implementacja wybranych fragmentów sieci

Wagi, w postaci macierzy, zostały zainicjalizowane przy użyciu funkcji *np.random.rand()* i opisują wagi połączeń pomiędzy warstwami. Wartości *bias*, podobnie jak wartości wag, na samym początku inicjalizowane są jako wartości losowe.
Proces uczenia składa się z dwóch etapów: propagacji do przodu (forward propagation) oraz propagacji wstecznej (backward propagation).
Propagacja do przodu polega na dostarczeniu zdjęcia na wejście sieci i uzyskaniu decyzji. Przy użyciu macierzy wag i biasów, dla każdej warstwy obliczamy wartości na kolejnych neuronach, a ta przekazywana jest poźniej jako argument funkcji $ReLU$ (dla warstw ukrytych) i *softmax* (dla warstwy wyjściowej).
Propagacja wsteczna zajmuje się aktualizacją wag i biasów na podstawie uzyskanych wartości błędu. Wykorzystuje ona metodę spadku gradientu, umożliwiając korektę parametrów sieci w kierunku minimalizacji funkcji kosztu.
Jako funkcję kosztu wybraliśmy *Średni Błąd Kwadratowy* (MSE), który jest powszechnie używany w problemach regresji oraz klasyfikacji, szczególnie w przypadku problemu predykcji cyfr na zbiorze danych MNIST. W funkcji kosztu MSE, dla każdego przykładu treningowego, obliczamy kwadrat różnicy pomiędzy wartością przewidzianą, a rzeczywistą. Następnie sumujemy te kwadraty dla wszystkich przykładów treningowych i uśredniamy, co daje nam ostateczną wartość funkcji kosztu. Wykorzystane MSE jest zdefiniowane wzorem:

$$\text{MSE} = \frac{1}{K_2}\sum_{j_2=1}^{K_2}[t_{j_2} - f(\sum_{j_1=0}^{K_1} w_{j_2,j_1}^{(2)} f(\sum_{i=0}^{N} w_{j_1,i}^{(2)} x_i^{(1)}))]$$

gdzie:

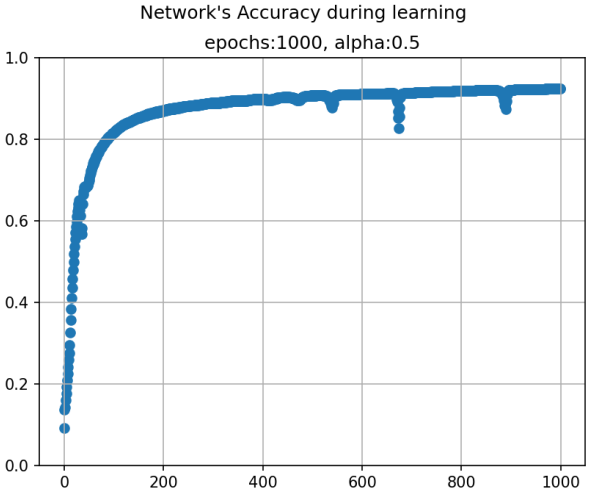
- N - liczba wejść sieci,
- $f()$ - funkcja aktywacji,
- w - macierz z wagami,
- $K_1, K2$ - liczba neuronów w obu warstwach sieci

1.5 Użyte parametry sieci

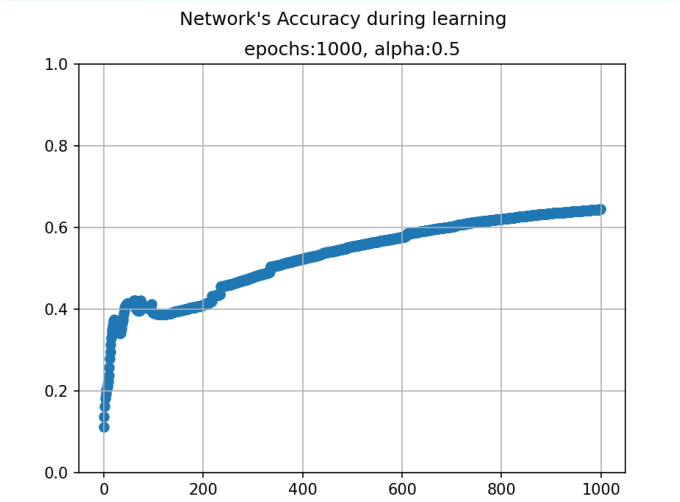
- Liczba wejść: 784 - każdy piksel obsługiwany jest przez osobny neuron,
- Liczba wyjść: 10 - wektor 10-cio elementowy składający się z możliwych kategorii klasyfikacji (0-9),
- Liczba warstw ukrytych: sieć posiada 1 warstwę ukrytą, co jest wystarczające do rozwiązania tego problemu klasyfikacji,
- Liczba neuronów w warstwie ukrytej: 10,
- Współczynnik uczenia $alpha = 0.5$.

1.6 Wpływ funkcji aktywacji na skuteczność działania sieci

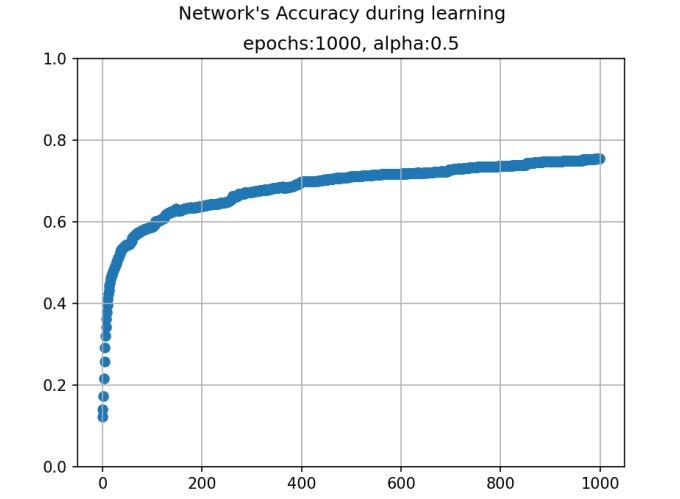
Wyniki przedstawione poniżej zostały uzyskane dla sieci z warstwą ukrytą, składającą się z dziesięciu neuronów. Dodatkowo pod każdym wykresem przedstawiona jest wartość $FinalAcc$, która jest wartością procentową skuteczności sieci dla zbioru danych testowych.



Rysunek 2: Zachowanie sieci przy użyciu funkcji $ReLU$. $FinalAcc = 91.88\%$.



Rysunek 3: Zachowanie sieci przy użyciu funkcji *Sigmoidalnej*. *FinalAcc* = 61.42%.



Rysunek 4: Zachowanie sieci przy użyciu funkcji *Tanh*. *FinalAcc* = 76.3%.

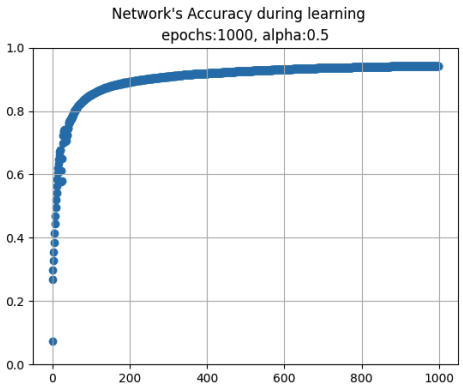
Wnioski:

- *ReLu* - Zgodnie z założeniami, opartymi na znalezionych informacjach o problemie klasyfikacji, jest to najlepsza funkcja dla danego przypadku. W końcowym etapie nauki na wykresie widać lekkie wahania, które na podstawie wykonanych testów, uznano, że zmieniają się wraz ze zmianą współczynnika uczenia *alpha*,
- *Sigmoid* - Funkcja wykazywała pewne problemy, procentowa wartość skuteczności sieci wahała się szczególnie na początku procesu nauki. Również ostateczny wynik nauczania okazał się najsłabszym z badanych,
- *Tanh* - Wynik działania sieci z wykorzystaniem funkcji aktywacji *Tanh* był lepszy niż wynik z wykorzystaniem funkcji *Sigmoidalnej*, chociaż w dalszym stopniu gorszy od tego, który zaprezentowała funkcja ReLU.

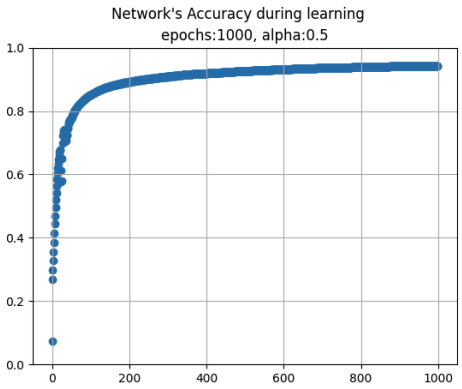
1.7 Otrzymane wyniki wraz z analizą końcową

Na zaimplementowanej sieci wykonano badania, zmieniając kolejno wybrany parametr i analizując wpłych tej zmiany na wynik działania sieci. Analogicznie jak wcześniej, pod każdym wykresem odpowiadajacym (niezależnemu od pozostałych) uruchomieniu całego procesu uczenia i testowania sieci, umieszczono wartość procentową *FinalAcc* odpowiadającą skuteczności sieci wyznaczonej na etapie jej testów.

Zmiana ilości neuronów:

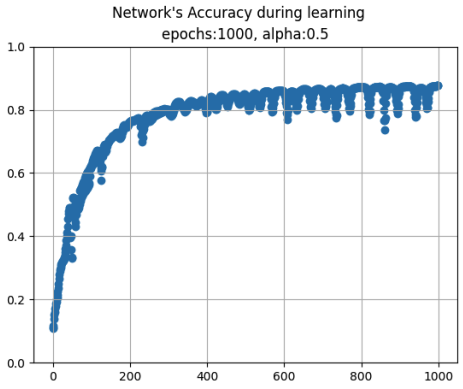


(a) Skuteczność dla 20 neuronów.
FinalAcc = 91.100000000001%.

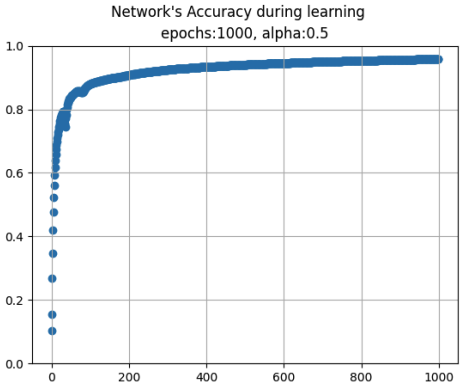


(b) Skuteczność dla 30 neuronów.
FinalAcc = 94.39999999999%.

Rysunek 5: Przykłady ilości neuronów dla zadowalającej skuteczności sieci.



(a) Skuteczność dla 5 neuronów.
FinalAcc = 87.8%.



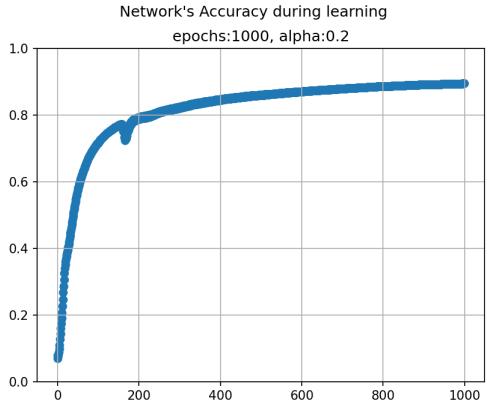
(b) Skuteczność dla 50 neuronów.
FinalAcc = **95.38%**.

Rysunek 6: Przykłady ilości neuronów dla niedouczenia
i największej osiągniętej skuteczności sieci.

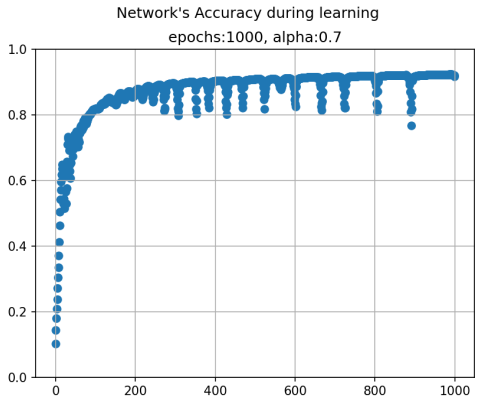
Na podstawie powyższych przykładowych badań i wielu innych, wywnioskowano, że zwiększanie ilości neuronów w warstwie ukrytej wpływa tylko i wyłącznie na

zwiększanie się wartości skuteczności sieci lub ze względu na brak możliwości testów na posiadanym sprzęcie, wartość liczby neuronów skutkująca przeuczeniem sieci, nie została osiągnięta.

Zmiana wartości współczynnika uczenia *alpha*:

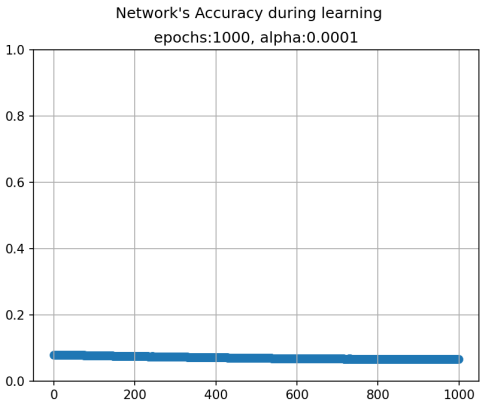


(a) Skuteczność dla $alpha = 0.2$. $FinalAcc = 89.34\%$.

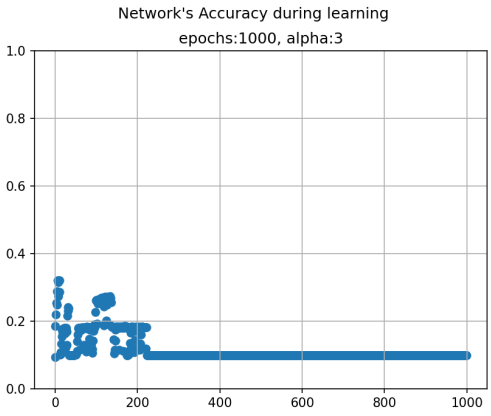


(b) Skuteczność dla $alpha = 0.7$. $FinalAcc = 91.42$.

Rysunek 7: Przykłady wartości współczynnika *alpha* dla zadowalającej skuteczności sieci.



(a) Skuteczność dla $alpha = 0.0001$. $FinalAcc = 6.81\%$.



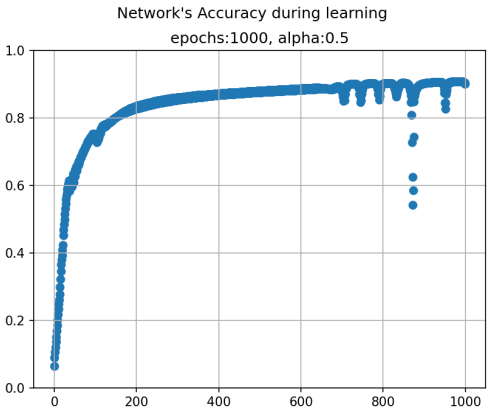
(b) Skuteczność dla $alpha = 3$. $FinalAcc = 10.6\%$.

Rysunek 8: Przykłady wartości współczynnika *alpha* dla niedouczenia i przeuczenia sieci.

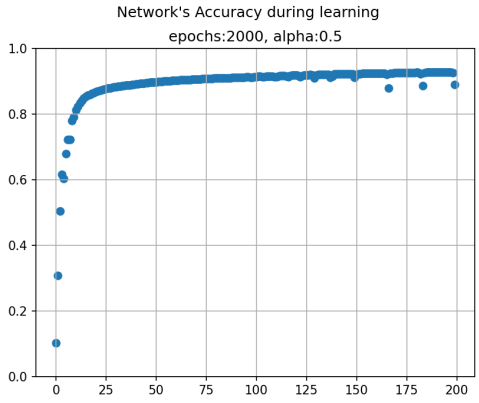
Na podstawie powyższych przykładowych badań i wielu innych, wywnioskowano, że optymalna wartość *alpha* zawiera się w przedziale $(0.2, 0.7)$, w którym nie wpływa znacząco na wartość skuteczności sieci. Zbyt duża wartość współczynnika

alpha skutkuje przeuczeniem sieci (Rysunek 8b). Badano również niższe wartości (Rysunek 8a), co doprowadziło do uzyskania niedouczenia przy *alpha* = 0.0001.

Zmiana ilości epok w procesie uczenia:

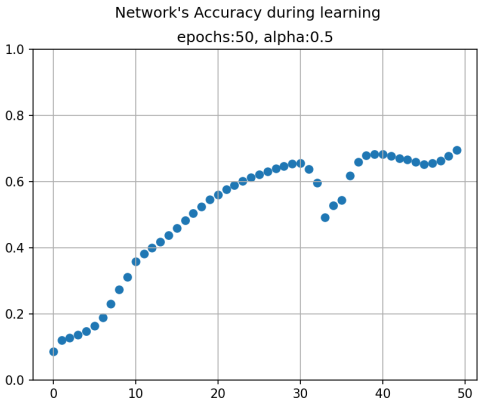


(a) Skuteczność dla ilości epok = 1000. *FinalAcc* = 89.4%.

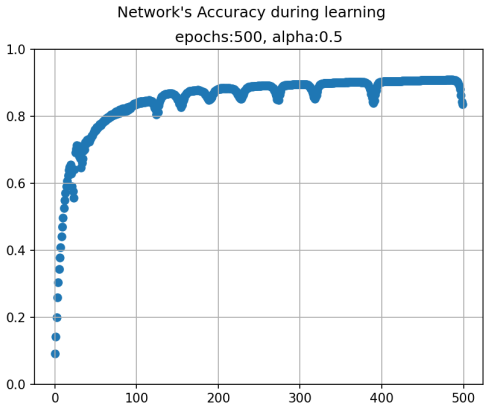


(b) Skuteczność dla ilości epok = 2000. *FinalAcc* = 92.3.

Rysunek 9: Przykłady wartości liczby epok dla zadowalającej skuteczności sieci.



(a) Skuteczność dla ilości epok = 50. *FinalAcc* = 70.62%.



(b) Skuteczność dla ilości epok = 500. *FinalAcc* = 83.88%.

Rysunek 10: Przykłady wartości ilości epok dla niedouczenia sieci.

Podobnie jak w przypadku ilości neuronów w warstwie ukrytej, w trakcie ekperymentowania nie udało się osiągnąć przuczenia poprzez zwiększenie liczby epok. Wysunięty został ten sam wniosek co poprzednio - dal tego problemu nie da się osiągnać przetrenowania poprzez zwiększanie liczby epok albo nie jesteśmy w stanie na posiadanych urządzeniach zasymulować takiego przypadku.

Wnioski ogólne:
Efekt przetrenowania sieci udało się osiągnąć tylko w 1 z 3 przeprowadzonych testów co najprawdopodobniej spowodowane jest ograniczeniami sprzętowymi. Najlepsza osiągnięta skuteczność sieci wynosi **95.38%**, a zostało to osiągnięte dla następujących parametrów:

- 1000 epok,
- *alpha* = 0.5,
- 50 neuronów,
- funkcja aktywacji - *ReLU*

2 Zadanie 2

2.1 Opis zadania

Zadanie opierało się na wykorzystaniu sieci neuronowej do aprokasymamcji danych dotyczących poziomu wody w morzu. Jako dane uczące sieć wykorzystano gotowe wartości poziomuu wody mierzone co godzinę, natomiast zadaniem zaprojektowanej sieci było stwierdzenie jakie wartości występowały pomiędzy pomiarami. Zdecydowano, że sieć będzie aproksymować poziom wody w odstępach 6 minutowych.

t[h]	0	1	2	3	4	5	6	7	8	9	10
h(t) [m]	1.0	1.32	1.6	1.54	1.41	1.01	0.6	0.42	0.2	0.51	0.8

Tabela 1: Zmierzone poziomy wody, jako zestaw danych uczących i walidujących.

2.2 Etap projektowania sieci

W projekcie zadaniem było stworzenie sieci neuronowej aproksymującej funkcję jednej zmiennej. Na wejście funkcji doporwadzano wartość czasu, a sieć zwracała dla danego argumenty wartość poziomu wody. Na etapie projektowania ustaliliśmy, że dla takiej aproksymacji warstwa wejściowa i wyjściowa sieci będą miały po jednym neuronie. Parametr *beta*, który wpływa na funkcję aktywacji *sigmoid*, został ustawiony na 1.25 na podstawie eksperymentów i ich analizy. Jako mechanizmy kontroli sieci zastosowaliśmy algorytm wstecznej propagacji błędu do uczenia sieci. Jako funkcję kosztu wybraliśmy *Średni Błąd Kwadratowy* (MSE), która jest powszechnie stosowana do problemów regresji.

2.3 Implementacja wybranych fragmentów sieci

Wagi i biasy zostały zainicjalizowane przy użyciu funkcji *np.random.rand()*, a biasy zostały początkowo ustawione na zero. Do aktywacji neuronów w warstwach ukrytej i wyjściowej zdecydowaliśmy się na funkcję *sigmoid*. Jej wzór to:

$$sigmoid(x) = \frac{1}{1 + e^{-\beta x}}$$

W związku z tym, że funkcja działa w ograniczonym zakresie, wartości zostały dopasowane poprzez ich przeskalowanie. Jako funkcję kosztu wybraliśmy średni błąd kwadratowy (MSE).
Proces propagacji składa się z dwóch etapów: propagacji do przodu (forward propagation) oraz propagacji wstecznej (backward propagation). Pierwszy obejmuje obliczenie wartości na kolejnych warstwach, natomiast drugi polega na aktualizacji wag i biasów na podstawie uzyskanych błędów wykorzystując do tego metodę spadku gradientu.

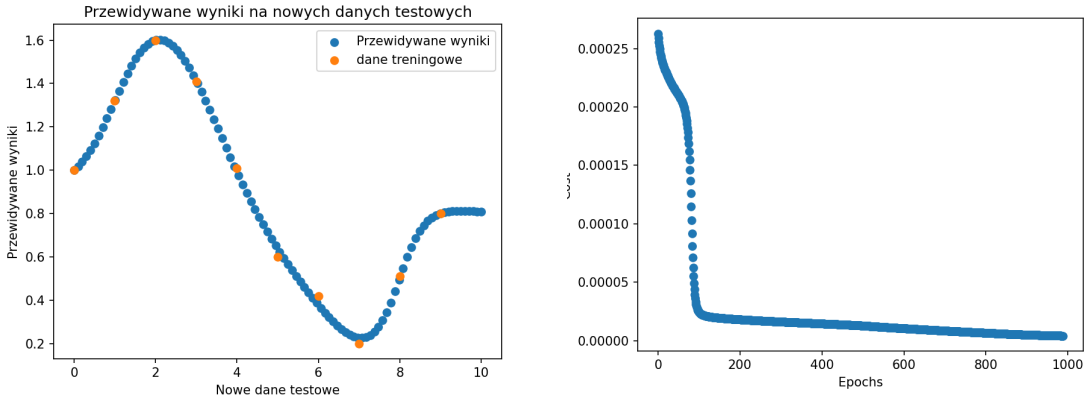
2.4 Użyte parametry sieci

- Liczba warstw ukrytych: W naszym przypadku najlepiej sprawdziło się 2 warstwy ukryte. Należy pamiętać jednak, że nie jest to wartość uniwersalna, i dla bardziej złożonych funkcji może ulec zmianie, i będzie konieczność dodania warstw,
- Liczba neuronów w warstwie ukrytej: W naszym przypadku najlepiej sprawdziło się 10 neuronów na warstwę. Należy pamiętać jednak, że nie jest to wartość uniwersalna, i dla bardziej złożonych funkcji może ulec zmianie, i będzie konieczność dodania neuronów w warstwie,
- Liczba wejść: 1 - sieć aproksymująca funkcję jednej zmiennej powinna zawierać jedno wejście,
- Liczba wyjść: 1 - sieć aproksymująca funkcję jednej zmiennej powinna zawierać jedno wyjście,
- Współczynnik uczenia $alpha = 0.1$.

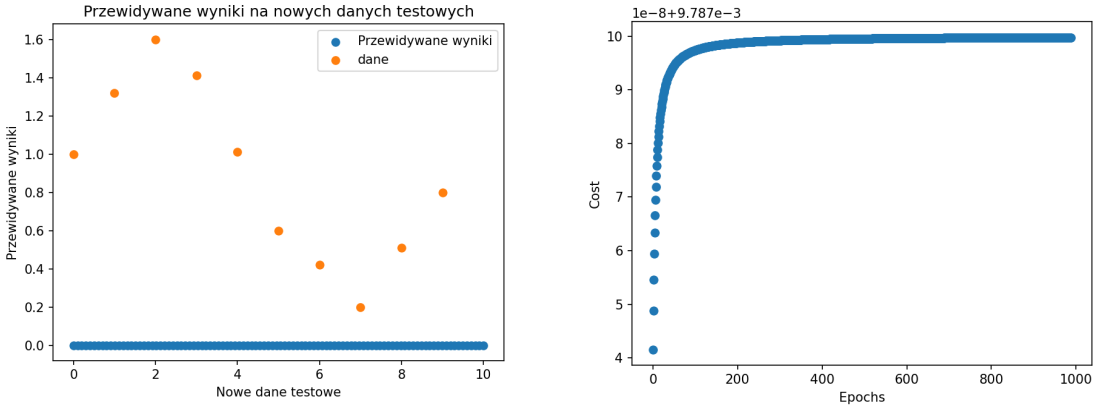
W następnych etapach przedstawiono zapamiętane parametry sieci, które gwarantowały największa skuteczność oraz efekty działania sieci dla granicznych wartości parametrów w celu prezentacji efektów niedouczenia bądź przeuczenia.

2.5 Wpływ funkcji aktywacji na skuteczność działania sieci

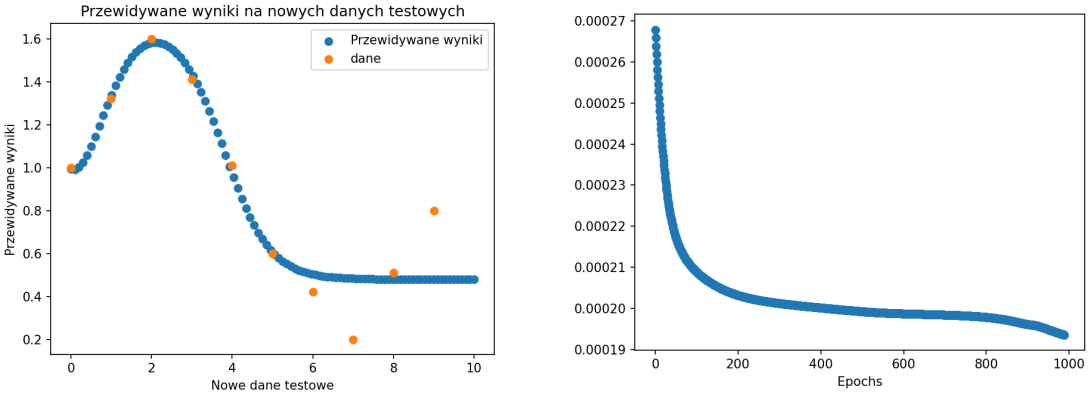
W tym punkcie zajęto się badaniem zachowania sieci przy użyciu różnych funkcji aktywacji. Każda z prób została przeprowadzona dla sieci o 2 warstwach ukrytych, każdej po 10 neuronów oraz procesie uczenia trwającego 1000 epok.



Rysunek 11: Zachowanie sieci przy użyciu funkcji Sigmoid



Rysunek 12: Zachowanie sieci przy użyciu funkcji ReLU



Rysunek 13: Zachowanie sieci przy użyciu funkcji Softmax

Wnioski:

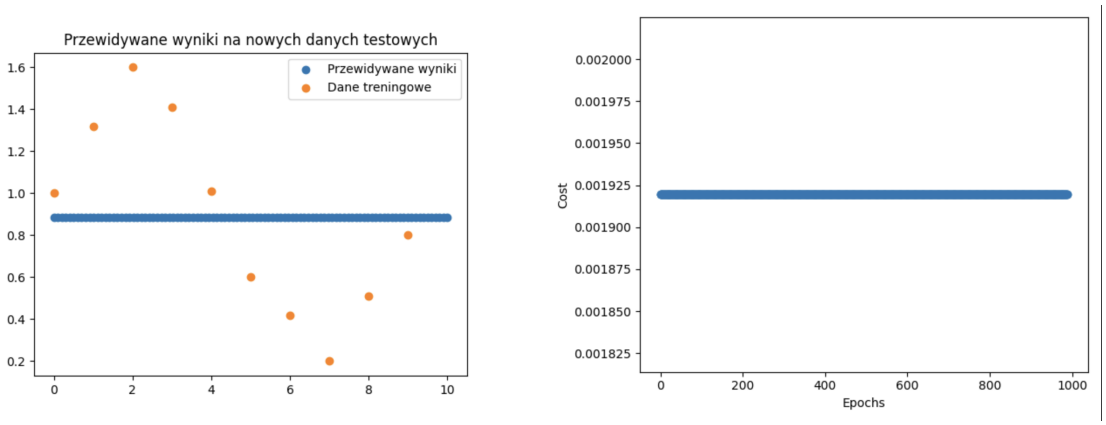
- *Sigmoid* - zgodnie z oczekiwaniami, jes to najlepsza funkcja dla problemu aproksymacji,
- *ReLU* - jej natura linii prostej oraz braku przegięcia uniemożliwiła sieci poprawną aproksymację,
- *Softmax* - ta funkcja przy problemie aproksymacji poradziła sobie tylko przy pierwszej zmianie kirunku kolejnych wartości, jednak dalej softmax ustabilizowała aproksymowaną wartość w jednym punkcie, co spowodowało pominięcie drugiego przegięcia.

2.6 Otrzymane wyniki wraz z analizą końcową

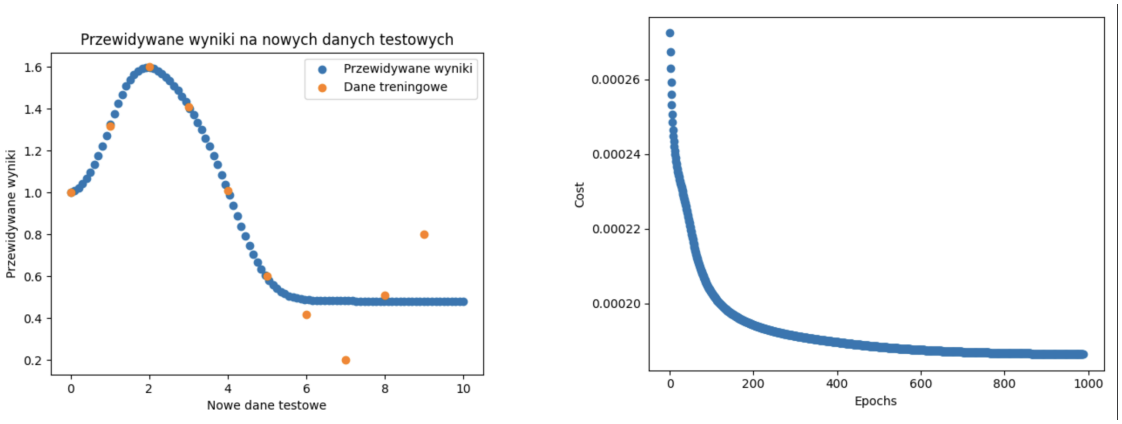
Po przeprowadzeniu treningu na dostarczonych danych otrzymaliśmy aproksymowane wyniki na nowych danych testowych. Wyniki te zostały przedstawione na wykresach, gdzie porównaliśmy je z rzeczywistymi danymi treningowymi. Treningi wykonywane były dla miliona epok, a wykres błędu w czasie został przedstawiony, aby zobaczyć, jak zmienia się on w trakcie uczenia.

Wykresy wraz z opisem zawierającym użyte parametry w sieci z sigmoidalną funkcją aktywacji:

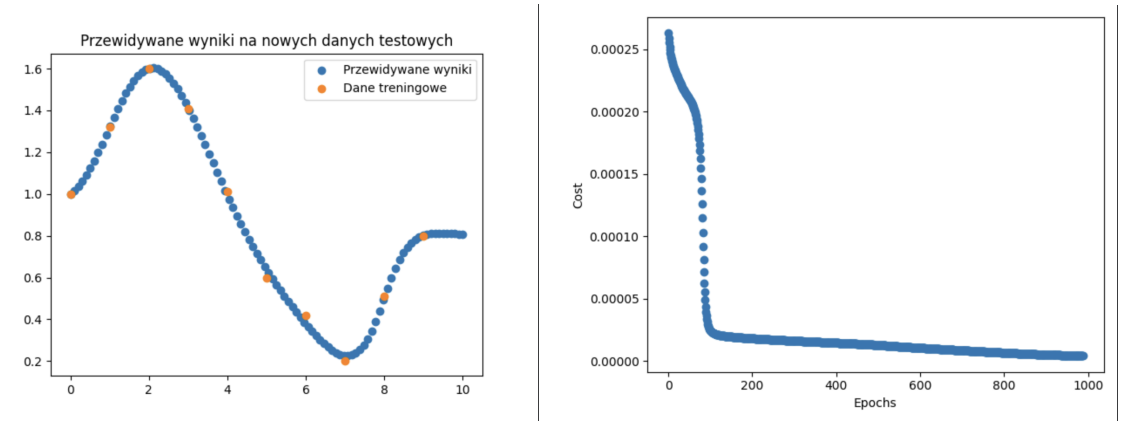
Zmiana liczby warstw ukrytych:



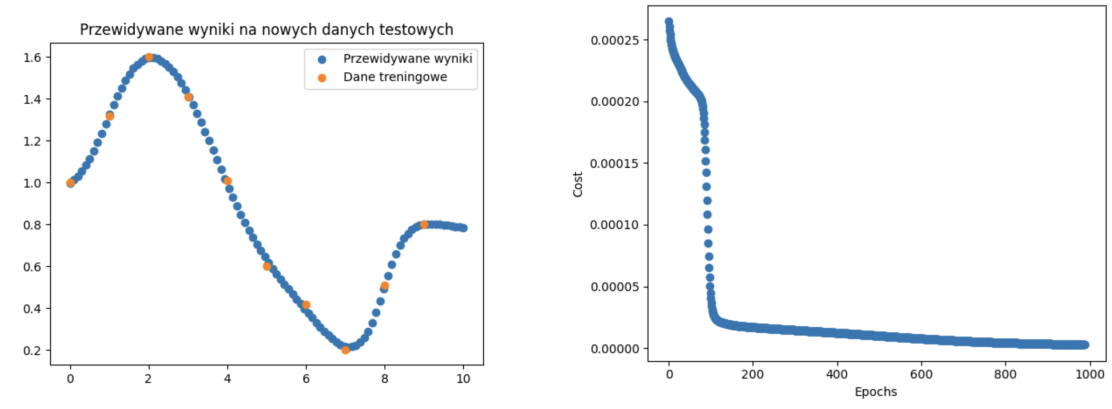
Rysunek 14: 10 neuronów, 0 w. ukrytych, 1m epok.



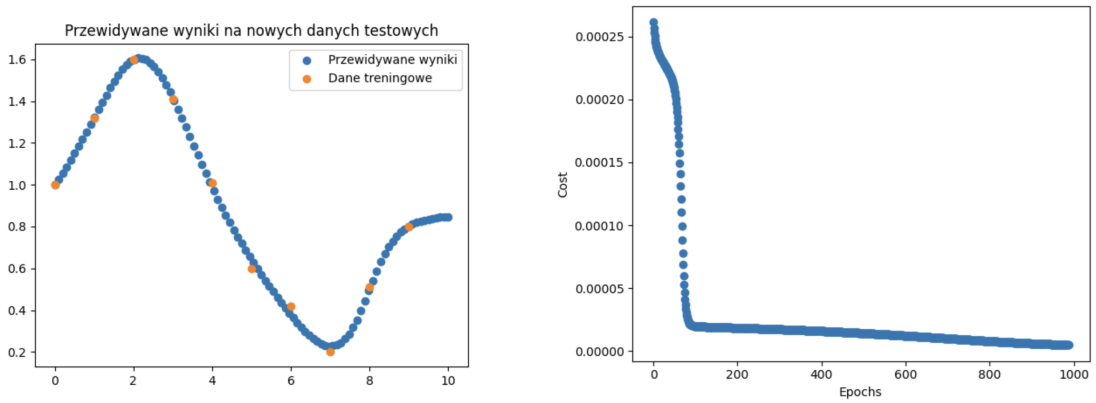
Rysunek 15: 10 neuronów, 1 w. ukryta, 1m epok.



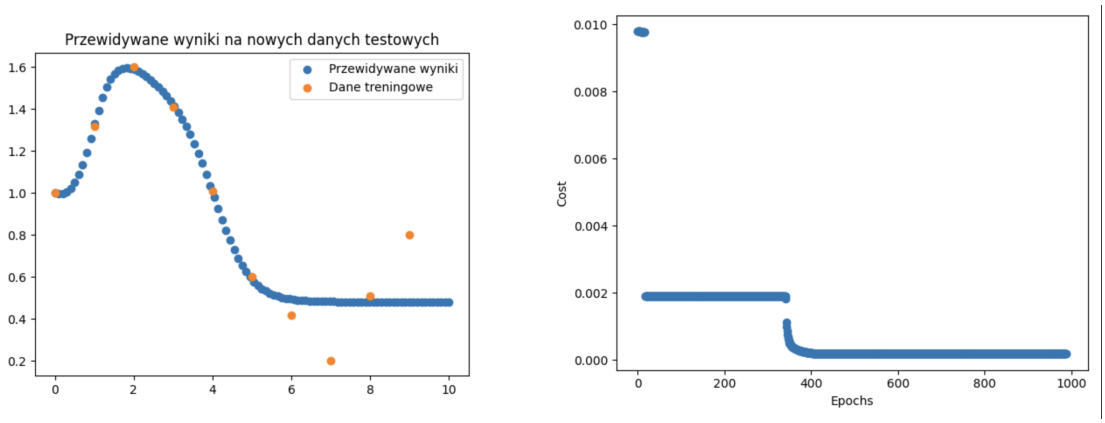
Rysunek 16: 10 neuronów, 2 w. ukryte, 1m epok.



Rysunek 17: 10 neuronów, 3 w. ukryte, 1m epok.



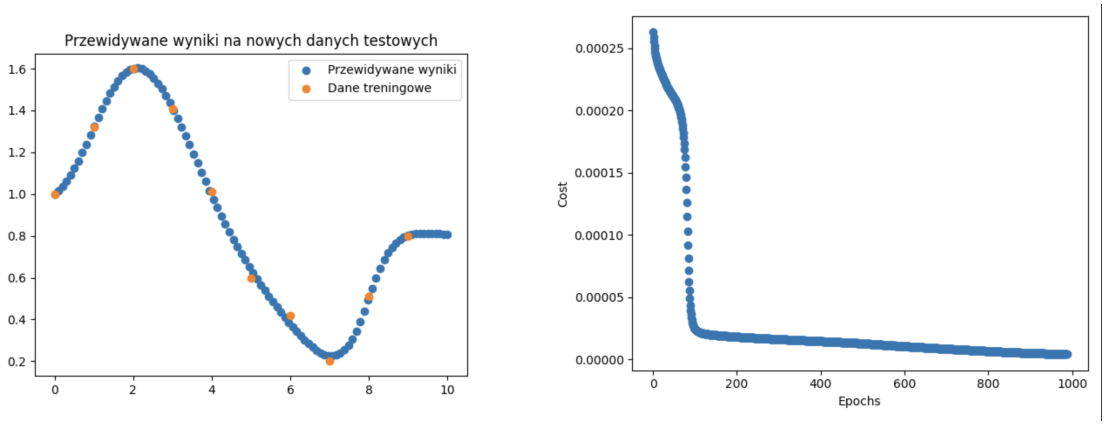
Rysunek 18: 10 neuronów, 4 w. ukryte, 1m epok.



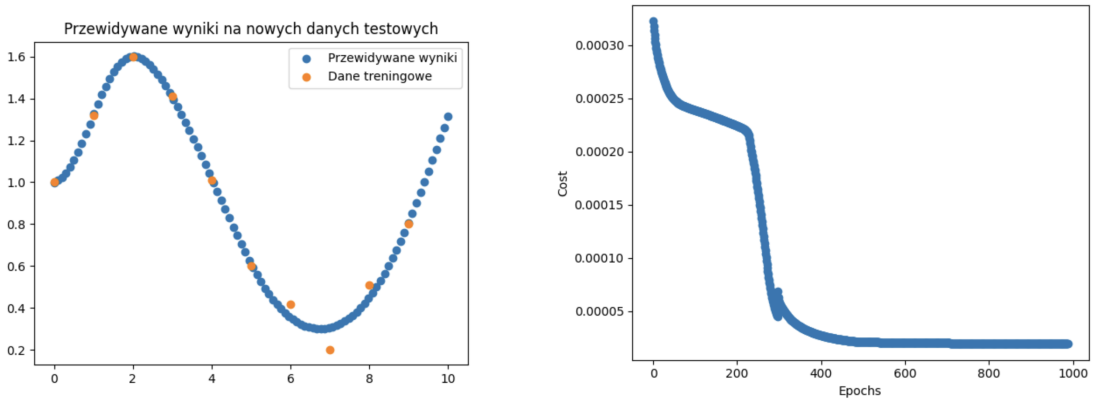
Rysunek 19: 30 neuronów, 20 w. ukrytych, 1m epok (Przykład przetrenowania sieci).

Z powyższych wykresów można wywniowskować, że dla danego problemu minimalnym rozmiarem do poprawnej ilości przegięć w funkcji aproksymowanej ilość warstw ukrytych wynosi 2, a zwiększanie jej w pewnym momencie zaczyna zmniejszać skuteczność aproksymacji sieci. Skuteczność zauważalnie spada, gdy liczba warstw ukrytych sieci przewyższy liczbę neuronów w warstwie oraz gdy obie te liczby osiągają rząd wartości większy niż 20.

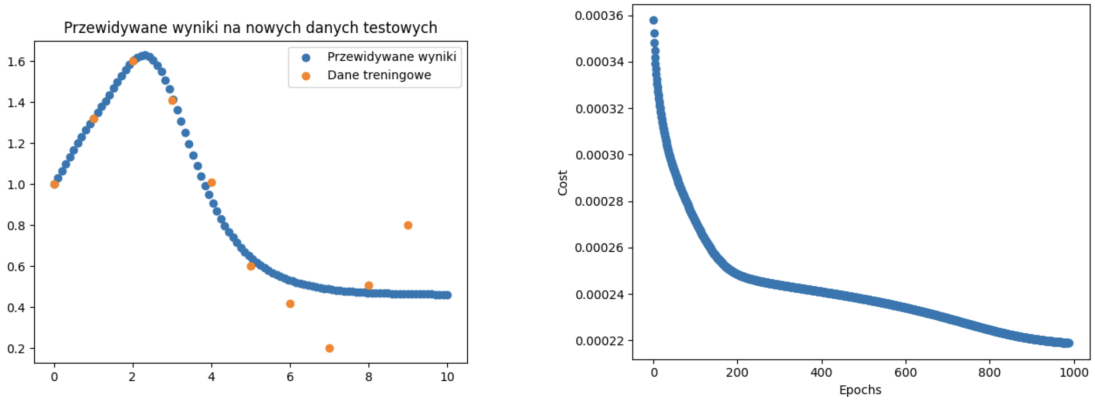
Zmiana współczynnika uczenia *etha*:
(na sieci 10 neuronów, 2 warstwy ukryte, 1m epok)



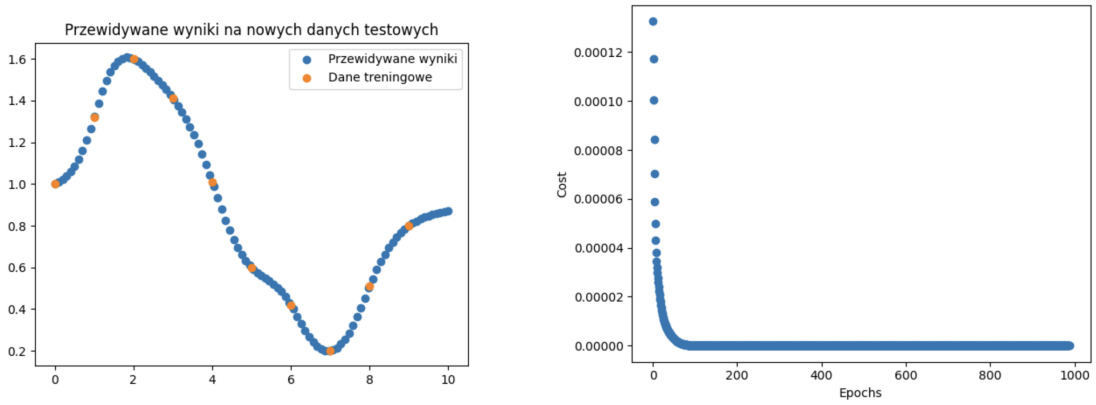
Rysunek 20: Współczynnik *etha* = 1.



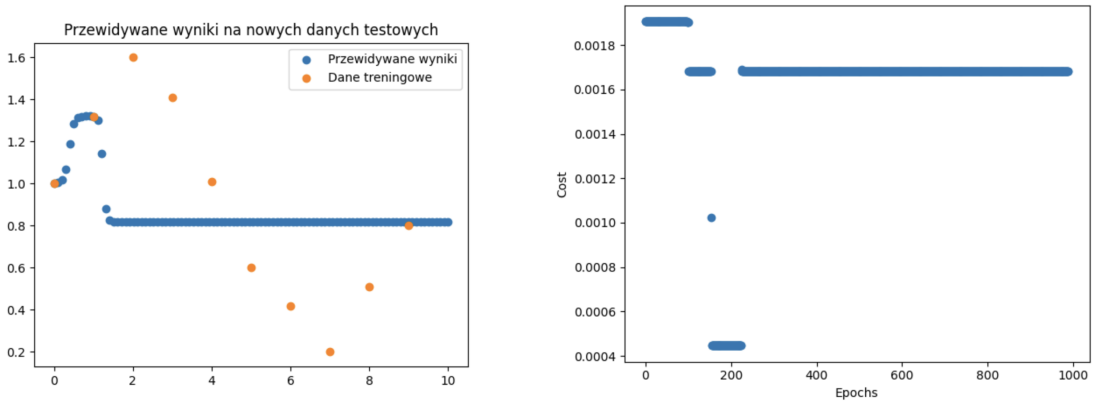
Rysunek 21: Współczynnik *etha* = 0.5.



Rysunek 22: Współczynnik *etha* = 0.25.



Rysunek 23: Współczynnik $etha = 5$.



Rysunek 24: Współczynnik $etha = 50$.

Powyższe wykresy pokazują wpływ zmiany współczynnika uczenia na skuteczność sieci. Można zauważyć że podobnie jak w poprzednich parametrach, wartość współczynnika posiada swój zakres w którym działa, lepiej bądź gorzej, ale działa. Poza tym zakresem przy dolnej granicy dochodzi do niedouczenia, natomiast nad górna granicą wartości występuje przeuczenie, które całkowicie zaburza skutecnzość sieci.