

Sieci Neuronowe sprawozdanie

Zuzanna Sieńko

Kwiecień 2025

1 Wstęp

Sprawozdanie zawiera opis oraz wyniki eksperymentów przeprowadzonych w ramach zajęć z Metod Inteligencji Obliczeniowej w Analizie Danych z bloku tematycznego Sieci Neuronowe (NN). Każda z sekcji obejmuje analizę poszczególnych podzadań związanych z tym tematem oraz przedstawia eksperymenty realizowane w ramach wybranego zagadnienia. Wyniki w raporcie są na podstawie przeprowadzonych eksperymentów, nie wszystkie są zgodne z wymaganymi progami błędu, jednak te były prezentowane podczas zajęć.

2 Implementacja tematów

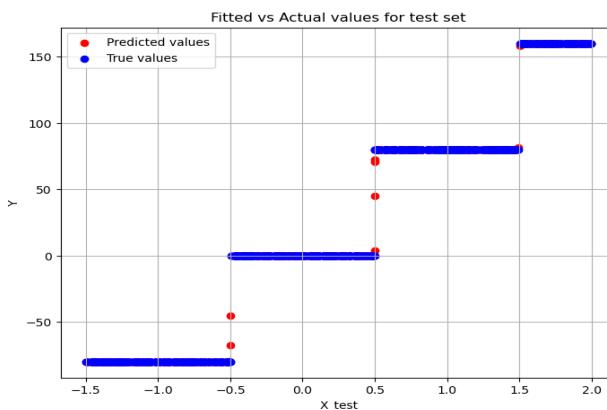
2.1 NN1: bazowa implementacja sieci neuronowej

2.1.1 Opis zadania

Zaimplementowano bazową strukturę sieci neuronowej, w której można ustawać liczbę warstw ukrytych i ilość neuronów w każdej warstwie oraz dowolne wag i biasy. Bazowa implementacja jako funkcję aktywacji przyjmuje funkcję sigmoidalną, na wyjściu sieci jest funkcja liniowa (na początku rozważamy zadania regresji).

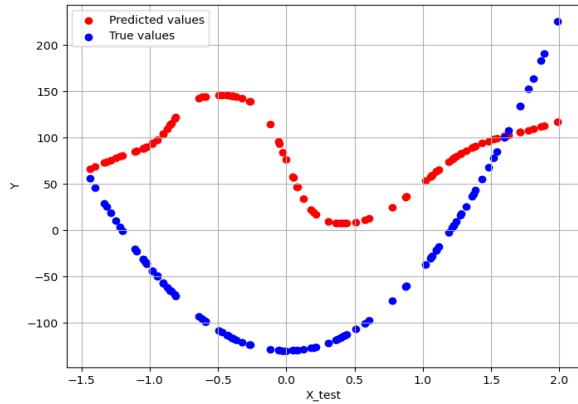
2.1.2 Eksperymenty

Dla zbiorów danych *Square Simple* oraz *Steps Large* próbowało dobrać wag i biasy dla różnych architektur ręcznie. Dla zbioru *Steps Large* zaprojektowano ręcznie sieć z jedną warstwą ukrytą (5 neuronów), gdzie odpowiednie wartości wag i biasów pozwalają aktywować niektóre neurony w zależności od przedziału wejściowego x . Dzięki temu, sumując aktywacje neuronów z wagami wyjściowymi i przesunięciem (bias = -80), uzyskiwane są konkretne wartości wyjściowe: -80, 0, 80, 160 dla różnych zakresów x . Wykorzystano duże wartości wag (1000), by uzyskać możliwie jednolite przejścia sigmoidów do wartości binarnych.

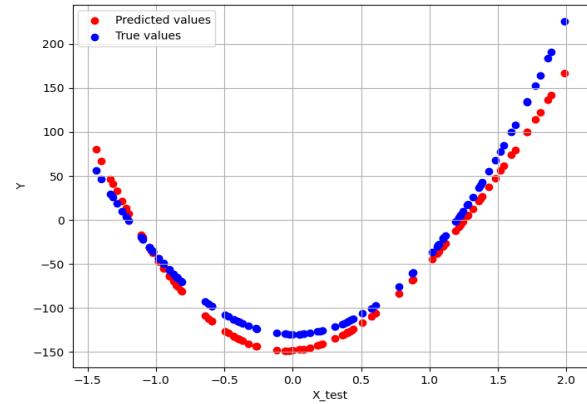


Rysunek 1: Dopasowanie sieci neuronowej z 5 neuronami w warstwie ukrytej z wagami i biasami opisanymi jak wyżej na zbiorze testowym (MSE 2.09).

Trudności pojawiły się przy zbiorze square simple, gdzie dobranie odpowiedniej architektury, wartości wag i biasów nie były oczywiste. Przeprowadzono symulacje, gdzie wagi i biasy były losowane z rozkładu jednostajnego z różnymi parametrami, lecz w żadnym przypadku błąd predykcji nie był zbliżony do oczekiwanej.



Rysunek 2: Porównanie danych testowych z przewidywaniami modelu, gdy wagi i biasy zostały ręcznie dobrane (MSE 22 238.29).



Rysunek 3: Porównanie danych testowych z przewidywaniami modelu, gdzie wagi i biasy zostały wybrane losowo w ramach symulacji (MSE 327.01).

2.1.3 Wnioski

Dobór wag i biasów dla sieci neuronowej jest wyzwaniem, zwłaszcza przy bardziej złożonych zbiorach danych. W przypadku zbioru *Square Simple* udało się ręcznie zaprojektować sieć, uzyskując oczekiwane wyniki. Natomiast przy *Square Simple* trudniej było dobrać odpowiednią architekturę, co sugeruje, że dla bardziej skomplikowanych danych ręczne ustawianie parametrów jest mniej efektywne.

2.2 NN2: implementacja propagacji wstecznej błędu

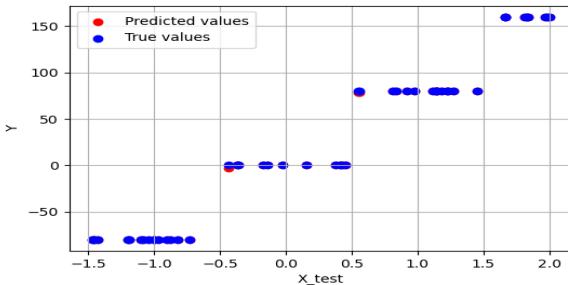
2.2.1 Opis zadania

W ramach tego zadania zaimplementowano propagację wsteczną błędy umożliwiającą sieci na dopasowywanie się wag i biasów do problemu na podstawie gradientu funkcji straty. Podstawowym mechanizmem propagacji wstecznej jest obliczanie błędu na wyjściu sieci, a następnie rozprzestrzenianie tego błędu wstecznie przez wszystkie warstwy, w celu obliczenia gradientów wag i biasów.

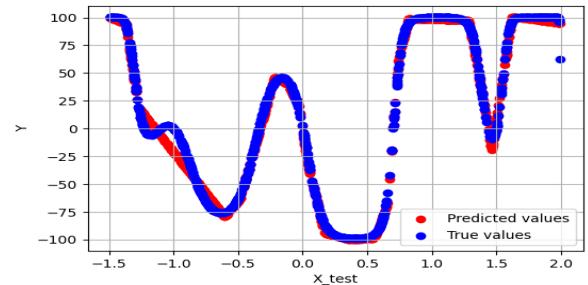
2.2.2 Eksperymenty

Zaimplementowany algorytm propagacji wstecznej został przetestowany na zaproponowanych zbiorach danych. Przedstawione są dopasowania wytrenowanych sieci neuronowych na dwóch zbiorach.

Wyniki eksperymentów przedstawiają się następująco:



Rysunek 4: Przewidywania sieci neuronowej dla danych wejściowych na zbiorze *Steps Small*.



Rysunek 5: Przewidywania sieci neuronowej dla danych wejściowych na zbiorze *Multimodal Large*.

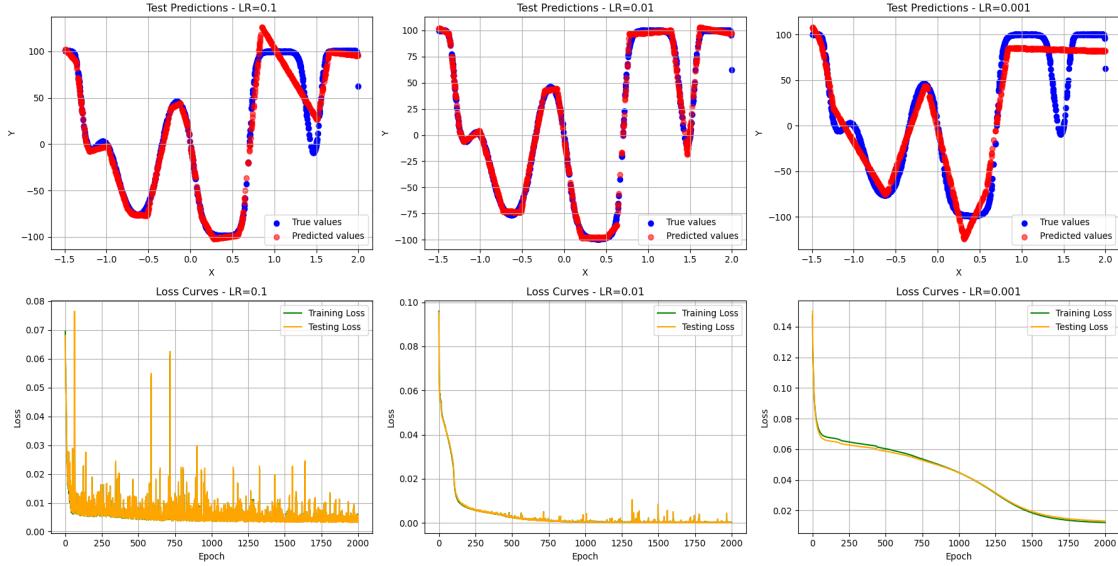
Wartości MSE na przedstawionych zbiorach wyniosły odpowiednio 0.32 i 33.13 na zbiorze testowym, co świadczy o dobrym dopasowaniu modelu do danych. Wynik ten sugeruje, że algorytm propagacji wstecznej skutecznie uczy się na podstawie dostarczonych danych, bez konieczności ręcznego ustawiania wartości wag i biasów dla sieci.

Zbiór danych	MSE na zbiorze testowym
Square Simple	0.64
Steps Small	0.32
Multimodal Large	33.13

Tabela 1: Wyniki treningu dla różnych zbiorów danych z algorytmem propagacji wstecznej.

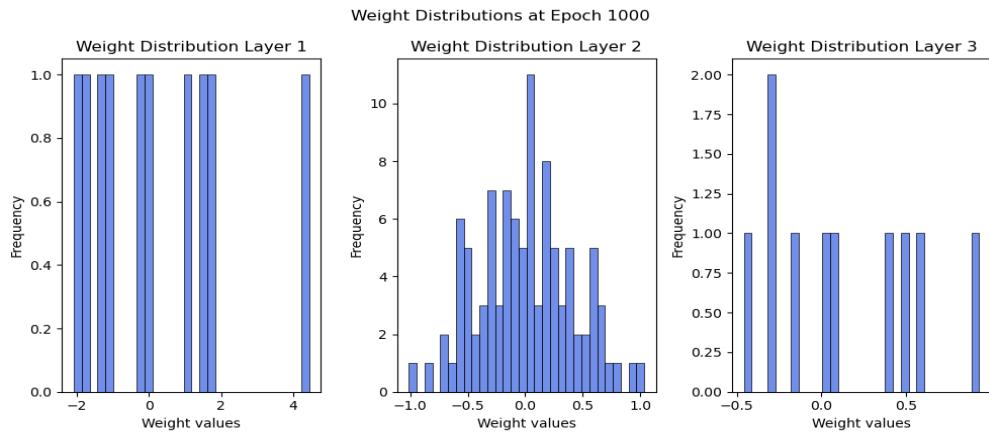
Przeprowadzono analizę wpływu współczynnika uczenia na zbieżność i dokładność modelu. Jak przedstawiono na Rysunku 6, dobór wartości ma kluczowe znaczenie dla efektywności uczenia sieci neuronowej. W przypadku zbioru *Multimodal Large* współczynniki 0.1, 0.01 i 0.001 dały odpowiednio wyniki MSE równe 132,17, 9,17 oraz 523,07. Wyniki wyraźnie wskazują, że nieodpowiedni dobór parametru prowadzi do istotnych problemów: zbyt wysoka wartość (0.1) uniemożliwia osiągnięcie minimum funkcji straty i niestabilny proces treningu, natomiast zbyt niska (0.001) znaczco spowalnia proces zbieżności, co znajduje odzwierciedlenie w znacznie wyższym błędzie MSE.

Model Comparison: Predictions vs Losses

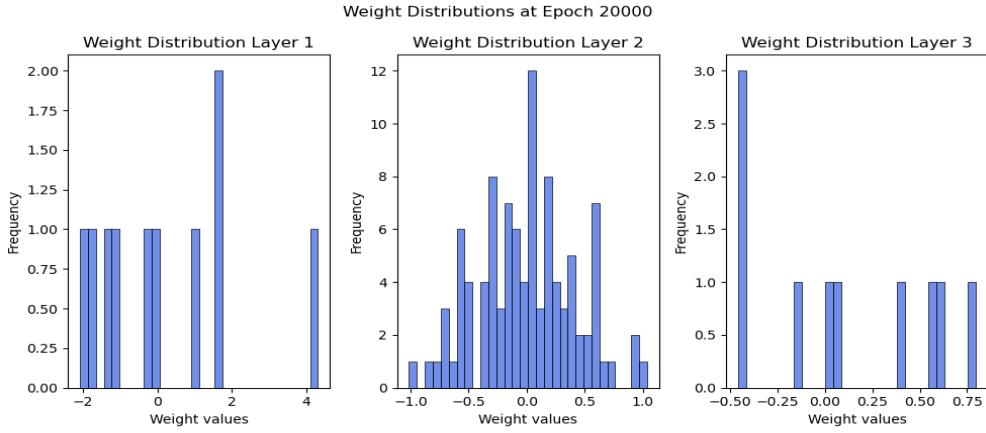


Rysunek 6: Porównanie zastosowania różnych współczynników uczenia dla tej samej architektury na zbiorze *Multimodal Large*. Wyniki MSE dla współczynników uczenia 0.1, 0.01, 0.001 to odpowiednio 132,17, 9.17, 523.07

Zaimplementowano również funkcje do wizualizacji wartości wag i ich wartości zmieniających się w czasie treningu - możliwa wizualizacja histogramów bądź heatmap, przedstawione poniżej w czasie treningu na zbiorze *Square Large*.



Rysunek 7: Wykres przedstawiający rozkład wag po 20 000 epokach treningu na zbiorze *Square Large*.



Rysunek 8: Wykres przedstawiający rozkład wag po 20 000 epokach treningu na zbiorze *Square Large*.

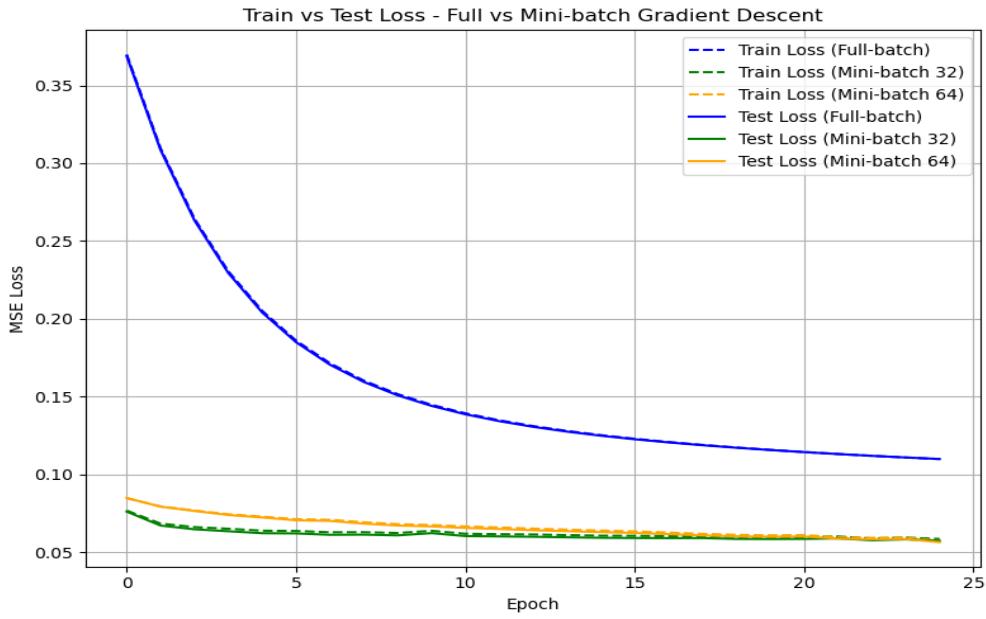
Wizualizacje przedstawiające rozkład wag w różnych etapach treningu (po 1 000 i 20 000 epokach) pokazują, jak zmienia się ich rozkład w miarę uczenia się modelu. Przy eksperymentach widać, że w początkowych etapach treningu wagi są rozproszone, a z czasem ich wartości stabilizują się, co sugeruje, że model skutecznie uczy się wzorców z danych.

2.2.3 Porównanie full batch i mini batch

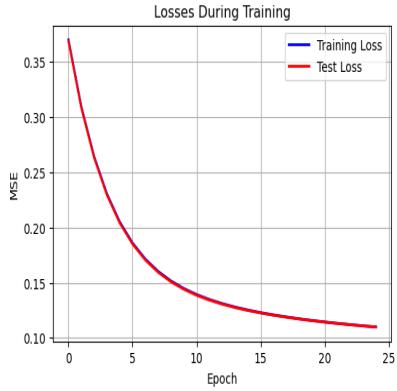
W full batchu dane treningowe są przetwarzane w całości przed aktualizowaniem parametrów sieci, co oznacza, że dokładny gradient obliczany jest raz na epokę (czyli po przejściu przez wszystkie dane). Może to prowadzić do wydłużenia czasu treningu, szczególnie w przypadku dużych zbiorów danych.

W mini-batchu (najczęściej 16, 32, 64, 128 itd.) do aktualizacji wag wykorzystywana jest część zbioru treningowego. Dzięki temu czas treningu jest krótszy, ponieważ parametry sieci neuronowej są aktualizowane częściej.

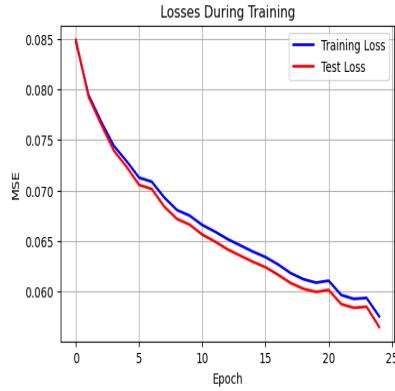
Przeprowadzono eksperymenty na zbiorze *Multimodal Large*, aby porównać wyniki trenowania z użyciem różnych rodzajów batchy - full batch, mini batch 32 oraz mini batch 64.



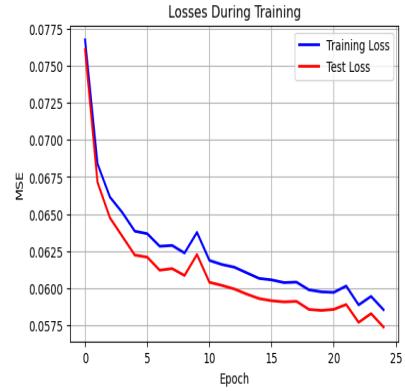
Rysunek 9: Wykres wartości funkcji straty dla zbioru treningowego i testowego z podziałem na rodzaj batcha.



Rysunek 10: Wykres wartości funkcji straty dla zbioru treningowego i testowego (full batch).



Rysunek 11: Wykres wartości funkcji straty dla zbioru treningowego i testowego (mini batch 64).



Rysunek 12: Wykres wartości funkcji straty dla zbioru treningowego i testowego (mini batch 32).

Z wykresu 42 wynika, że stosowanie mini-batchy przyspiesza zbieżność do optimum. Jednocześnie, na wykresie, widać, że pełny batch charakteryzuje się większą stabilnością procesu uczenia w porównaniu do mini-batchy, o czym świadczą mniejsze fluktuacje wartości funkcji straty podczas treningu. Dodatkowo, porównując czasy treningu dla 100 epok, czas dla pełnego batcha wyniósł 471,00 sekundy, podczas gdy dla mini-batchy 32 i 64 wynosił odpowiednio 32,02 oraz 40,64 sekundy.

2.2.4 Wnioski

Z przeprowadzonych eksperymentów wynika, że implementacja propagacji wstecznej błędu w sieci neuronowej skutecznie umożliwia modelowi uczenie się na podstawie danych. Uzyskane wyniki wskazują na dobrą zdolność sieci do dopasowania się do danych, co potwierdza wartość MSE na zbiorze testowym.

Dobór współczynnika uczenia do problemu i architektury sieci neuronowej jest kluczowy do stabilnego i zbieżnego procesu uczenia.

Wizualizacje rozkładu wag w trakcie treningu pokazują, jak model dostosowuje swoje parametry, co może świadczyć o efektywnym procesie uczenia. W szczególności, zmiana rozkładu wag w trakcie treningu sugeruje, że sieć skutecznie uczy się reprezentacji danych.

Porównanie metod treningowych (full batch vs mini batch) wykazuje, że mini-batchy przyspieszają proces zbieżności do optymalnych wag, co jest korzystne w przypadku dużych zbiorów danych. Jednocześnie, pełny batch zapewnia większą stabilność procesu uczenia, co może być przydatne w sytuacjach wymagających precyzyjnych i stabilnych aktualizacji wag. Dodatkowo, czasy treningu dla mini-batchy są znacznie krótsze niż w przypadku pełnego batcha, co pozwala na redukcję czasu obliczeń bez straty dokładności.

2.3 NN3: implementacja momentu i normalizacji gradientu

W ramach tego zadania celem było zaimplementowanie technik momentum oraz normalizacji gradientu (RMSProp) w procesie uczenia sieci neuronowej. Te rozszerzenia klasycznego algorytmu propagacji wstecznej mają na celu poprawę efektywności optymalizacji, przyspieszając zbieżność oraz zwiększać stabilność procesu trenowania.

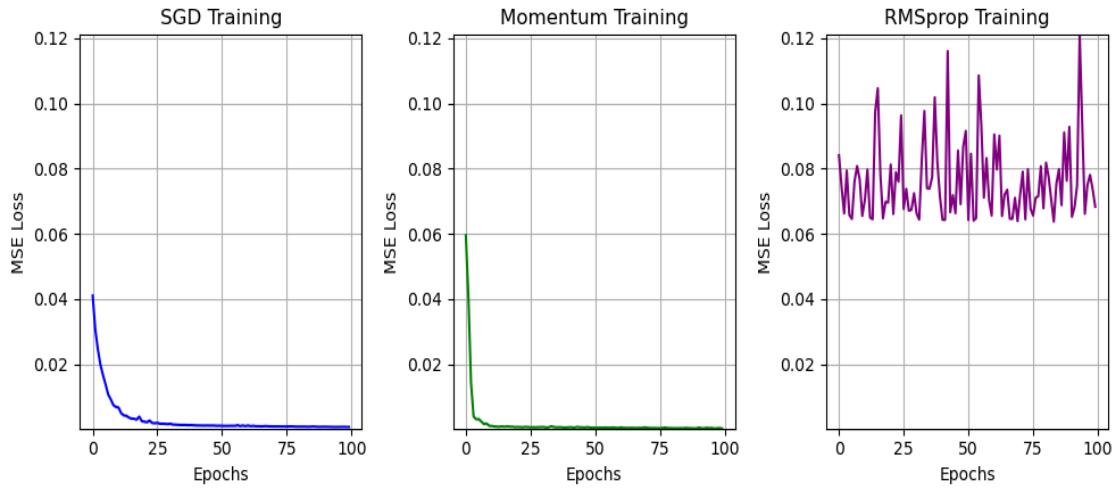
2.3.1 Eksperymenty

Przeprowadzono testy różnych optymalizatorów na wszystkich zbiorach danych dotyczących zadania. Wyniki zostały przedstawione w poniższej tabeli.

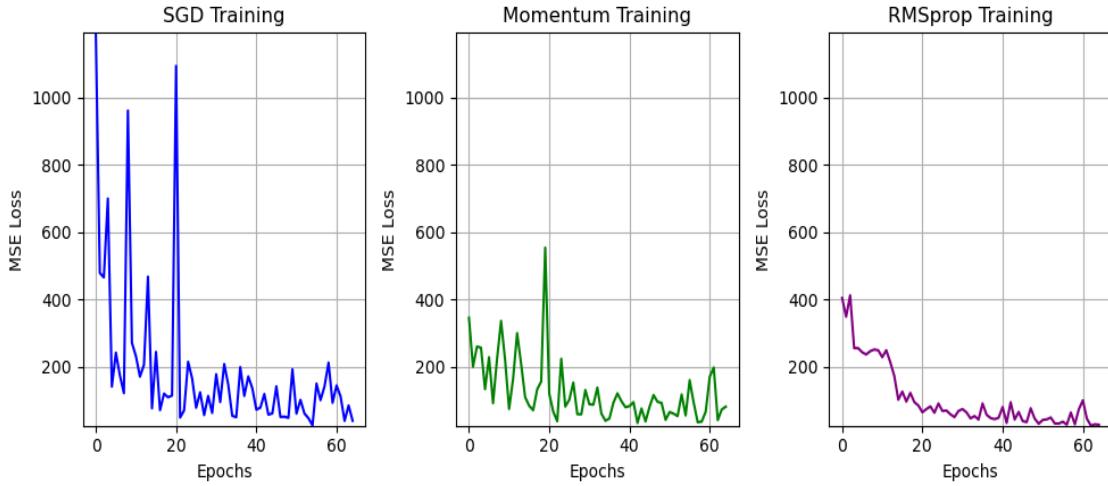
Zbiór danych	Brak optymalizatora (MSE)	Momentum (MSE)	RMSProp (MSE)
Square Large	17.03	4.76	30.95
Steps Large	16.95	23.50	10.01
Multimodal Large	141.29	6.86	34.06

Tabela 2: Wyniki optymalizacji dla różnych zbiorów danych i optymalizatorów.

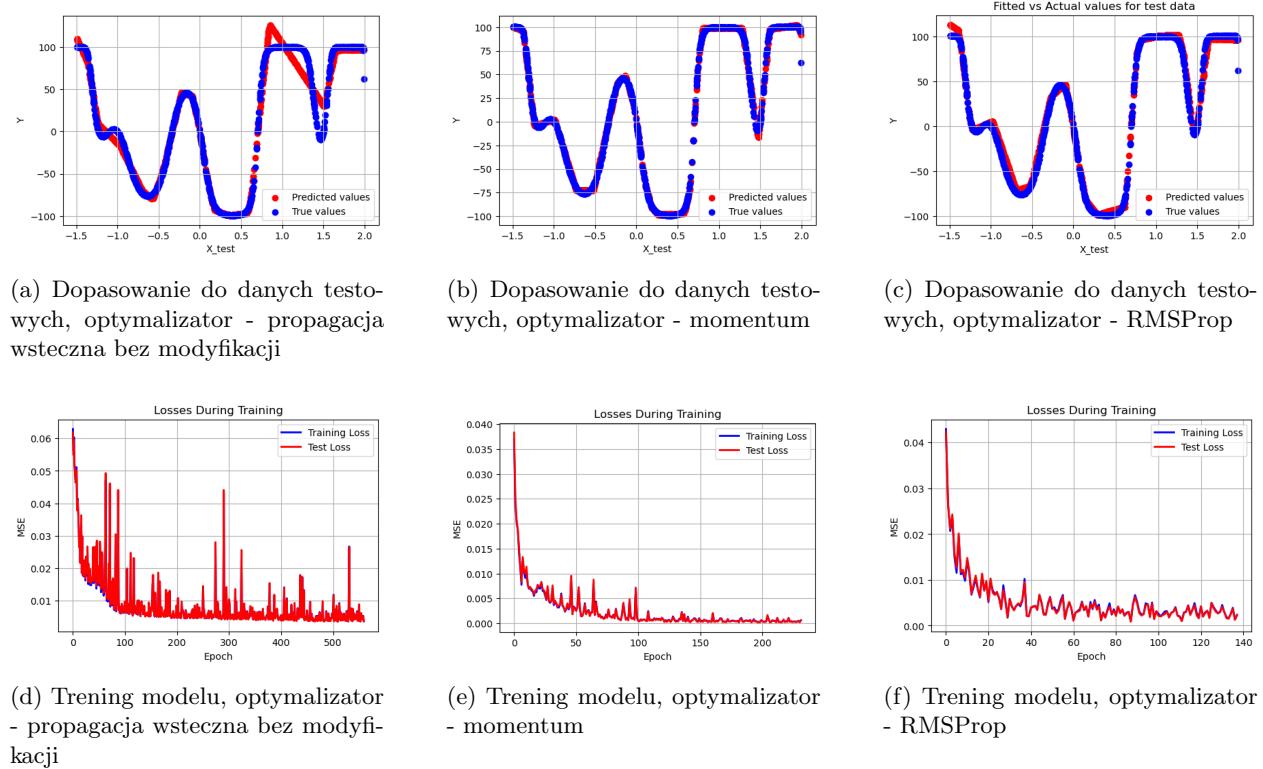
Porównano zachowanie algorytmów na różnych zbiorach danych, aby zobaczyć ich zachowania w zależności od skomplikowania i rodzaju danych treningowych w początkowych epokach. Wizualizacje procesu trenowania sieci dla poszególnych zbiorów danych wyglądały następująco:



Rysunek 13: Proces treningu sieci neuronowej na zbiorze danych *Sqaure Large* z wykorzystaniem różnych optymalizatorów w początkowych epokach. Obserwujemy brak zbieżności dla optymalizatora RMSProp.



Rysunek 14: Proces treningu sieci neuronowej na zbiorze danych *Steps Large* z wykorzystaniem różnych optymalizatorów w początkowych epokach.



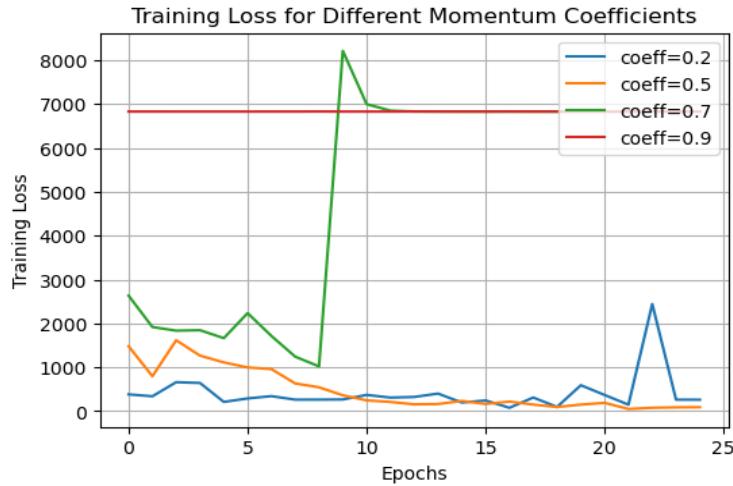
Rysunek 15: Zastosowanie różnych optymalizatorów na zbiorze *Multimodal Large*, note: zastosowany early stopping

Można zaobserwować, że wyniki treningu różnią się w zależności od wybranego optymalizatora. Czasami jeden optymalizator radzi sobie lepiej, a innym razem inny okazuje się bardziej efektywny. Jednak najważniejszym wnioskiem jest to, że sieć może uczyć się bardzo wolno lub w ogóle nie zbiegać, jeśli dobierzemy nieodpowiedni optymalizator do danego zadania.

Dla zbioru danych *Steps Large* zaobserwowano zjawisko związane z wpływem parametru momentum na proces uczenia się modelu. W przypadku zastosowania domyślnej wartości momentum wynoszącej około 0.9

(czyli silnego uwzględniania poprzednich gradientów w aktualizacji wag), model nie był w stanie się uczyć, a wartość błędu pozostawała na wysokim, praktycznie niezmiennym poziomie.

Natomiast po obniżeniu momentum do 0.5 — co oznacza, że aktualizacje wag są w dużej mierze oparte na bieżącym gradiencie — sieć zaczęła poprawnie się uczyć i skutecznie dopasowywać do danych. Sugeruje to, że zbyt duży wpływ przeszłych gradientów mógł przeszkadzać w dotarciu do odpowiedniego minimum funkcji straty w tym konkretnym przypadku.



Rysunek 16: Proces treningu sieci neuronowej na zbiorze danych *Steps Large* z wykorzystaniem różnych wartości momentum.

Zbiór danych	Momentum = 0.2 (MSE)	Momentum = 0.5 (MSE)	Momentum = 0.7 (MSE)	Momentum = 0.9 (MSE)
Steps Large	313.68	121.92	6883.98	6883.94

Tabela 3: Wpływ różnych wartości parametru *momentum* na błąd MSE na zbiorze *Steps Large* (w początkowych epokach treningu).

2.3.2 Wnioski

Dobór odpowiedniego optymalizatora oraz jego parametrów ma istotny wpływ na efektywność procesu uczenia sieci neuronowej. W wielu przypadkach wykorzystanie klasycznej propagacji wstecznej w połączeniu z podstawowym wariantem optymalizacji propagacji wstecznej, może okazać się niewystarczające – model może trenować się bardzo wolno lub w ogóle nie zbiegać do rozwiązania.

Dlatego warto eksperymentować z bardziej zaawansowanymi technikami optymalizacji. Równie ważne jak sam wybór algorytmu, jest także odpowiednie dostrojenie jego parametrów – takich jak współczynnik uczenia czy współczynnik momentum. Jak pokazały przeprowadzone eksperymenty, niewielka zmiana wartości jednego parametru może znaczco wpływać na jakość i szybkość uczenia. Przykładowo, dla zbioru *Steps Large* zastosowanie domyślnego momentum (0.9) całkowicie uniemożliwiło efektywne trenowanie modelu, podczas gdy zmniejszenie go do 0.5 pozwoliło sieci osiągnąć dobre dopasowanie już po kilku epokach.

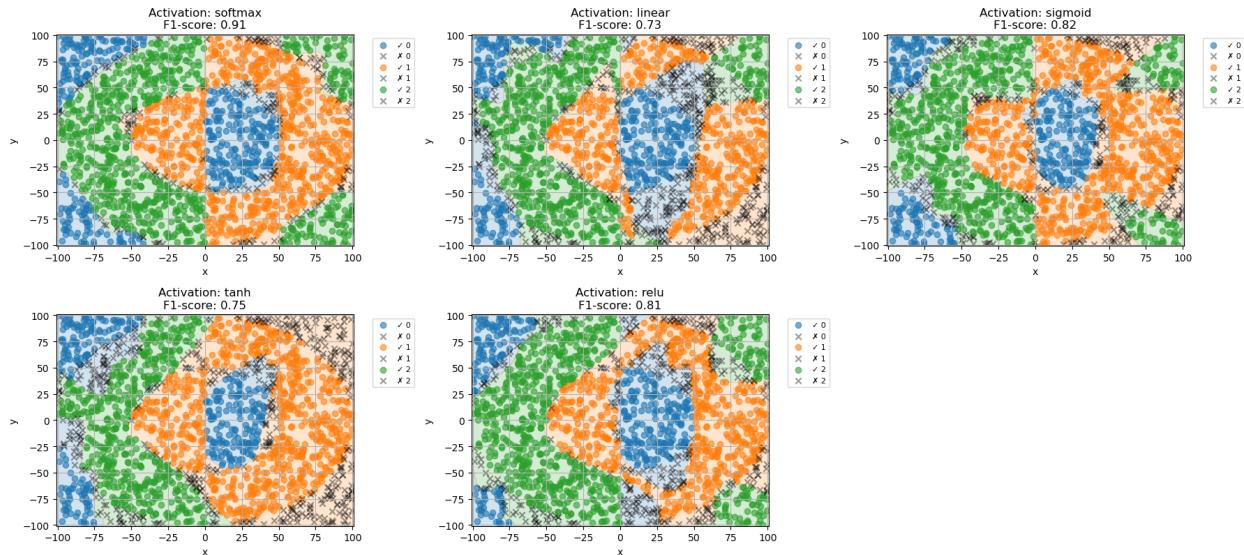
2.4 NN4: zadania klasyfikacji

Aby umożliwić dokładne przewidywania w zadaniach klasyfikacji, w ostatniej warstwie sieci zaimplementowano funkcję softmax (dotychczas liniowa w przypadku regresji). Funkcja Softmax przekształca wyniki z ostatniej warstwy sieci do wartości mieszczących się w przedziale $[0, 1]$, które sumują się do 1. Dzięki temu można je interpretować jako prawdopodobieństwa przynależności danych do poszczególnych klas. Jako funkcję strat zastosowano Cross-Entropy loss (z modyfikacją clip z epsilon przewidywanej wartości dla klasy aby uniknąć $\log(0)$). Przed przekazaniem do sieci zbiorów treningowych i testowych zastosowano one-hot-encoding przy wczytywaniu danych. Jako miarę do oceny dopasowania modelu wybrano F-score w wersji micro.

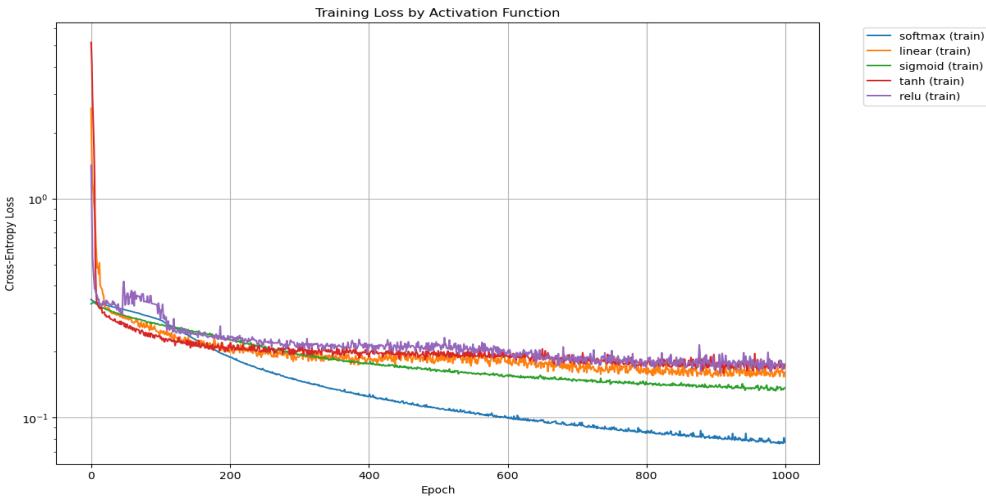
2.4.1 Eksperymenty

W celu weryfikacji efektywności funkcji softmax w zadaniach klasyfikacji, przeprowadzono eksperymenty na zbiorach danych. Przetestowano wpływ różnych funkcji aktywacji w warstwie wyjściowej, stabilność procesu uczenia mierzoną wartością funkcji kosztu, jakość klasyfikacji ocenianą metryką Fscore oraz czas zbieżności modelu.

Przeprowadzono analizę różnych funkcji aktywacji w warstwie wyjściowej na zbiorze danych *Rings3 Regular*. Testy objęły zarówno funkcje typowo stosowane w klasyfikacji (softmax), jak i rozwiązania alternatywne. Wyniki przedstawiono na Rysunku 17.



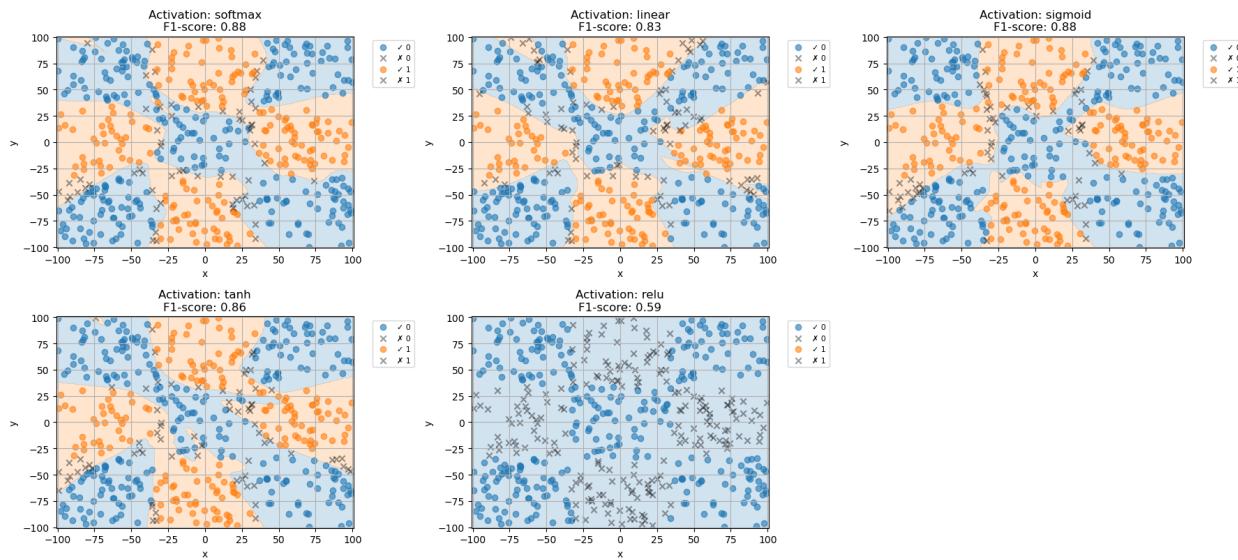
Rysunek 17: Porównanie skuteczności różnych funkcji aktywacji w warstwie wyjściowej. Wartości F1-score dla poszczególnych konfiguracji: softmax (0.91), ReLU (0.81), sigmoid (0.82), tanh (0.75), linear (0.73).



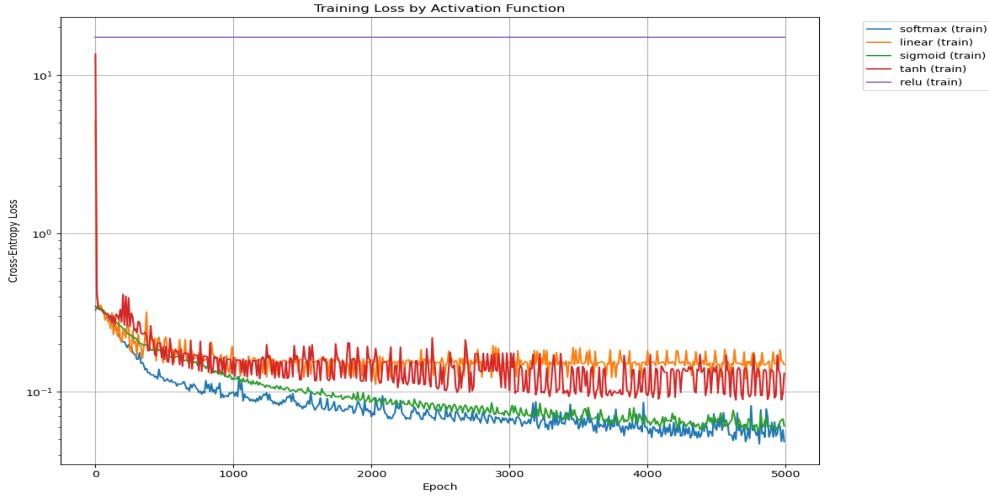
Rysunek 18: Krzywe uczenia dla różnych funkcji aktywacji wyjściowej. Linia niebieska (softmax) wykazuje najszybszą zbieżność i najniższą końcową wartość funkcji stratu.

Eksperymenty jednoznacznie wykazały przewagę funkcji softmax w warstwie wyjściowej, która osiągnęła najwyższą dokładność (F1-score 0.91). W przeciwieństwie do rozwiązań alternatywnych (liniowa: 0.73, tanh: 0.75), softmax zapewnia najbardziej wyraźne granice decyzyjne.

Analogiczne porównanie dla zbioru *XOR3*



Rysunek 19: Porównanie skuteczności różnych funkcji aktywacji w warstwie wyjściowej. Wartości Fscore dla poszczególnych konfiguracji: softmax (0.88), ReLU (0.59), sigmoid (0.88), tanh (0.86), linear (0.83).



Rysunek 20: Krzywe uczenia dla różnych funkcji aktywacji wyjściowej. Linia niebieska (softmax) wykazuje najszybszą zbieżność i najniższą końcową wartość funkcji straty.

Wszystkie funkcje aktywacji oprócz ReLU uzyskały wysokie Fscore, zbliżone do wyników softmaxu, który dodatkowo okazał się najstabilniejszy.

Na każdym zbiorze zaproponowanym w zadaniu przeprowadzono trening sieci neuronowej do zadania klasyfikacji. Na wszystkich zbiorach danych sieci osiągały bardzo dobre wyniki metryki Fscore i szybką zbieżność.

Zbiór danych	F1 Score	Czas uczenia [s]	Liczba epok do zbieżności
Rings3 Regular	0.97	25.51	2000
Easy	0.99	2.11	1000
Xor	0.97	9.72	2000

Tabela 4: Porównanie skuteczności klasyfikacji dla różnych zbiorów danych z funkcją softmax na wyjściu.

2.4.2 Wnioski

Zastosowanie funkcji softmax w warstwie wyjściowej poprawia jakość klasyfikacji, zapewniając lepsze dopasowanie do rozkładu klas w danych. W porównaniu z alternatywnymi funkcjami aktywacji, softmax wykazał się nie tylko wyższą dokładnością predykcji, ale także większą stabilnością procesu uczenia. Dodatkowo, kombinacja softmax z funkcją straty cross-entropy zapewniła optymalne warunki dla propagacji gradientu, prowadząc do szybkiej i stabilnej zbieżności.

2.5 NN5: testowanie różnych funkcji aktywacji

Kolejnym zadaniem było testowanie architektur sieci i porównywanie liczby warstw ukrytych, liczby neuronów w warstwach ukrytych i użytych funkcji aktywacji w kontekście porównania skuteczności i szybkości uczenia się sieci neuronowej. Przeprowadzono eksperymenty z funkcjami aktywacji, takimi jak: liniowa, sigmoidalna (sigmoid), tangens hiperboliczny (tanh) oraz funkcja ReLU. Niezmienne na wyjściu w przypadku zadań regresji jest funkcja liniowa, a dla zadań klasyfikacji softmax.

W ramach testów porównano czas trenowania oraz skuteczność modelu na zbiorze testowym, aby sprawdzić, która kombinacja architektury sieci i funkcji aktywacji prowadzi do najlepszych rezultatów.

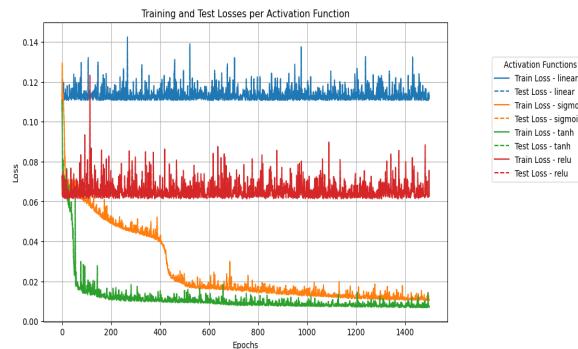
2.5.1 Eksperymenty

Zaimplementowano metodę do przeprowadzania symulacji treningu sieci neuronowej w zależności od architektury. Tworzy modele sieci neuronowej w zależności od zdefiniowanej architektury, monitoruje proces trenowania, porównuje wyniki biorąc pod uwagę czas trenowania i skuteczność modelu na zbiorze testowym, a także wizualizuje dokładność predykcji na zbiorze testowym.

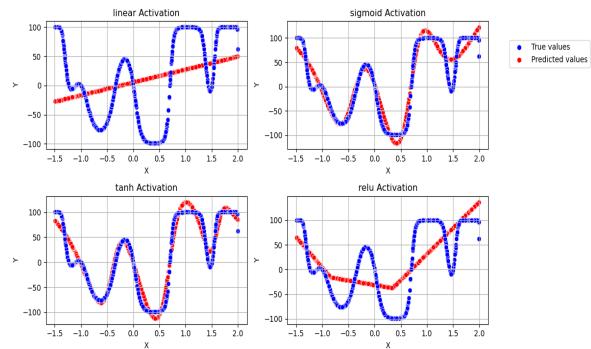
Logika przeprowadzanych eksperymentów: zwiększamy liczbę warstw z niewielką liczbą neuronów i sprawdzamy czy dodanie potem większej liczby neuronów wyraźnie wpływa na wyniki.

Testy na zbiorze *Multimodal Large*

- 5 neuronów w warstwie ukrytej



Rysunek 21: Krzywa treningu dla 1 warstwy ukrytej z 5 neuronami.



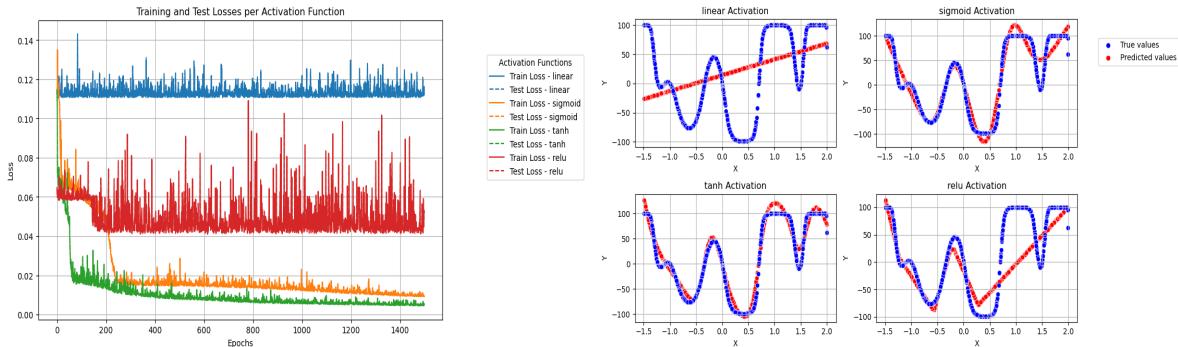
Rysunek 22: Przewidywanie sieci neuronowej dla 1 warstwy ukrytej z 5 neuronami.

Eksperyment ujawnił wyraźny podział na efektywne (sigmoid, tanh) i mało skuteczne (linear, ReLU) funkcje aktywacji w kontekście tego problemu. Podczas gdy pierwsza grupa osiągnęła satysfakcyjny poziom dopasowania, druga wykazała się znaczco wyższym błędem, sugerując brak dostatecznych zdolności uczenia sieci.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	4473.45	155.32
Sigmoid	418.38	178.28
Tanh	269.96	187.43
ReLU	2480.64	175.36

Tabela 5: Porównanie skuteczności regresji (MSE) i czasu uczenia dla różnych funkcji aktywacji

- 20 neuronów w warstwie ukrytej



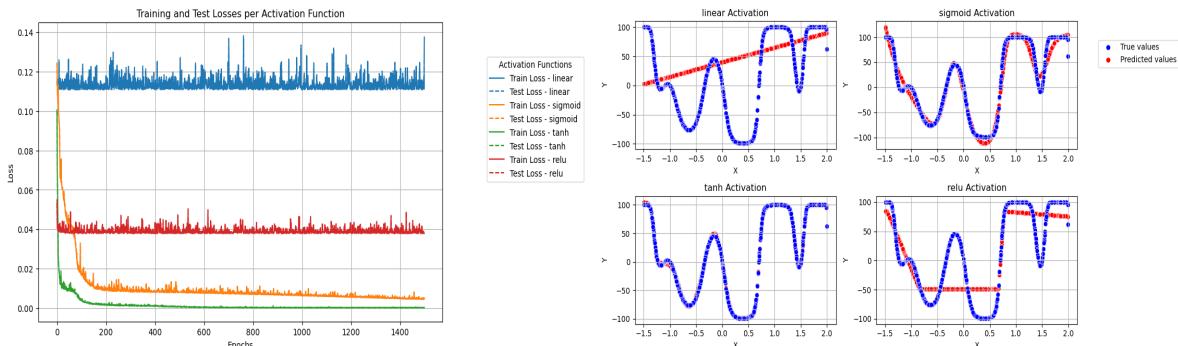
Rysunek 23: Krzywa treningu dla 1 warstwy ukrytej z 20 neuronami.
Rysunek 24: Przewidywania sieci neuronowej dla 1 warstwy ukrytej z 20 neuronami.

Tanh pozostaje najbardziej stabilną funkcją aktywacji niezależnie od rozmiaru warstwy. Większa pojemność modelu (20 neuronów) pozwala lepiej wykorzystać potencjał funkcji ReLU, choć wciąż ustępuje ona tanh/sigmoid.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	4486.25	571.14
Sigmoid	366.28	700.16
Tanh	210.93	664.26
ReLU	2048.28	593.17

Tabela 6: Analiza efektywności różnych funkcji aktywacji. Funkcja tanh (MSE = 210.93) i sigmoid (MSE = 366.28) wykazały się najlepszą zdolnością predykcyjną, podczas gdy funkcja liniowa była najszybsza, lecz najmniej dokładna. ReLU, pomimo pośredniego czasu uczenia, osiągnęła stosunkowo słabe wyniki MSE.

- 2 warstwy ukryte, 5 neuronów w każdej warstwie



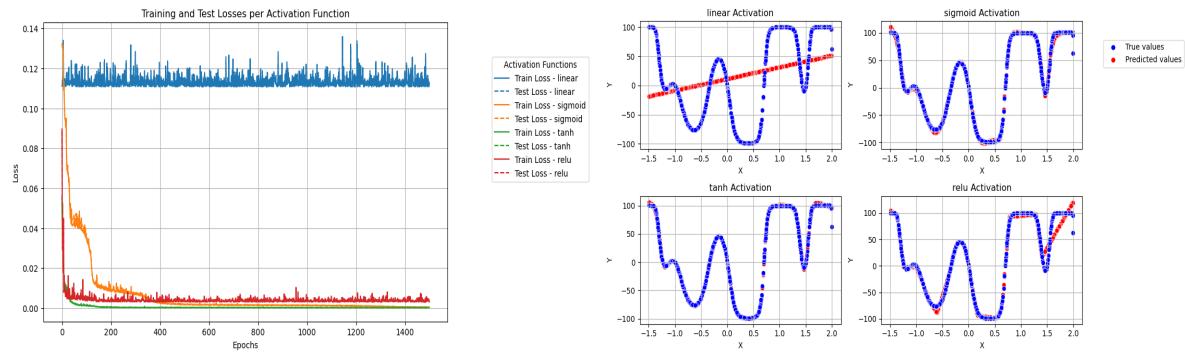
Rysunek 25: Krzywa treningu dla 2 warstw ukrytych z 5 neuronami.
Rysunek 26: Przewidywania sieci neuronowej dla 2 warstw ukrytych z 5 neuronami.

Podobne obserwacje do poprzedniego eksperymentu z architekturą z 1 warstwą ukrytą z 5 neuronami, sigmoid i tanh jako aktywacje radzą sobie najlepiej, warto zwrócić uwagę na już bardzo niską warstość MSE dla tanh, która wynosi 6.78. Zgodnie z przewidywaniem czasu treningów się wydłużają, ale głębsza sieć w tym przypadku prowadzi do lepszej predykcji.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	5471.75	176.88
Sigmoid	190.64	265.42
Tanh	6.78	242.81
ReLU	1518.02	235.92

Tabela 7: Porównanie skuteczności regresji (MSE) i czasu uczenia dla różnych funkcji aktywacji. Wyróżniono najlepszy wynik MSE.

• 3 warstwy ukryte, 5 neuronów w każdej warstwie



Rysunek 27: Krzywa treningu dla 3 warstw ukrytych z 5 neuronami. Rysunek 28: Przewidywanie sieci neuronowej dla 3 warstw ukrytych z 5 neuronami.

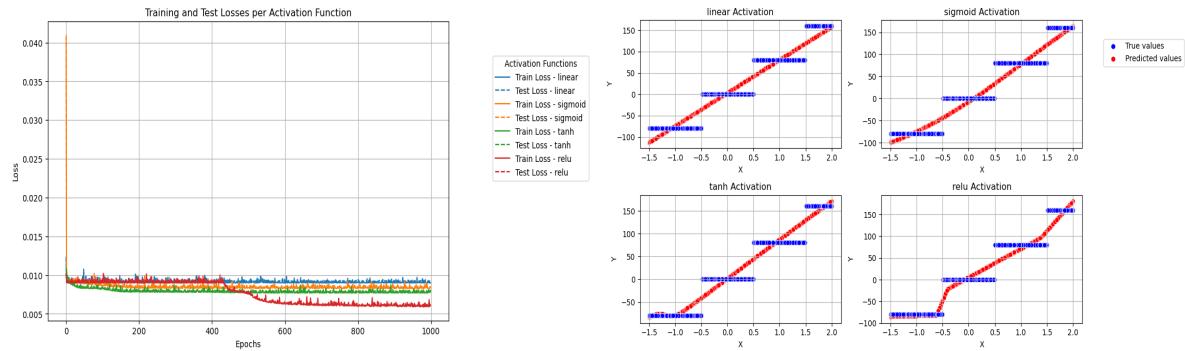
Zastosowanie głębszej architektury sieci przyniosło lepsze wyniki, co jest zgodne z wcześniejszymi obserwacjami dotyczącymi skuteczności funkcji aktywacji takich jak tanh oraz sigmoid. Warto jednak zauważyć, że funkcja aktywacji ReLU bardzo dobrze dopasowała się do tej architektury, co może świadczyć o jej wysokiej skuteczności w przypadku głębszych sieci neuronowych. Zwraca również uwagę znacznie dłuższy czas treningu dla funkcji sigmoid w porównaniu do pozostałych funkcji aktywacji.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	4499.77	242.79
Sigmoid	15.91	389.65
Tanh	5.19	290.24
ReLU	124.09	265.31

Tabela 8: Porównanie skuteczności regresji (MSE) i czasu uczenia dla różnych funkcji aktywacji. Ciekawe wyniki pogrubione.

Przeprowadzono podobne eksperymenty na zbiorze *Steps Large*

- **10 neuronów w warstwie ukrytej**



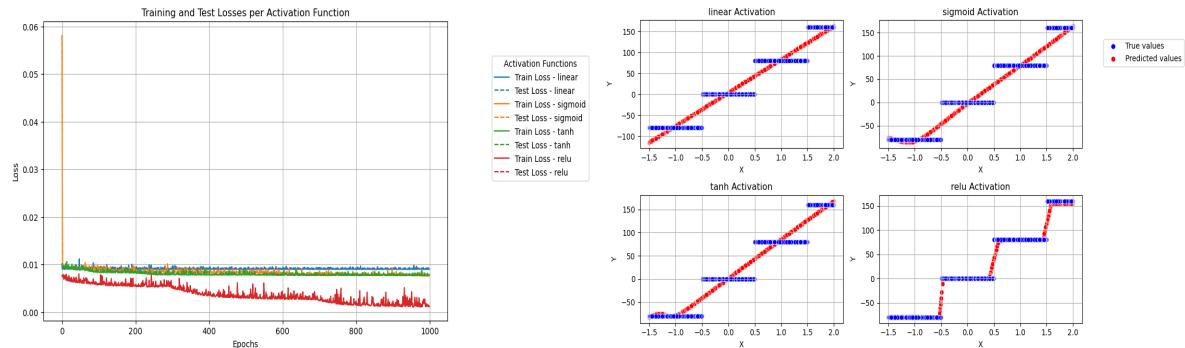
Rysunek 29: Krzywa treningu dla 1 warstwy ukrytej z 10 neuronami.
Rysunek 30: Przewidywania sieci neuronowej dla 1 warstwy ukrytej z 10 neuronami.

Analiza wyników wykazała ograniczone możliwości predykcyjne testowanej architektury niezależnie od zastosowanej funkcji aktywacji. Wśród badanych wariantów wyróżniła się funkcja ReLU, osiągając względnie najlepsze rezultaty, podczas gdy funkcja liniowa odznaczała się najkrótszym czasem uczenia.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	528.91	191.93
Sigmoid	494.22	300.43
Tanh	467.21	385.87
ReLU	352.66	396.65

Tabela 9: Porównanie skuteczności regresji (MSE) i czasu uczenia dla różnych funkcji aktywacji.

- **1 warstwy ukryta, 20 neuronów warstwie**



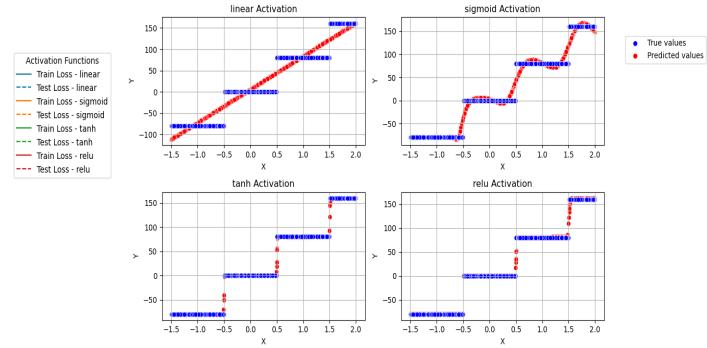
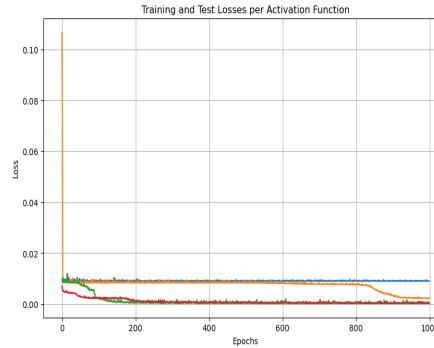
Rysunek 31: Krzywa treningu dla 1 warstwy ukrytej z 20 neuronami.
Rysunek 32: Przewidywania sieci neuronowej dla 1 warstwy ukrytej z 20 neuronami.

W tym przypadku zaobserwowano już wyraźniejsze dopasowanie modelu z funkcją aktywacji ReLU, co potwierdzają niższe wartości błędu MSE na zbiorze testowym. Może to sugerować, że ReLU lepiej sprawdza się w przypadku modeli o większej liczbie neuronów lub głębszej architekturze.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	530.65	208.1
Sigmoid	448.15	269.68
Tanh	459.35	400.13
ReLU	69.34	378.41

Tabela 10: Porównanie efektywności różnych funkcji aktywacji. Funkcja ReLU osiągnęła znacząco lepsze wyniki ($MSE = 69.34$) w porównaniu do pozostałych funkcji. Warto zauważyć, że funkcja liniowa, pomimo najkrótszego czasu obliczeń, wykazała się najsłabszą zdolnością predykcyjną.

- **3 warstwy ukryte, 10 neuronów w każdej warstwie**



Rysunek 33: Krzywa treningu dla 3 warstw ukrytych z 10 neuronami. Rysunek 34: Przewidywania sieci neuronowej dla 3 warstw ukrytych z 10 neuronami.

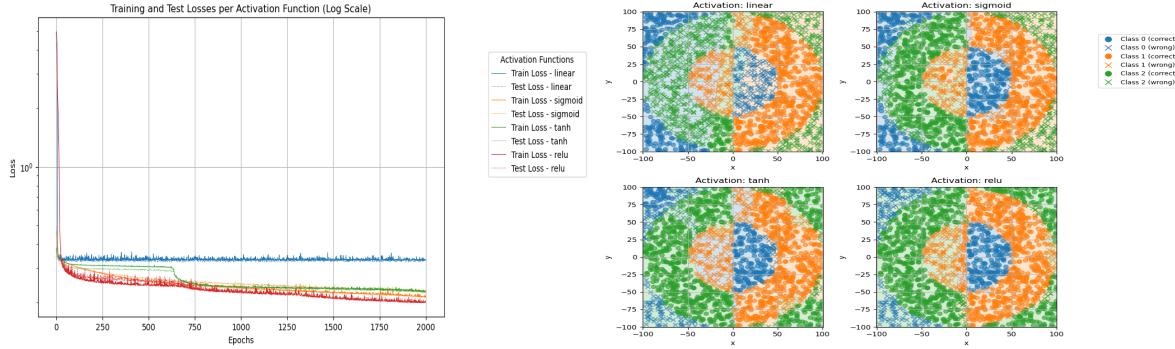
Głębsza architektura sieci osiągnęła lepsze dopasowanie do danych od poprzednich architektur. Funkcje tanh i ReLU osiągnęły porównywalne wyniki znacząco przewyższając pozostałe funkcje, choć widać, że przy sigmoidzie sieć zaczyna się lepiej dopasowywać niż poprzednio. Warto zauważyć, że ReLU charakteryzuje się najkrótszym czasem uczenia wśród skutecznych funkcji aktywacji. Podobne obserwacje były dla architektury z 2 warstwami ukrytymi z 10 neuronami.

Funkcja aktywacji	MSE	Czas uczenia [s]
Linear	530.36	530.06
Sigmoid	124.76	555.07
Tanh	15.49	520.37
ReLU	15.97	504.34

Tabela 11: Porównanie efektywności funkcji aktywacji w głębszej architekturze sieci.

Eksperymenty dla zadania klasyfikacji na zbiorze *Rings3*

- **5 neuronów w warstwie ukrytej**



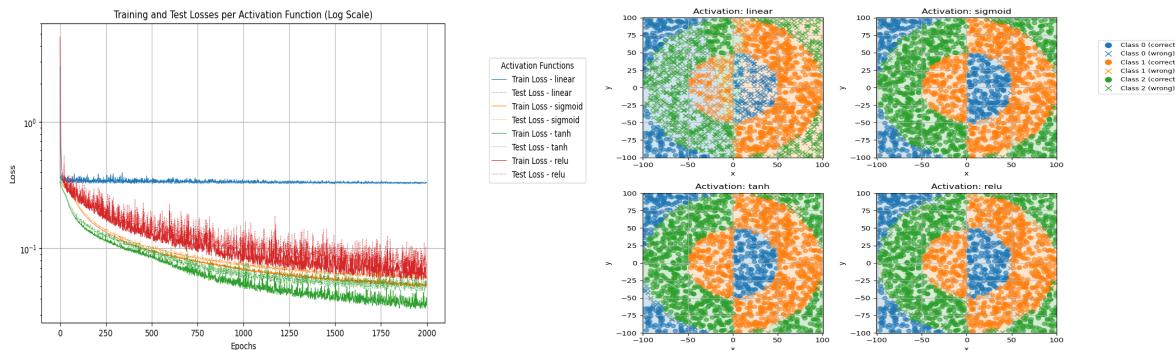
Rysunek 35: Krzywa treningu dla 1 warstwy ukrytej z 5 neuronami. Rysunek 36: Przewidywania sieci neuronowej dla 1 warstwy ukrytej z 5 neuronami.

Wszystkie funkcje aktywacji osiągnęły porównywalny wynik Fscore na zbiorze testowym, wynoszący około 0.7. Taki wynik nie jest zadowalający, ponieważ nie przekłada się on na dobre dopasowanie danych testowych.

Funkcja aktywacji	Fscore	Czas uczenia [s]
Linear	0.42	17.27
Sigmoid	0.72	23.11
Tanh	0.71	19.67
ReLU	0.74	19.51

Tabela 12: Porównanie skuteczności klasyfikacji (Fscore) i czasu uczenia dla różnych funkcji aktywacji.

- **1 warstwy ukryta, 20 neuronów warstwie**



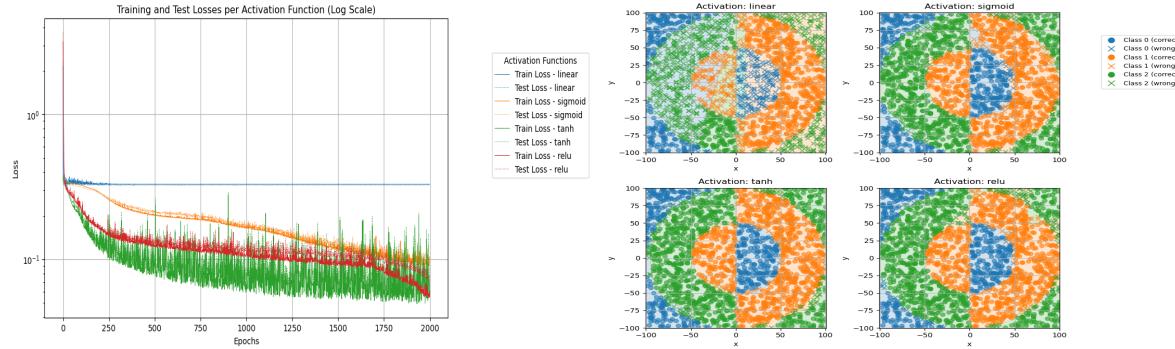
Rysunek 37: Krzywa treningu dla 1 warstwy ukrytej z 20 neuronami. Rysunek 38: Przewidywania sieci neuronowej dla 1 warstwy ukrytej z 20 neuronami.

Dla modelu z większą liczbą parametrów można wyciągnąć podobne wnioski jak poprzednio, jednak tym razem Fscore dla funkcji aktywacji (poza liniową) jest zdecydowanie większy co ma odzwierciedlenie w wykresie obrazującym przewidywania sieci neuronowej. Analogicznie sytuacja wyglądała dla architektur z 2 warstwami ukrytymi z ilością neuronów zaprezentowanych jak w tych przykładach.

Funkcja aktywacji	Fscore	Czas uczenia [s]
Linear	0.42	14.56
Sigmoid	0.94	20.39
Tanh	0.93	18.62
ReLU	0.91	15.9

Tabela 13: Porównanie skuteczności klasyfikacji (Fscore) oraz czasu uczenia dla różnych funkcji aktywacji.

- **3 warstwy ukryte, 5 neuronów w każdej warstwie**



Rysunek 39: Krzywa treningu dla 3 warstw ukrytych z 5 neuronami.
Rysunek 40: Przewidywanie sieci neuronowej dla 3 warstw ukrytych z 5 neuronami.

Głębsza architektura sieci osiągnęła lepsze dopasowanie do danych od poprzednich architektur. Podobne obserwacje były dla architektury z 3 warstwami ukrytymi z 10 neuronami, lecz wtedy architektura używająca sigmoid jako funkcję aktywacji osiągnęła Fscore na poziomie 0.74, więc prawdopodobnie doszło tam do overfittu, pozostałe funkcje podobnie do przykładu tej architektury.

Funkcja aktywacji	Fscore	Czas uczenia [s]
Linear	0.43	27.95
Sigmoid	0.90	37.74
Tanh	0.92	25.91
ReLU	0.91	24.18

Tabela 14: Porównanie skuteczności klasyfikacji (Fscore) oraz czasu uczenia dla różnych funkcji aktywacji.

2.5.2 Wnioski

Głębsze architektury prowadzą do dłuższego czasu treningu, ale również poprawy zdolności predykcyjnych modelu. Większa liczba neuronów w warstwie ukrytej poprawia zdolność modelu do uchwycenia bardziej skomplikowanych zależności w danych. W testach na większej liczbie neuronów zauważono poprawę w skuteczności predykcji, zwłaszcza w połączeniu z funkcją aktywacji ReLU i tanh. Należy jednak pamiętać o overfitting, jak w przypadku zbioru *Rings3* i funkcji sigmoid. Najlepsze wyniki w zakresie dokładności predykcji osiągnięto przy użyciu funkcji tanh, zwłaszcza w głębszych sieciach. Sigmoid również dawał dobre rezultaty, ale z wyraźnie dłuższym czasem uczenia. ReLU sprawdziło się w głębszych sieciach, ale nie była to najefektywniejsza funkcja w przypadku małych architektur. Funkcja liniowa okazała się najsłabsza pod względem dokładności.

2.6 NN6: zjawisko przeuczenia i regularyzacja

Zadanie polegało na zaimplementowaniu mechanizmu regularyzacji wag w sieci oraz mechanizmu zatrzymywania uczenia przy wzroście błędu na zbiorze walidacyjnym (early stopping).

Regularizacja polega na dodaniu kary w loss function (kara za duże wartości wag), aby zapobiegać nadmiernemu dopasowaniu się sieci do danych treningowych. Zaimplementowano regularyzacje L1 i L2, różnica polega na sposobie obliczania kary (L1 bierzemy sumę wartości absolutnych wag, L2 suma kwadratów wartości wag).

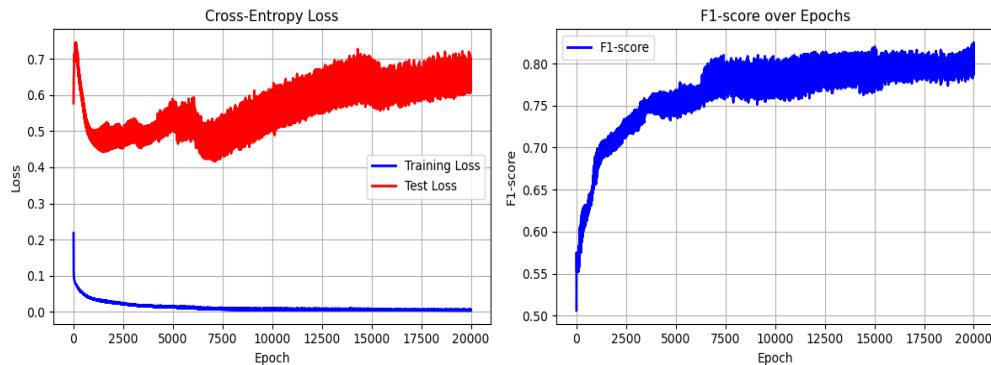
Early stopping monitoruje funkcje straty na zbiorze testowym i przerywa proces uczenia w momencie, gdy przez określoną liczbę epok nie obserwuje się poprawy wyników.

2.6.1 Eksperymenty

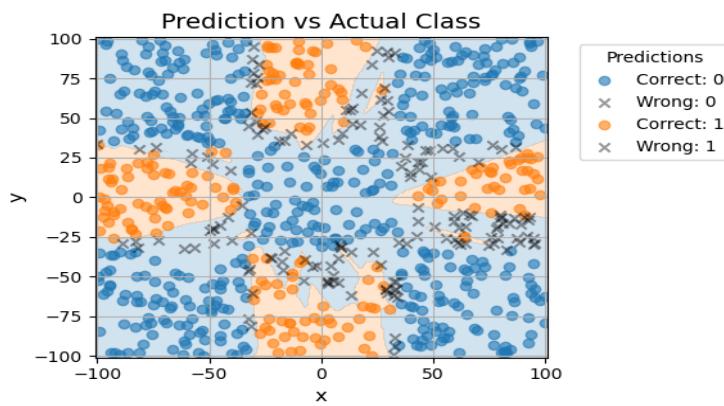
Na każdym z wykorzystanych zbiorów danych przeprowadzono trening sieci neuronowej w kilku konfiguracjach: bez regularyzacji i bez early stoppingu, z zastosowaniem early stoppingu, a także z regularyzacją L1 i L2 dla różnych wartości parametrów regularizacyjnych.

Wyniki dla zbioru *XOR*

Trening sieci neuronowej bez zastosowania technik ograniczających przeuczenie prowadzi do overfittingu — mimo dalszej poprawy wyników na zbiorze treningowym, wartość funkcji straty na zbiorze testowym zaczyna rosnąć.



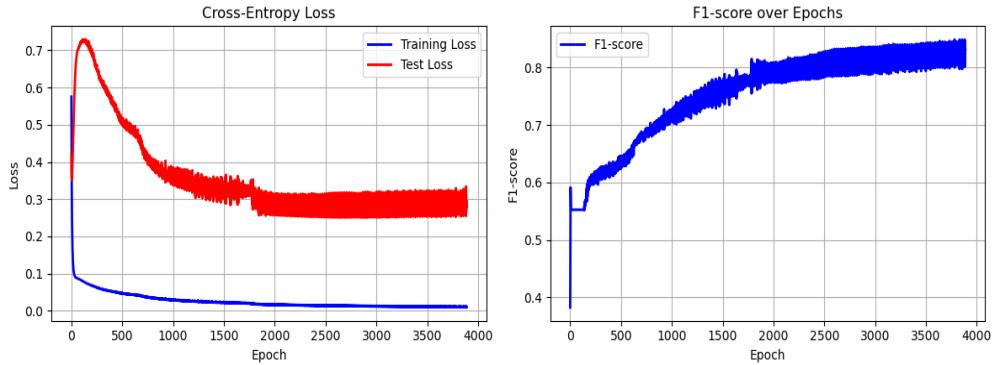
Rysunek 41: Wykres wartości funkcji straty dla zbioru treningowego i testowego oraz wartości Fscore na zbiorze testowym bez regularyzacji i early stoppingu.



Rysunek 42: Predykcje modelu na zbiorze testowym z sieci bez regularyzacji i early stoppingu. Fscore=0.8.

Następnie przetestowano model z early stoppingiem. Sieć była oryginalnie trenowana przez 20 000 epok, więc patience został ustalony na 1000. Osiągnięty Fscore na zbiorze testowym to 0.84, co jest lepszym

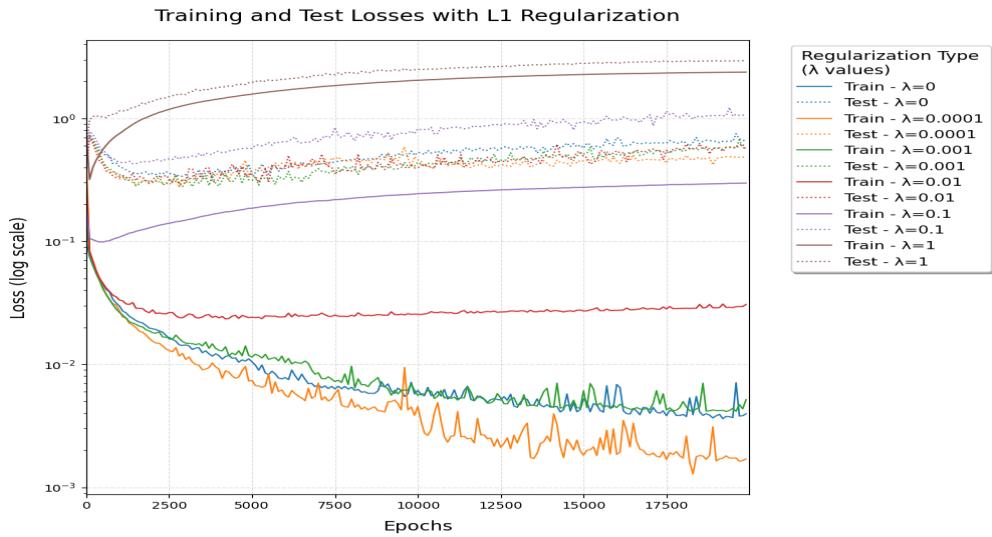
wynikiem niż w poprzednim przykładzie sieci bez early stoppingu. Sieć została nauczona na 5 000 epokach, co przyczyniło się również do krótszego czasu obliczeń.



Rysunek 43: Wykres wartości funkcji sterty dla zbioru treningowego i testowego oraz wartości Fscore na zbiorze testowym z early stoppingiem.

Zastosowano również regularyzacje L1 i L2, eksperymentując z parametrymi.

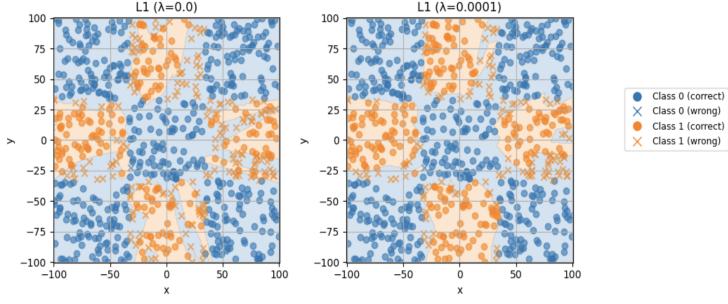
Dla regularyzacji L1 eksperyment z parametrami 0.0001, 0.001, 0.01, 0.1, 1.00 wykazał poprawę predykcji sieci dla każdego z testowanych parametrów. Początkowa sieć neuronowa bez regularyzacji (parametr regularyzacji 0.00) osiągnęła Fscore na poziomie 0.78, natomiast przy użyciu regularyzacji L1 z parametrem 0.0001 osiągnięto Fscore 0.85. Na Rysunku 46 widać również jak zmieniły się dopasowania do danej klasy. Można również zaobserwować, że sieci trenowane z regularizacją wag mają większe wartości funkcji sterty podczas treningu, co jest spodziewane, ponieważ do funkcji straty dodajemy kary za wielkość wag.



Rysunek 44: Wykres wartości funkcji sterty dla zbioru treningowego i testowego z regularizacją L1.

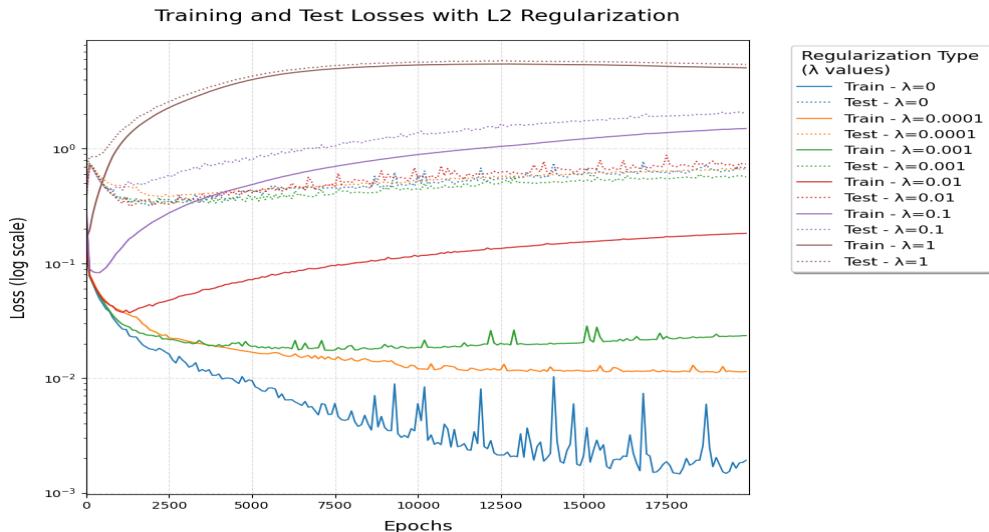
Wartość λ	Fscore
0.00 (brak regul.)	0.78
0.0001	0.85
0.001	0.82
0.01	0.83
0.1	0.81
1.00	0.83

Rysunek 45: Skuteczność klasyfikacji dla różnych wartości regularyzacji L1



Rysunek 46: Predykcje modelu dla $\lambda = 0$ i $\lambda = 0.0001$

W przypadku regularyzacji L2 przetestowano taki sam zestaw parametrów jak w przypadku regularyzacji L1. Również w tym przypadku dla wyniki metryki Fscore były lepsze dla każdej zastosowanej regularyzacji niż dla modelu bez regularyzacji.



Rysunek 47: Wykres wartości funkcji starty dla zbioru treningowego i testowego z regularyzacją L2.

Dla każdej wartości parametru regularyzacji osiągnięto wynik Fscore większy niż na oryginalnej sieci.

Wartość λ	Fscore
0.00 (brak regul.)	0.78
0.0001	0.81
0.001	0.83
0.01	0.84
0.1	0.82
1.00	0.84

Tabela 15: Skuteczność klasyfikacji (Fscore) dla różnych wartości parametru regularyzacji L2. Najlepsze wyniki osiągnięto dla $\lambda = 0.01$ i $\lambda = 1.00$.

Jednak nie na wszystkich zbiorach danych dotyczących zadania regularyzacja poprawiła zdolności predykcyjne modelu. Na zbiorze *Rings5* żadna regularyzacja nie osiągnęła sukcesu, natomiast zastosowanie early stoppingu zwiększyło Fscore z 0.77 do 0.82. W przypadku zbioru *Multimodal Large* regularyzacja L2

zmnieszyła MSE na zbiorze treningowym, a early stopping spowodował redukcje czasu obliczeniowego (z 50 000 epok do 5 około 5 000)i zwiększenie dokładności sieci (MSE 95.69 do MSE 85.53).

2.6.2 Wnioski

Zastosowanie regularyzacji oraz metody early stopping przyczynia się do poprawy zdolności modeli do generalizacji. Early stopping pozwala skrócić czas treningu, często jednocześnie zwiększając dokładność predykcji (ale nie zawsze, możemy nie dojść do minimum, jeśli proces treningu będzie za szybko przerwany). Regularyzacja przynosi podobne korzyści, jednak jej skuteczność zależy od odpowiedniego doboru parametrów.