Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Sztuczna Inteligencja

Sprawozdanie z laboratorium

Autor

Aleksandra Walczybok

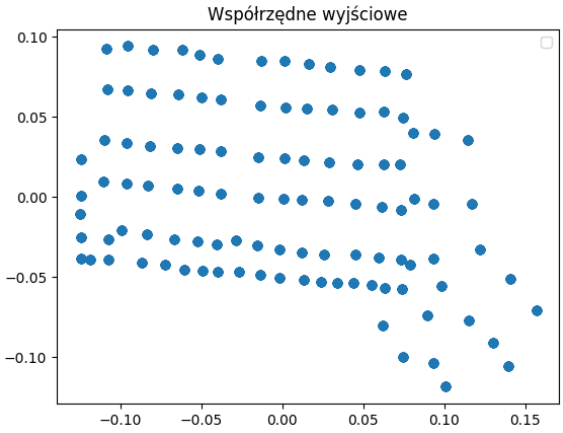
nr albumu: **272454**

kierunek: **Inżynieria systemów**

2 styczeń 2025

1 Wstęp – sformułowanie problemu

Celem zadania było zaimplementowanie sieci neuronowej, która na podstawie odczytu siły sygnału radiowego określi położenie (pozycję) odbiornika. Na wejściu znajduje się odczyt siły sygnału z 16 nadajników radiowych. Na wyjściu należy uzyskać pozycję (x,y) odbiornika.



Rysunek 1: Współrzędne punktów wyjściowych

2 Opis rozwiązania

2.1 Wybór architektury sieci neuronowej

W rozważaniach wzięto pod uwagę 5 możliwości stworzenia sieci neuronowej. W każdej z nich warstwa wejściowa zawierała 16 węzłów, a wyjściowa 2 węzły - takie liczby wynikały z charakteru danych oraz problemu.

Były to następujące konfiguracje:

- 16 węzłów - 4 węzły - 2 węzły
- 16 węzłów - 8 węzłówów - 2 węzły
- 16 węzłów - 10 węzłów - 2 węzły
- 16 węzłów - 8 węzłow - 4 węzły - 2 węzły
- 16 węzłów - 10 węzłów - 4 węzły - 2 węzły

Aby wybrać odpowiednią architekturę, zastosowano metodę k-krotnej walidacji krzyżowej. K-krotna walidacja to technika oceny modelu, w której dane są dzielone na k równych części (foldów). Model jest trenowany na $k-1$ foldach, a testowany na pozostałym foldzie; proces powtarza się k-krotnie, za każdym razem zmieniając fold testowy, a końcowy wynik to średnia z wszystkich iteracji. Dane podzielono na 5 foldów. Jako metrykę do oceny poprawności sieci użyto MSE (mean squared error). Początkowe paramtery sieci ustawiono na następujące:

- num epoch = 10000

- learning rate = 0.01
- momentum = 0
- activation func=sigmoid
- activation derivative=sigmoid derivative

Wyniki:

	16-4-2	16-8-2	16-10-2	16-8-4-2	16-10-8-2
0	0.003391	0.002407	0.001643	0.002173	0.003170
1	0.002637	0.002278	0.001768	0.002006	0.002737
2	0.003515	0.002322	0.001699	0.002356	0.003107
3	0.003220	0.002376	0.001802	0.002239	0.003146
4	0.002981	0.002097	0.001941	0.002039	0.002880
avg	**0.003149**	**0.002296**	**0.001770**	**0.002163**	**0.003008**

Tabela 1: Wyniki dla różnych konfiguracji sieci.

Okazało się, że najlepsze wyniki osiąga architektura złożona z 16 węzłów w pierwszej warstwie, w kolejnej 10 węzłów i w ostatniej 2 węzły. Zbliżone wyniki w każdym foldzie świadczą o stabilności modelu i jego zdolności do generalizacji na różnych podzbiorach danych.

2.2 Opis matematyczny sieci

Definiujmy następujące macierze:

$$X \in R^{m\times 16}, \quad W_1 \in R^{16\times 10}, \quad b_1 \in R^{1\times 10}$$
$$H \in R^{m\times 10}, \quad W_2 \in R^{10\times 2}, \quad b_2 \in R^{1\times 2}$$
$$P \in R^{m\times 2}, \quad Y \in R^{m\times 2}$$

1. Propagacja w przód (Forward Propagation)

Podczas propagacji w przód obliczamy wartości dla każdej warstwy. Dla pierwszej warstwy ukrytej:

$$z_1 = XW_1 + b_1$$
$$h_1 = \sigma(z_1)$$

gdzie σ to funkcja aktywacji przyjęta jako sigmoida.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Następnie, dla drugiej warstwy ukrytej:

$$z_2 = h_1W_2 + b_2$$
$$h_2 = \sigma(z_2)$$

Na końcu:

$$p = h_2$$

gdzie p to wynik wyjściowy sieci.

2. Funkcja kosztu

Funkcja kosztu C jest obliczana jako średni błąd kwadratowy pomiędzy wartościami rzeczywistymi y a prognozami p:

$$C = \frac{1}{m}\sum_{i=1}^{m}(y_i - p_i)^2$$

gdzie m to liczba próbek w zbiorze danych, y to wartości rzeczywiste, a p to wartości przewidywane przez sieć.
Jako funkcję kosztu wybrano średni błąd kwadratowy, ponieważ zadanie dotyczy regresji - predykcji współrzędnych.

3. Propagacja wsteczna (Backpropagation)

Podczas propagacji wstecznej obliczamy gradienty dla wag i biasów.

Najpierw obliczamy gradient funkcji kosztu względem wyjścia:

$$\frac{\partial C}{\partial p} = 2 \cdot \frac{p - y}{m}$$

Następnie obliczamy gradienty dla wag i biasów drugiej warstwy:

$$\frac{\partial C}{\partial W_2} = h_1^T \cdot \frac{\partial C}{\partial p}$$

$$\frac{\partial C}{\partial b_2} = \sum \frac{\partial C}{\partial p}$$

Dalej, obliczamy gradienty dla pierwszej warstwy ukrytej:

$$\frac{\partial C}{\partial h_1} = \frac{\partial C}{\partial p} \cdot W_2^T$$

$$\frac{\partial C}{\partial z_1} = \frac{\partial C}{\partial h_1} \cdot \sigma'(z_1)$$

$$\frac{\partial C}{\partial W_1} = X^T \cdot \frac{\partial C}{\partial z_1}$$

$$\frac{\partial C}{\partial b_1} = \sum \frac{\partial C}{\partial z_1}$$

4. Aktualizacja wag

Ostatecznie, aktualizujemy wagi i biasy za pomocą algorytmu gradientu prostego:

$$W_1 := W_1 - \eta \frac{\partial C}{\partial W_1}$$

$$b_1 := b_1 - \eta \frac{\partial C}{\partial b_1}$$

$$W_2 := W_2 - \eta \frac{\partial C}{\partial W_2}$$

$$b_2 := b_2 - \eta \frac{\partial C}{\partial b_2}$$

gdzie η to współczynnik uczenia.

Drugim sposobem na zaaktualizowanie wag jest zastosowanie momentum.
Momentum wprowadza dodatkowy składnik v, który akumuluje wcześniejsze gradienty, co pozwala na szybsze i bardziej stabilne uczenie. Wzory dla aktualizacji wag z użyciem momentum są następujące:

1. Aktualizacja prędkości (v):

$$v_t = \beta v_{t-1} + (1 - \beta) \nabla L$$

gdzie:

- v_t to prędkość w kroku t,
- β to współczynnik momentum ($\beta \in [0, 1]$),
- ∇L to gradient funkcji kosztu w bieżącym kroku.

2. Aktualizacja wag (W):

$$W_t = W_{t-1} - \eta v_t$$

gdzie:

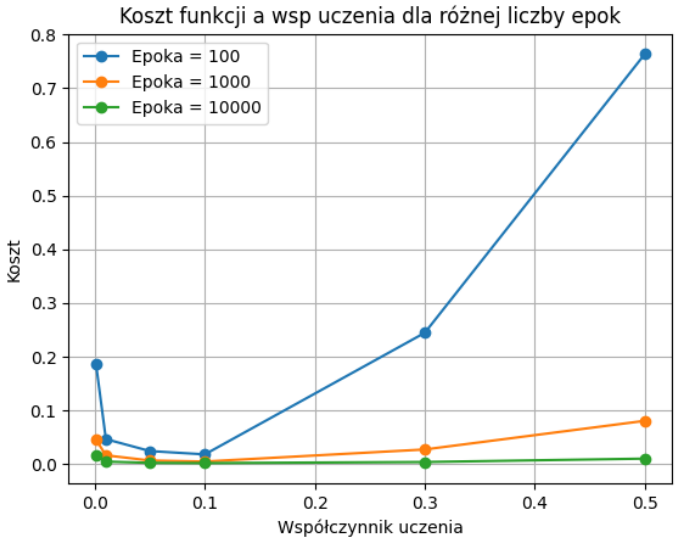
- η to współczynnik uczenia (*learning rate*),
- W_t to wagi w kroku t,
- v_t to prędkość w kroku t.

2.3 Wybór hiperparamterów

2.3.1 Współczynnik uczenia

Współczynnik uczenia (η) to kluczowy hiperparametr, który kontroluje, jak duże kroki wykonuje model w kierunku minimalizacji funkcji kosztu podczas aktualizacji wag. Jego wartość musi być odpowiednio dobrana – zbyt mały współczynnik powoduje wolne uczenie, a zbyt duży może prowadzić do niestabilności i braku konwergencji modelu.

Zależności funkcji kosztu od różnych wartości współczynnika pokazano na wykresie:



Rysunek 2: Wybór współczynnika uczenia

Do znalezienia optymalnej wartości użyto również k-krotnej walidacji krzyżowej. Wyniki zaprezentowano w tabeli poniżej:

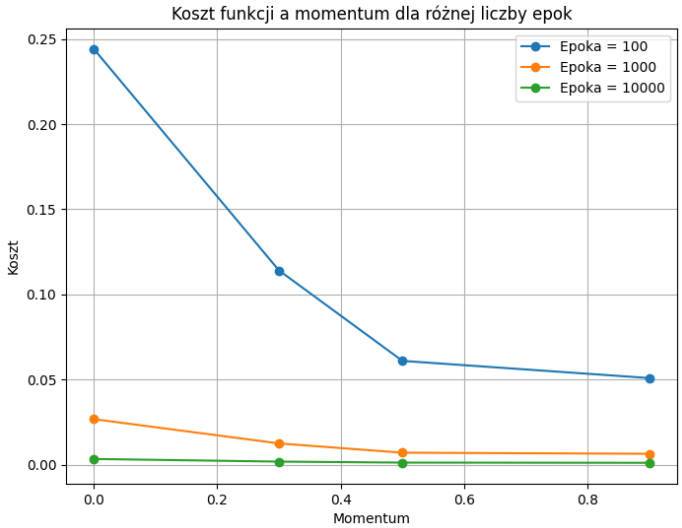
	0.001	**0.01**	**0.05**	**0.1**	**0.3**	**0.5**
0	0.006578	0.001643	0.000906	0.000702	0.000522	0.001499
1	0.006619	0.001768	0.000934	0.000732	0.000556	0.001446
2	0.006501	0.001699	0.000862	0.000658	0.000497	0.001370
3	0.007299	0.001802	0.000923	0.000710	0.000522	0.001457
4	0.007184	0.001941	0.000972	0.000736	0.000528	0.001461
avg	**0.006836**	**0.001770**	**0.000919**	**0.000708**	**0.000525**	**0.001447**

Tabela 2: Wyniki dla różnych wartości współczynnika uczenia.

Analizując wykres oraz wyniki k-krotnej walidacji, widać, iż najlepiej działa współczynnik o wartości 0.1 oraz 0.3 dla liczby epok równej 10000.

2.3.2 Momentum

Momentum to technika optymalizacyjna, która dodaje "pamięć" o poprzednich krokach w procesie aktualizacji wag, aby przyspieszyć konwergencję i zmniejszyć oscylacje w dolinach funkcji kosztu. Dzięki temu model efektywniej porusza się w kierunkach zbieżnych i unika lokalnych minimów, co przekłada się na bardziej stabilny proces uczenia.



Rysunek 3: Wybór momentum

Do znalezienia optymalnej wartości użyto również k-krotnej walidacji krzyżowej. Wyniki zaprezentowano w tabeli poniżej:

	0.0	**0.3**	**0.5**	**0.9**
0	0.000522	0.000465	0.000445	0.000387
1	0.000556	0.000503	0.000484	0.000406
2	0.000497	0.000457	0.000439	0.000380
3	0.000522	0.000469	0.000451	0.000385
4	0.000528	0.000469	0.000454	0.000395
avg	**0.000525**	**0.000473**	**0.000455**	**0.000391**

Tabela 3: Wyniki dla różnych wartości momentum.

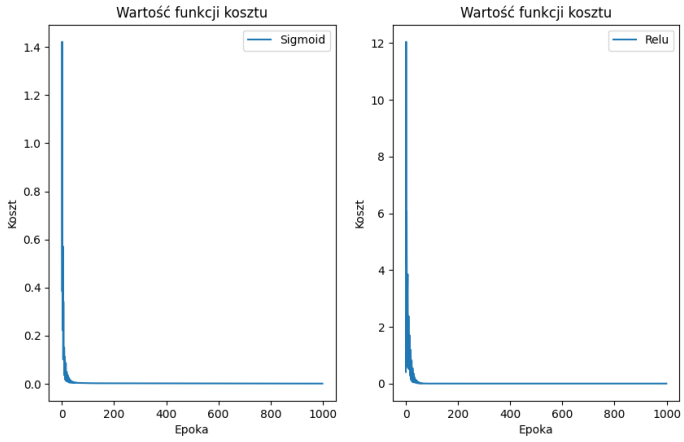
Anlizując wykres oraz tabelę nie ma wątpliwości, że najlepszym wyborem jest momentum o wartości równiej 0.9.

2.3.3 Funkcje aktywacji

Funkcja aktywacji decyduje o tym, czy neuron powinien zostać aktywowany na podstawie sumy wejściowych sygnałów. Dzięki funkcjom aktywacji, takim jak sigmoid i ReLU, model jest w stanie uczyć się nieliniowych zależności, co jest kluczowe dla rozwiązywania bardziej skomplikowanych problemów.

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

$$\text{ReLU}(x) = \max(0, x)$$



Rysunek 4: Wybór funkcji aktywacji

Dla sigmoidy:
Średni koszt po wszystkich epokach: 0.006340688200203722
Ostatni koszt: 0.0008143269001520867

Dla ReLU:
Średni koszt po wszystkich epokach: 0.05570483315510474
Ostatni koszt: 0.004225800376968005

Wyniki wskazują, że lepszym wyborem jest sigmoida.

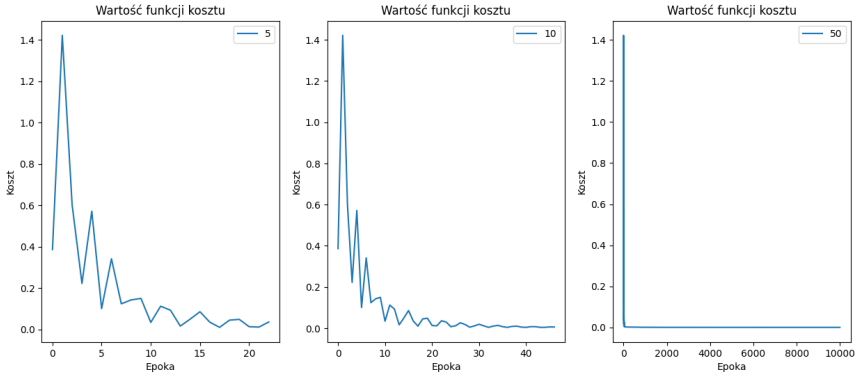
2.3.4 Warunek stopu

W projekcie przyjęto sprawdzanie wartości kosztu funkcji i zapamiętywanie najlepszych wag podczas całego uczenia. Dodatkowo wprowadzono warunek stopu. Testowano różne wartości parametru *patience*, który mówi o tym, ile jest dopuszczalnych epok, podczas których nie zmienia się wartość funkcji kosztu.

Dla patience = 5:
Średni koszt: 0.2024516813817938
Ostatni koszt: 0.036171943893651186

Dla patience = 10:
Średni koszt: 0.104333574315995
Ostatni koszt: 0.0059391214460272415

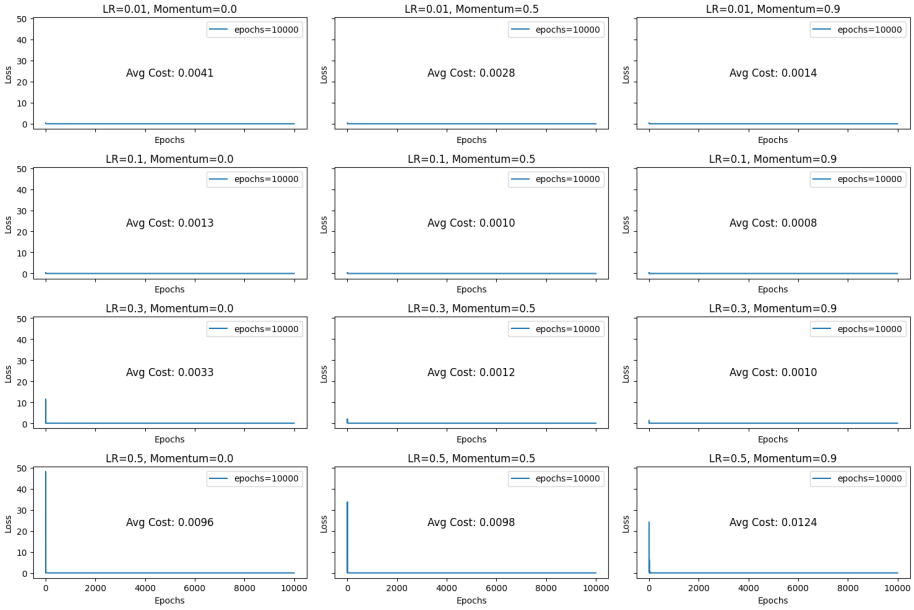
Dla patience = 50:
Średni koszt: 0.0010378968254335724
Ostatni koszt: 0.000378623262669399



Rysunek 5: Wpływ różnych warunków stopu na sieć

Małe wartości ‘patience‘ powodują wczesne zatrzymanie uczenia sieci. Różnica błędów, jaka jest między wartościami ‘patience‘, jest jednak bardzo duża, przez co zdecydowano użyć wartości 50, gdyż gwarantuje ona o wiele mniejszą wartość błędu.

Dodatkowe porównanie hiperparametrów:



Rysunek 6: Badanie różnych hiperamatrów sieci

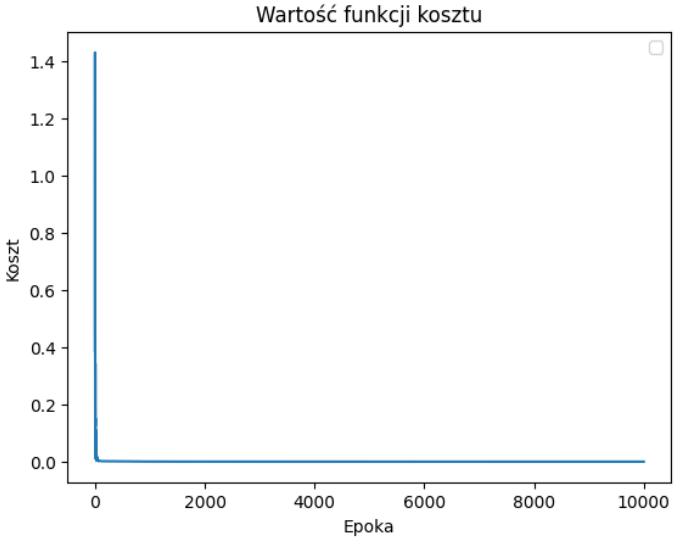
Najlepsze wartości kosztu osiąga połączenie momentum=0.9 i lr=0.1 lub lr=0.3. Te wartości wyszły również podczas wcześniejszych analiz.

Ostatecznie przyjęto następujące hiparametry:

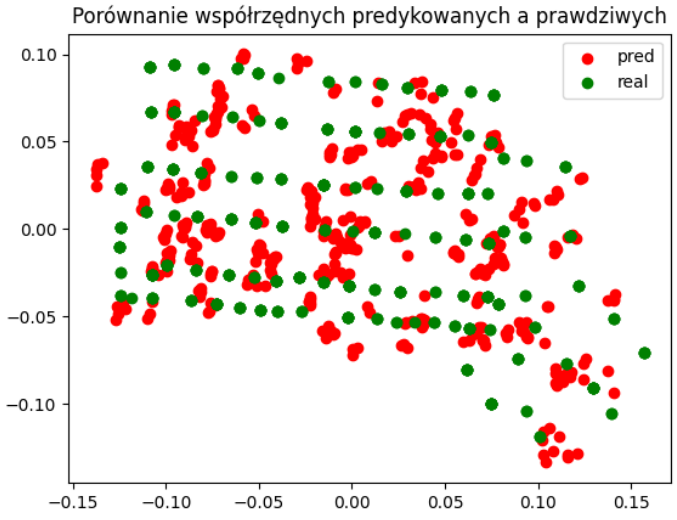
- num epoch = 10000
- lr = 0.3
- momentum = 0.9
- patience = 50
- input layer = 16
- hidden layer = 10
- output layer = 2

3 Testowanie sieci

Podczas ostatecznego uczenia i testowania sieci podzielono zbiór na zbiór treningowy (80%) oraz testowy (20%).



Rysunek 7: Wykres kosztu podczas uczenia



Rysunek 8: Predykowane współrzędne

Średni koszt po wszystkich epokach: 0.0010364569626836048

Ostatni koszt: 0.00037304242111729986

4 Wnioski

Na podstawie przeprowadzonych badań i analiz można wyciągnąć następujące wnioski:

- **Wybór architektury sieci neuronowej:** Spośród rozważanych konfiguracji, najlepsze wyniki osiągnęła architektura 16-10-2. Zapewniła ona najniższą średnią wartość błędu MSE (0.001770),.
- **Wpływ współczynnika uczenia (η):** Optymalną wartością współczynnika uczenia okazało się 0.3 i 0.1)).
- **Wpływ momentum:** Wprowadzenie momentum o wartości 0.9 pozwoliło znacząco obniżyć wartość błędu (średnia wartość 0.000391), a także przyspieszyło proces uczenia poprzez lepsze wykorzystanie wcześniejszych gradientów.
- **Funkcje aktywacji:** Funkcja sigmoidalna okazała się skuteczniejsza od ReLU, zapewniając niższy koszt końcowy oraz stabilniejsze uczenie. Mimo że ReLU jest bardziej efektywna w wielu zastosowaniach, w tym przypadku sigmoida lepiej poradziła sobie z problemem regresji.
- **Warunek stopu:** Przyjęcie wartości *patience* równej 50 pozwoliło na uzyskanie minimalnego błędu przy jednoczesnym zachowaniu stabilności procesu uczenia. Mniejsze wartości *patience* skutkowały przedwczesnym zakończeniem uczenia, co prowadziło do gorszych wyników.
- **Ogólna skuteczność modelu:** Finalnie zaimplementowana sieć neuronowa z dobranymi hiperparametrami osiągnęła bardzo niskie wartości błędu MSE na zbiorze testowym, co dowodzi, że zastosowane podejście jest skuteczne w problemie lokalizacji odbiornika na podstawie siły sygnałów radiowych.

Mimo dobrych wyników funkcji kosztu, pokazane na wykresie porównanie współrzędnych prawdziwych z predykcjami wskazuje na możliwość dalszej optymalizacji poprzez eksplorację bardziej zaawansowanych metod, takich jak adaptacyjne optymalizatory (np. Adam) lub większe zbiory danych treningowych.

5 Dodatek

Kody źródłowe umieszczone zostały w repozytorium GitHub:
`https://github.com/aleksandra0014/MLP/tree/main/task2`.