Metody inteligencji obliczeniowej

Sieci neuronowe MLP – sprawozdanie

Jakub Kaproń 327282

kwiecień 2025

1 Bazowa implementacja

Celem pierwszej pracy domowej była podstawowa implementacja sieci neuronowej z algorytmem propagacji w przód oraz próba ręcznego dobrania wag do dwóch zadań regresji, na zbiorze reprezentującym funkcję kwadratową i schodkową. Używaną funkcją aktywacji był sigmoid.

Moja pierwotna implementacja nie była najbardziej optymalna obliczeniowo, gdyż zamiast wykorzystywać operacje na macierzach, używałem pętli, która w każdej iteracji wyznaczała predykcję tylko dla jednej operacji. W późniejszych pracach domowych jednak poprawiłem ten błąd, więc tu przedstawię poprawiony algorytm.

W pętli po $i = 0, \ldots, n-1$ (n to liczba warstw) wyznaczamy macierze wartości aktywacji w $(i+1)$-szej warstwie $A^{(i+1)}$ w następujący sposób (w kolejnych wierszach znajdują się wartości aktywacji dla poszczególnych neuronów $(i+1)$-szej warstwy):

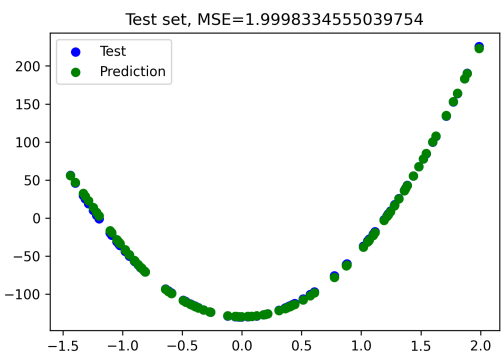
$$A^{(i+1)} = f^{(i+1)}\left(A^{(i)}W^{(i+1)} + b^{(i+1)^T}\right),$$

gdzie:

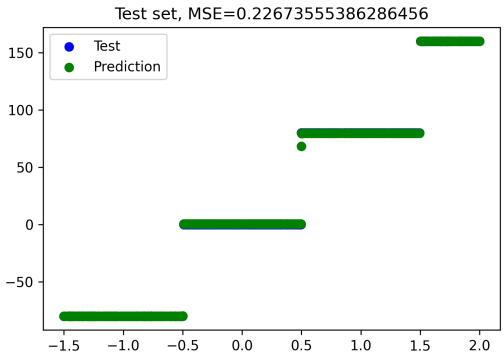
- $A^{(0)} := X$, $A^{(n)} = \hat{Y}$,
- $f^{(i+1)}$ – funkcja aktywacji w $(i+1)$-tej warstwie,
- $W^{(i+1)}$ oznacza macierz wag przejścia z i-tej do $(i+1)$-szej warstwy (liczba wierszy odpowiada liczbie neuronów w i-tej warstwie, a liczba kolumn liczbie neuronów w następnej warstwie,
- $b^{(i+1)}$ – wektor (pionowy) biasów w $(i+1)$-szej warstwie,
- przez $+$ mam na myśli dodanie wektora biasów do każdego wiersza macierzy po lewej stronie znaku.

Do rozwiązania obydwu zadań użyłem sieci z jedną warstwą ukrytą, składającą się z 5 neuronów. Przy rozwiązywaniu zadań kierowałem się następującymi zasadami:

1. wagi połączeń z warstwy wejściowej do warstwy ukrytej odpowiadają za stromość schodka (im większy moduł tym większa stromość) i kierunek (jeśli dodatnia, to schodek w górę, a jeśli ujemna, to w dół),
2. wagi połączeń z warstwy ukrytej do wyjściowej odpowiadają wysokość schodka,
3. biasy w warstwie ukrytej służą do regulacji położenia horyzontalnego, a wyjściowy do wertykalnego.



(a) *square-simple*



(b) *steps-large*

Rysunek 1: Wyniki zadania na zbiorach testowych

2 Propagacja wsteczna błędu

Podobnie jak w poprzednim rozdziale, pierwotna implementacja działała na pętlach, zamaist macierzowo. Ponownie, opisana wersja będzie tą poprawioną.

Zmiany wag obliczane są, przy założeniu MSE jako funkcji straty, w następujący sposób:

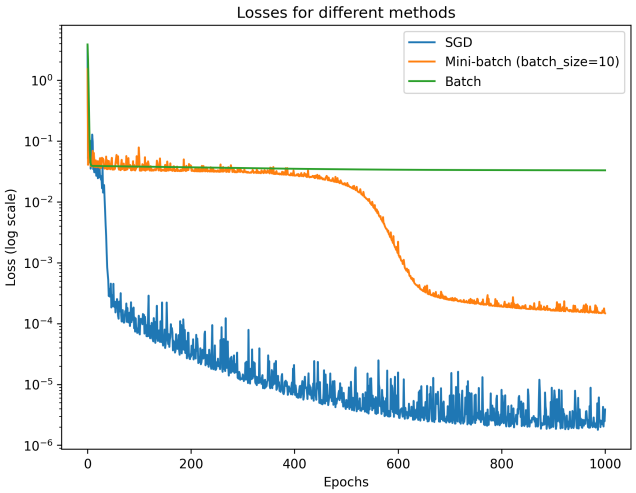
$$\Delta W^{(i)} = -\eta f\left(A^{(i-1)}\right)^T E^{(i)} \cdot \frac{1}{batch\_size},$$
$$\Delta b^{(i)} = -\eta \sum_j {E^{(i)}}^T_{\cdot,j} \cdot \frac{1}{batch\_size},$$

gdzie

- η to współczynnik uczenia,
- $A^{(i)}$ oznacza tu, inaczej niż w poprzednim rozdziale, macierz wartości aktywacji przed nałożeniem funkcji aktywacji,
- $$E^{(i)} = \begin{cases} (\hat{Y} - Y) * f^{(i)'}\left(A^{(i)}\right) & i = n, \\ E^{(i+1)} {W^{(i+1)}}^T * f^{(i)'}\left(A^{(i)}\right) & i < n, \end{cases}$$

  gdzie * oznacza mnożenie element po elemencie.

Uzyskane wyniki były zgodne z logiką i informacjami przekazanymi na wykładzie. Im mniejszy batch, tym funkcja straty zbiegała szybciej do swojego minimum w sensie liczby epok, ale cechowała się większymi lokalnymi fluktuacjami (była mniej stabilna) i wiązała się z większym kosztem obliczeniowym, a więc dłuższym ogólnym czasem trenowania. Poniżej załączam porównanie szybkości zbieżności poszczególnych metod dla zbioru *square-simple* (Rysunek 2), gdzie czas trenowania sieci przez 1000 epok dla SGD, Mini-batch (rozmiar batcha 10) i Batch wynosił kolejno 7.5 sek., 1.2 sek., 0.5 sek.

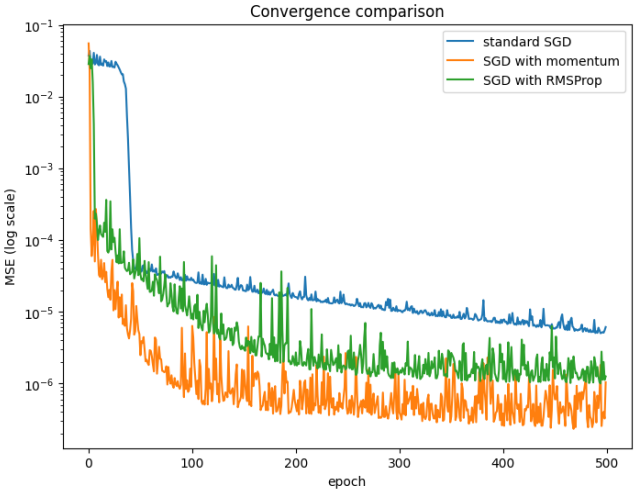


Rysunek 2: Porównanie szybkości zbieżności poszczególnych metod dla zbioru *square-simple*.

3 Moment i normalizacja gradientu

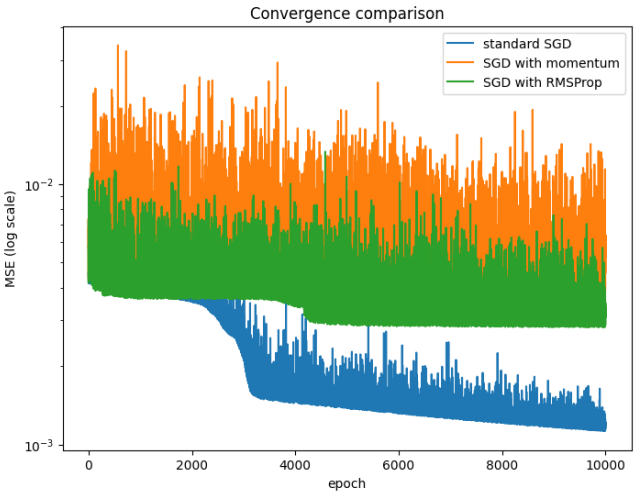
W tym zadaniu porównywałem zbieżność sieci uczącej się przy pomocy standardowej wstecznej propagacji, z momentem oraz z normalizacją gradientu RMSProp, metodą SGD. Na każdym ze zbiorów uzyskałem nieco różniące się wyniki.

Na *square-large* najlepsze okazało się uczenie z momentem, a najgorsze standardowe SGD. Zmodyfikowane metody zbiegały szybciej, a uzyskane końcowe rezultaty były lepsze.



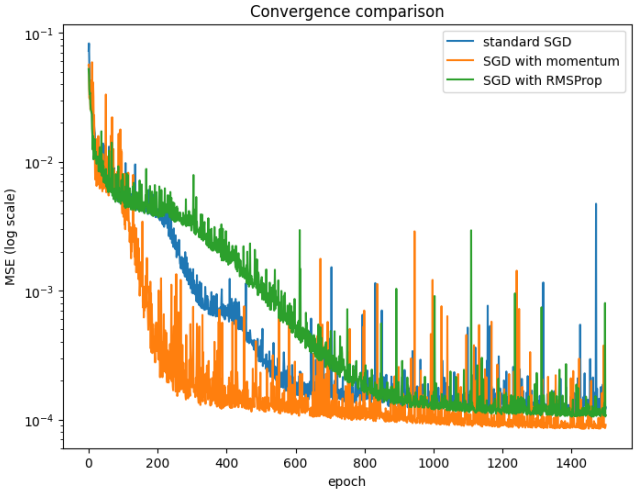
Rysunek 3: Porównanie szybkości zbieżności poszczególnych metod dla zbioru *square-large*.

W przypadku zbioru *steps-large*, uczenie z obydwoma modyfikacjami okazało się wolniejsze i mniej stabilne.



Rysunek 4: Porównanie szybkości zbieżności poszczególnych metod dla zbioru *steps-large*.

Natomiat na *multimodal-large* najszybciej zbiegało uczenie z momentem, a najwolniej z RMSProp, chociaż końcowy rezultat w przypadku RMSProp okazał się nieco lepszy od standardowego SGD.



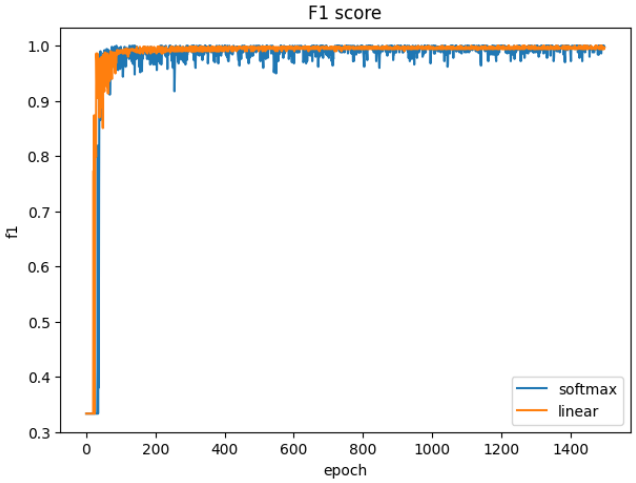
Rysunek 5: Porównanie szybkości zbieżności poszczególnych metod dla zbioru *multimodal-large*.

Uznałem moment za bardziej odpowiednią metodę, biorąc pod uwagę nie tylko to, że przeprowadzone eksperymenty pokazały, że to najczęściej ta metoda daje najszybszą zbieżność, ale też fakt, że ma o jeden parametr mniej do dobrania niż RMSProp.

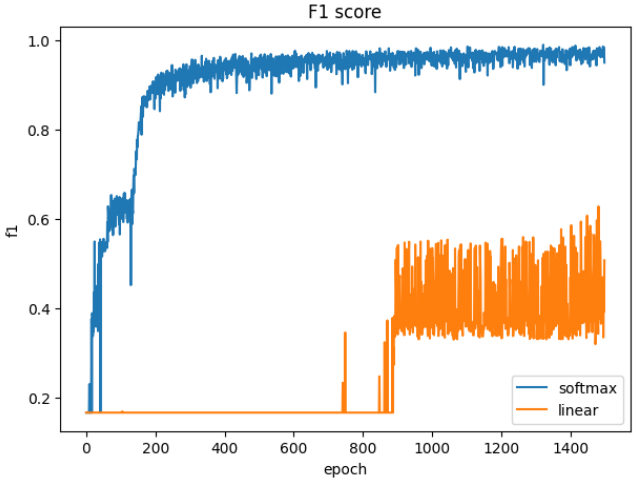
4 Zadanie klasyfikacji

Celem tej pracy domowej było rozwiązanie zadania klasyfikacji przy pomocy funkcji aktywacji (na warstwie wyjściowej) softmax oraz liniowej i porównania ich skuteczności. W przypadku softmax używałem cross entropy jako funkcji straty, co wymagało odpowiedniego zmodyfikowania algorytmu wstecznej propagacji ($E^{(n)} = \hat{Y} - Y$), natomiast przy liniowej funkcji używałem MSE. Wyniki porównywałem na podstawie F1-score.

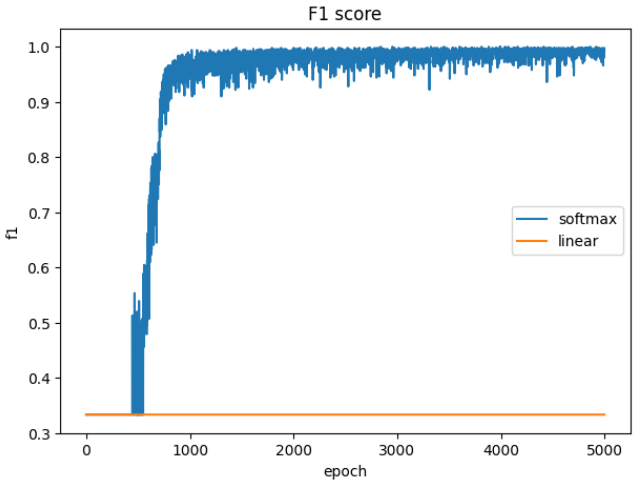
We wszystkich przypadkach sieć z softmaxem spisywała się lepiej. Sieci z liniową funkcją liniową w zależności od skomplikowania zbioru dawały bardzo różne rezultaty: od niewiele gorszych od sieci z softmax, przez znacząco gorsze po przypadek gdzie w ogóle nie można było zaobserwować zbieżności.



Rysunek 6: Wyniki na zbiorze *easy* – porównywalna wydajność.



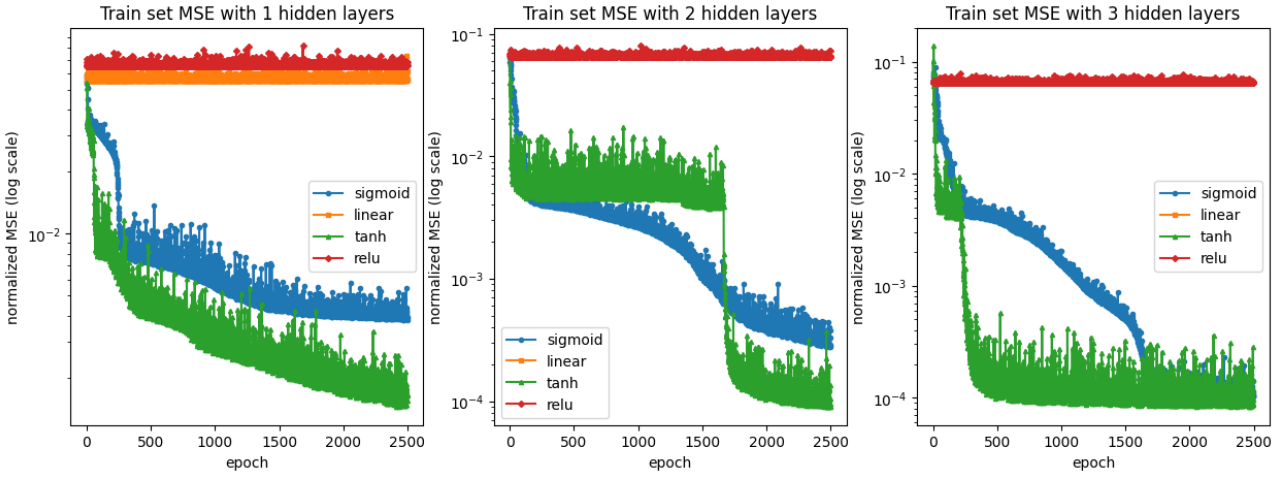
Rysunek 7: Wyniki na zbiorze *rings3-regular* – softmax zauważalnie bardziej wydajny.



Rysunek 8: Wyniki na zbiorze *xor3* – funkcja liniowa nie zbiega.

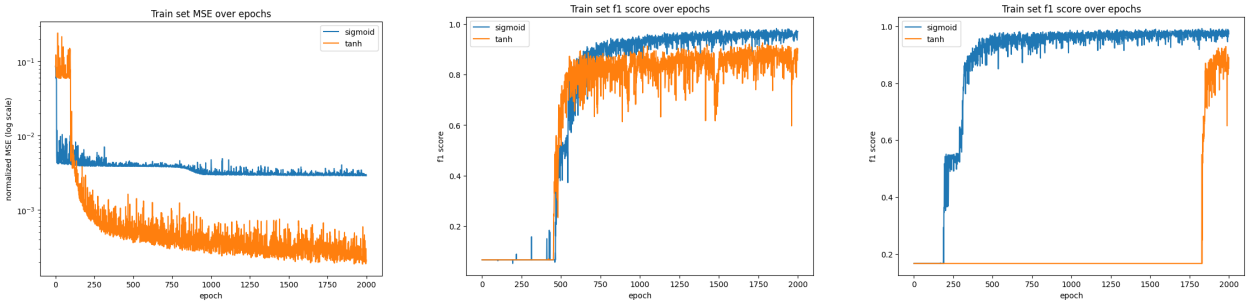
5 Testowanie różnych funkcji aktywacji

W tym zadaniu należało zbadać efektywność sieci z różnymi funkcjami aktywacji (sigmoid, liniowa, tanh, ReLU) zarówno w regresji jak i klasyfikacji. Wstępnej selekcji liczby warstw ukrytych i funkcji aktywacji dokonałem na zbiorze *multimodal-large*. Na podstawie wyników wstępnych testów, do dalszego badania wybrałem sieci z sigmoidem i tanh, obydwie z trzema warstwami ukrytmi liczącymi po 10 neuronów.



Rysunek 9: Wyniki wstępnych testów na zbiorze *multimodal-large*.

Następnie, gdy przetestowałem te sieci na pozostałych zbiorach można było zauważyć zależność, że w zadaniach regresji lepiej radziły sobie sieci z tanh, natomiast w klasyfikacji – sigmoid.



(a) *steps-large* (b) *rings5-regular* (c) *rings3-regular*

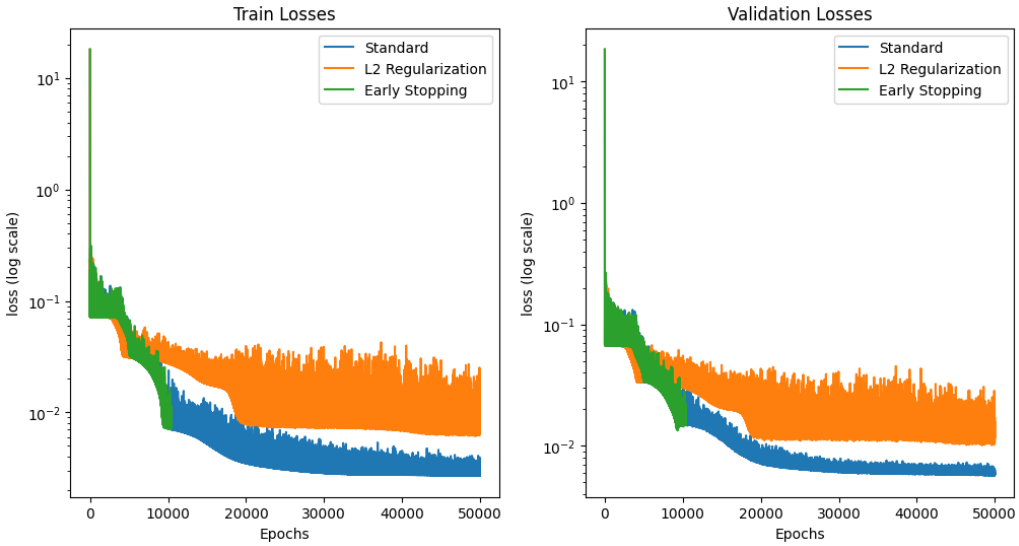
Rysunek 10: Porównanie efektywności wybranych architektur.

6 Zjawisko przeuczenia i regularyzacja

Przebadałem skuteczność przeciwdziałania pojawieniu się zjawiska przeuczenia przy pomocy regularyzacji L2 oraz mechanizmowi wczesnego zatrzymywania. Ten pierwszy zaimplementowałem we wstecznej propagacji jako dodanie do zmiany każdej z wag jej wartości podzielonej przez liczbę wszystkich wag i pomnożonej przez współczynnik będący parametrem. Innymi słowy zmodyfikowałem gradient funkcji straty tak, aby brał pod uwagę wysokość wag. Jednakże przy ewaluacji modelów wyliczana była wartość zwykłej funkcji straty.

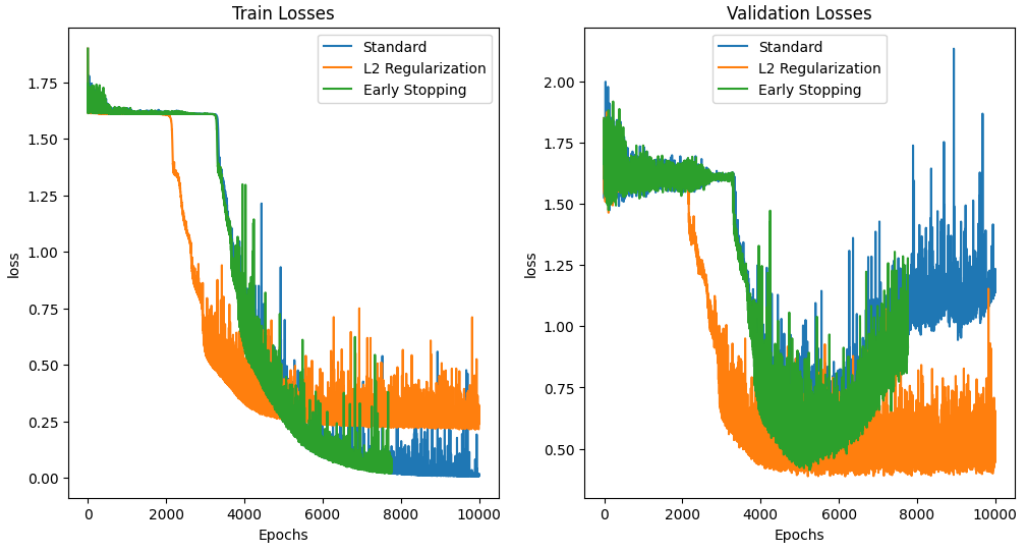
Wczesne zatrzymywanie natomiast polegało natomiast na monitorowaniu funkcji straty na zbiorze walidacyjnym (w tym przypadku testowym) i zatrzymaniu uczenia oraz wybraniu wag dających minimum funkcji straty, po nie uzyskaniu poprawy po pewnej liczbie epok (parametr cierpliwości). Dodałem także dodatkowy parametr kontrolujący o jaki współczynnik musi być większa nowa wartość funkcji kosztu od minimum, aby ten przypadek był zaliczony jako brak poprawy, co miało w na celu zapobiegnięcie zatrzymywania się uczenia przy długo trwających fluktuacjach.

W przypadku zbioru *multimodal-sparse* nie zaobserwowałem zjawiska przeuczenia – zarówno na treningowym, jak i testowym zbiorze, funkcja straty cały czas spadała, pomimo aż 50 tyś. epok.



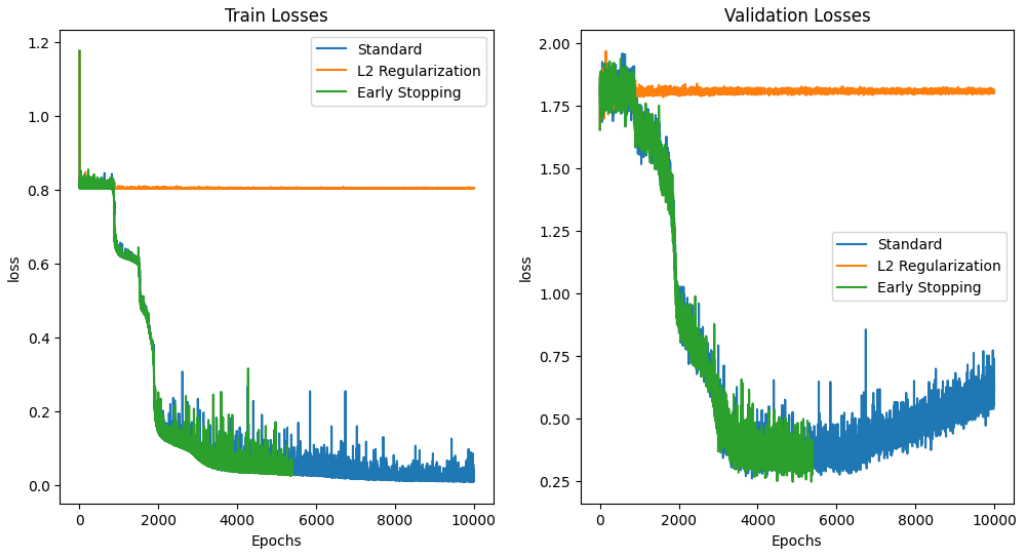
Rysunek 11: Funkcja straty na przestrzeni uczenia sieci na zbiorze *multimodal-sparse*.

Na zbiorze *rings5-sparse* zostało ono już jednak zaobserwowane, a zaimplementowane metody zadziałały jak powinny – obie dały lepszą końcową wartość funkcji straty na zbiorze testowym.

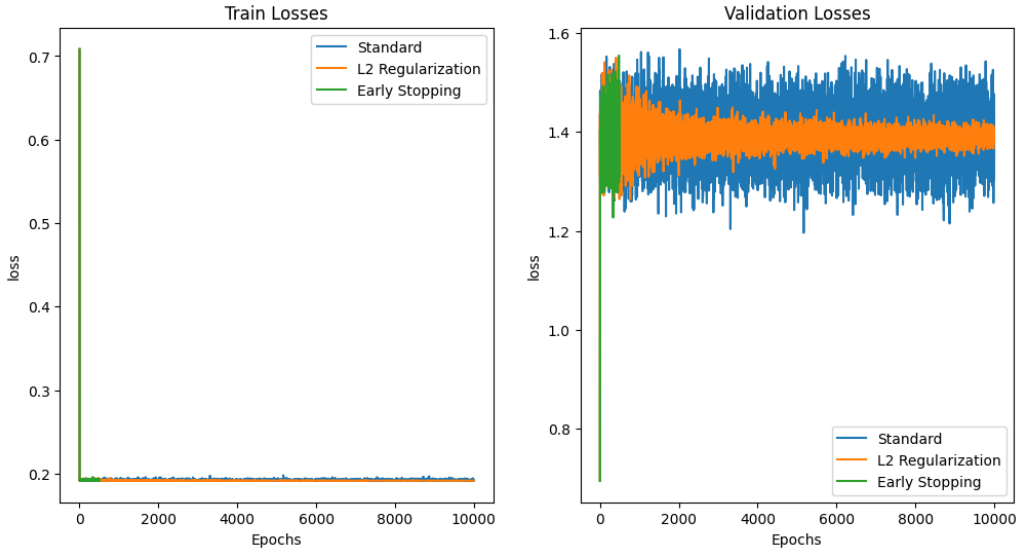


Rysunek 12: Funkcja straty na przestrzeni uczenia sieci na zbiorze *rings5-sparse*.

Na dwóch pozostałych zbiorach wystąpiły pewne problemy. Na *rings3-balance* sieć uczona z regularyzacją L2 nie chciała się nauczyć – przyporządkowywała wszystkie punkty do jednej klasy, ale reszta metod działała normalnie. Natomiast na *xor3-balance* żaden z modeli nie chciał się w ogóle nauczyć.



Rysunek 13: Funkcja straty na przestrzeni uczenia sieci na zbiorze *rings3-balance*.



Rysunek 14: Funkcja straty na przestrzeni uczenia sieci na zbiorze *xor3-balance*.