WSI - Sieci neuronowe

Jan Szymczak, Adam Sokołowski

Maj 2024

1 Cel ćwiczenia

Implementacja perceptronu wielowarstwowego oraz wybranego algorytmu optymalizacji gradientowej (Stochastyczny Spadek Gradientu) z algorytmem propagacji wstecznej. Zaimplementowany przez nas algorytm obsługiwał 3 możliwe funkcje aktywacji: sigmoid (na którym przeprowadzane były wszystkie testy), Relu oraz tangens hiperboliczny. Funkcje straty z kolei były do wyboru następujące: błąd średniokwadratowy (używany przez nas w testach) oraz cross-entropy.

2 Wyniki eksperymentów

2.1 Test zaproponowany przez prowadzącego

Na samym początku przetestowano działanie programu na funkcji logicznej XOR.

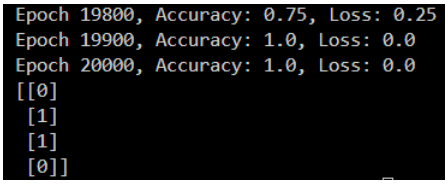

Figure 1: Test XOR

Jak widać na powyższym obrazku wyniki są zgodne z oczekiwaniami.

2.2 Wpływ parametru learning rate

Parametr ten wpływa na szybkość zmian gradientu funkcji optymalizacji. Zbyt mała wartość wydłuża znacznie czas działania programu (potrzeba więcej iteracji, aby sieć mogła się nauczyć) lub powodować "utykanie" rozwiązania w minimum lokalnym, a nie globalnym. Z kolei zbyt duża może powodować, że zmiany gradientu będą za bardzo gwałtowne i program będzie "przeskakiwać" rozwiązanie optymalne, nigdy go nie znajdując. W celu przetestowania i znalezienia optymalnej wartości parametru, trenowaliśmy model na danych walidacyjnych dla wartości parametru lr z przedziału $(0; 2.1)$ ze skokiem równym 0.1 dla 2 warstw ukrytych z wielkością odpowiednio 128 i 64, oraz dla maksymalnej ilości iteracji (epochs) równej 1000. Tak prezentuje się wykres wartości funkcji straty w zależności od parametru lr:

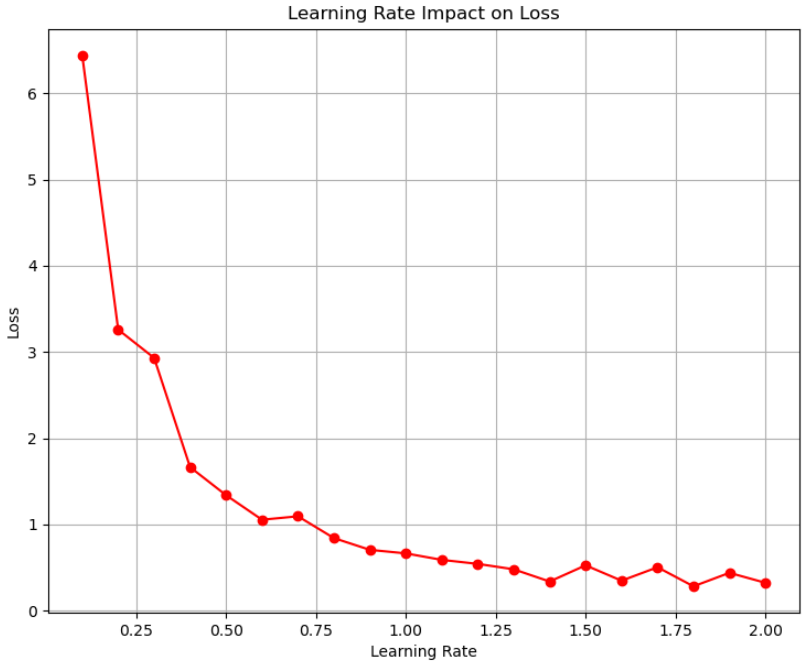

Figure 2: Wartość funkcji straty od parametru lr

A tak wykres dokładności predykcji (na zbiorze walidacyjnym):

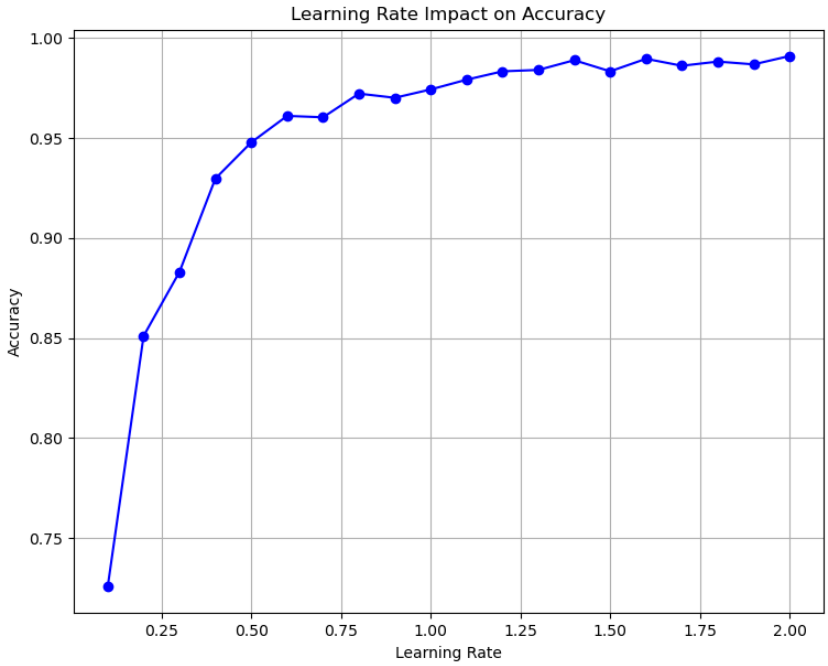

Figure 3: Dokładność od parametru lr

Mimo, że dokładność dalej stale się zwiększała, uznaliśmy, że w dalszych testach przyjmiemy wartość parametru jako 2, nie szukając lepszego rozwiązania. Aby pokazać jednak, że zbyt duża wartość parametru wypacza działanie programu, przeprowadziliśmy jednorazowy test dla *lr* równego 30. Wyniki dla tego testu wyglądały następująco:

```
Epoch 1900, Accuracy: 0.10020876826722339, Loss: 20.862212943632567
Epoch 2000, Accuracy: 0.10160055671537926, Loss: 10.696590118302018
Test Accuracy on MNIST: 0.09722222222222222
```

Figure 4: Ostateczna uzyskana dokładność w niedziałającym programie

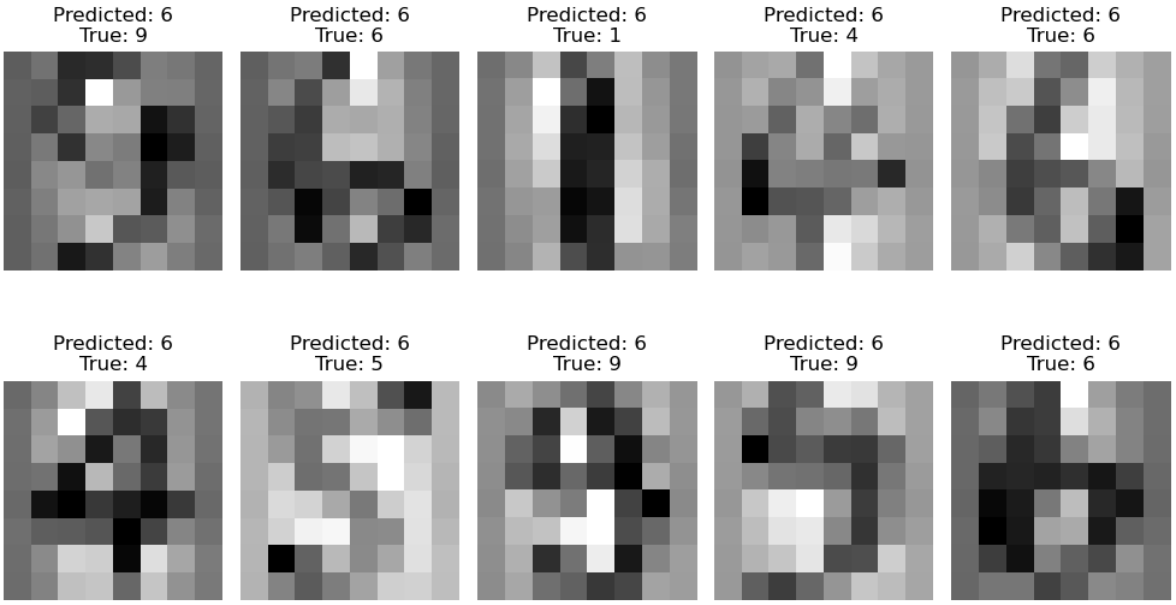

Figure 5: Przewidywania uzyskane na niedziałającym programie

Algorytm w ogóle się nie uczył, przeskakiwał rozwiązania optymalne i wszędzie przewidział 6, co jest oczywiście dalekie od prawdy.

2.3 Wpływ parametru epochs (epoki)

Parametr epok określa, ile razy cały zbiór danych zostanie przetworzony przez algorytm uczenia. Innymi słowy, jedna epoka oznacza, że każdy przykład w zbiorze danych został użyty raz do zaktualizowania wag modelu.

Dobór liczby epok jest bardzo ważny gdyż, jeśli liczba epok jest zbyt mała, model może nie mieć wystarczającej liczby iteracji, aby nauczyć się wzorców w danych. Skutkuje to niedouczeniem (underfitting). Natomiast jeśli liczba epok jest zbyt duża, model może zacząć uczyć się szczegółów specyficznych dla danych treningowych, co prowadzi do przeuczenia (overfitting). W takim przypadku model osiąga bardzo dobrą wydajność na danych treningowych, ale słabą na danych walidacyjnych.

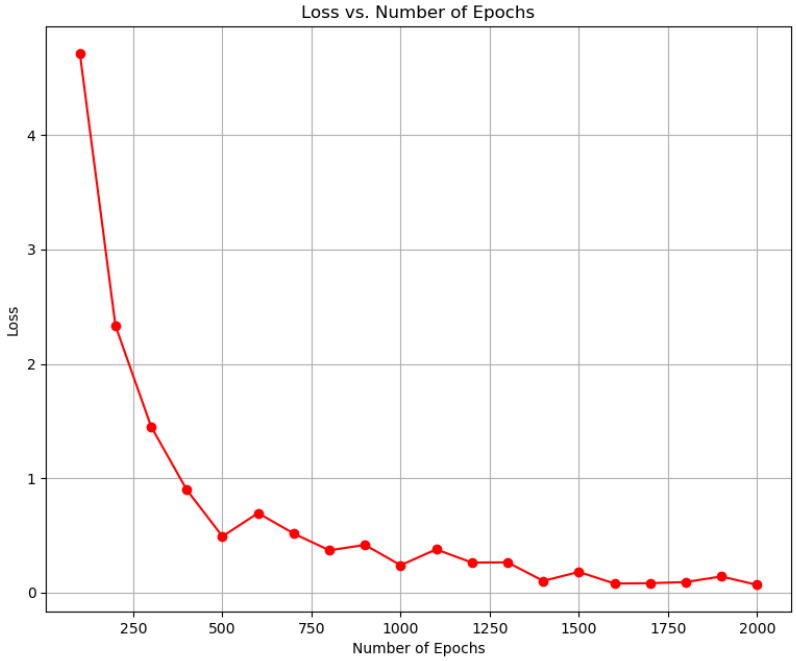

Figure 6: Wartość funkcji straty od parametru epochs

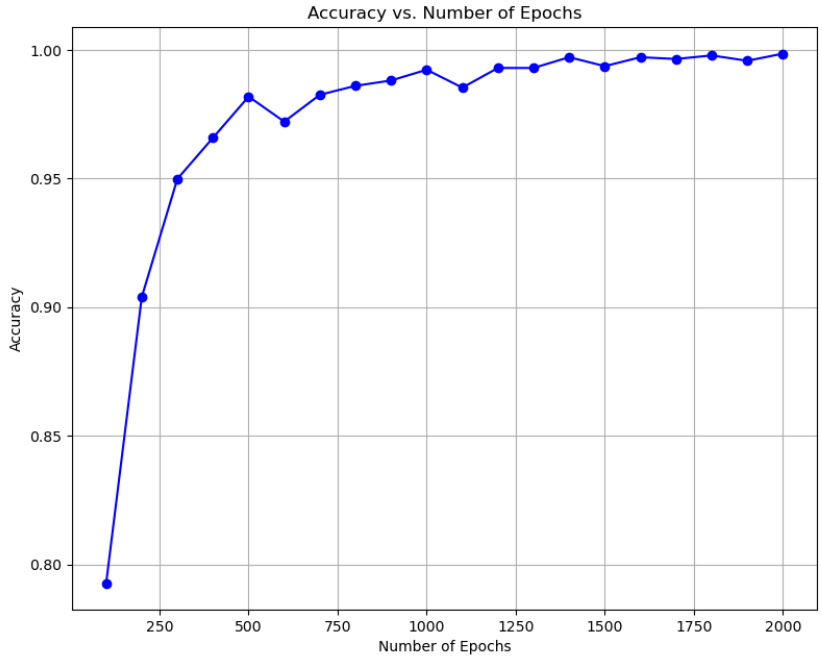

Figure 7: Dokładność od parametru epochs

Pokazane wyżej zdjęcia były dla zbioru walidacyjnego. Overfitting nie występuje, Accuray jest blisko jedności, a loss zera.

2.4 Wpływ wielkości warstw ukrytych

W celu przetestowania wpływu wielkości warstw ukrytych, przeprowadziliśmy testy dla kolejno: 1 warstwa o wielkości 3, 2 warstwy o wielkości [3, 3], [10, 10], [30, 60] oraz [128, 64] oraz 3 warstw o wielkości [30, 30, 30]. Reszta parametrów to: *epochs*=1000 oraz *lr*=2, do nauki użyto zbioru uczącego, a walidacji walidacyjnego. Tak prezentują się wyniki:

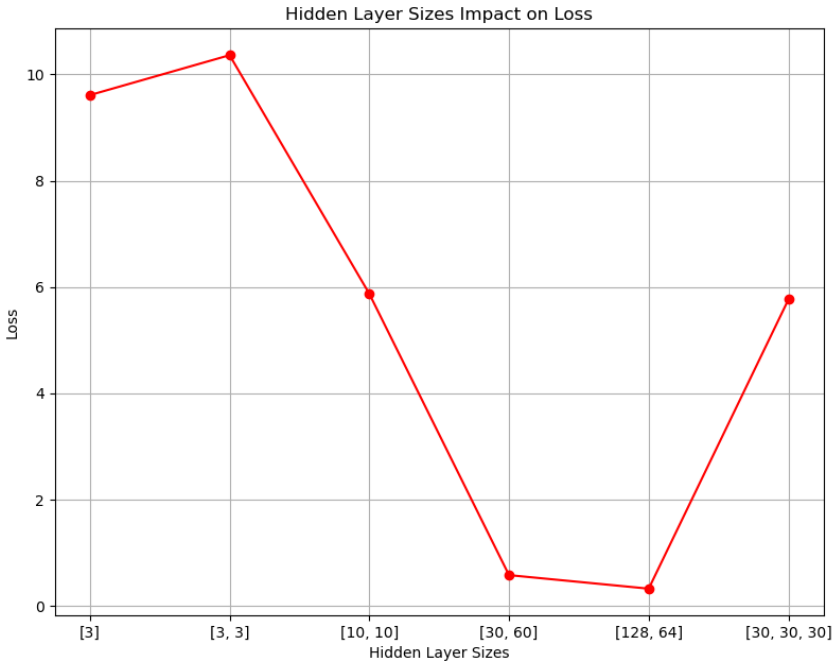

Figure 8: Wartość funkcji straty dla różnych wielkości warstwy ukrytej

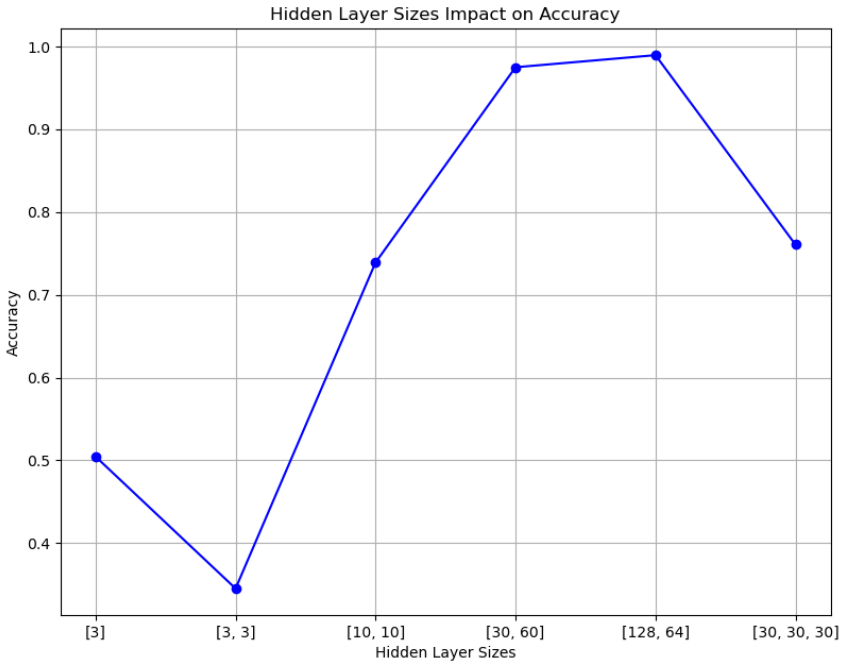

Figure 9: Dokładność dla różnych wielkości warstwy ukrytej

Najlepsze wyniki zdecydowanie uzyskaliśmy dla warstw o wielkości [128, 64] i z taka wielkością przeprowadziliśmy ostateczny test na danych testowych. Zauważyliśmy, że wraz ze wzrostem wielkości warstwy ukrytej, algorytm potrzebował większej ilości iteracji (epochs), aby dojść do poprawnych wyników. Algorytm na początku trochę "stał w miejscu" z dokładnością oraz wartością funkcji straty, aby po przejściu pewnej ilości epok zacząć znacząco się uczyć.

2.5 Ostateczny test

Ostateczny test przeprowadziliśmy z parametrami kolejno: *epochs*=2000, *lr*=2 oraz wielkością warstwy ukrytej [128, 64]. Ostateczna dokładność wyniosła odpowiednio 99.8% dla danych trenujących i 96.7% dla danych testowych. Tak wygląda 10 przykładowych przewidywań programu:

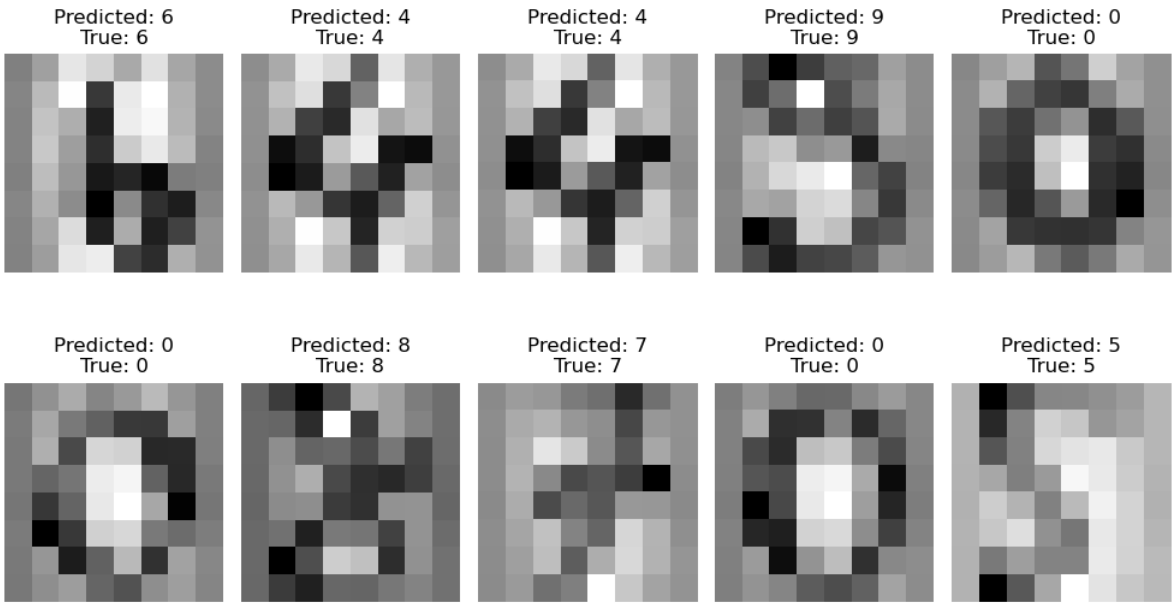

Figure 10: Przewidywania uzyskane w ostatnim teście

Jak widać, wszystkie cyfry zostały trafione.