Akademia Górniczo-Hutnicza
Wydział Informatyki
Kierunek Informatyka

Metody Obliczeniowe w Nauce i Technice

Laboratorium 10

Równania różniczkowe - spectral bias

Wojciech Michaluk, Kyrylo Iakymenko

Kraków, 29 maja 2024

1 Wprowadzenie

W ramach dzisiejszego laboratorium będziemy kontynuować temat równań różniczkowych. Zajmiemy się tym razem estymacją rozwiązań równania różniczkowego zwyczajnego za pomocą sieci neuronowych *PINN* (Physics-informed Neural Network), korzystając z ich implementacji przy użyciu biblioteki `pytorch`.

PINN to sieci neuronowe, które pozwalają modelować równania wynikające z praw fizyki, po dostarczeniu im danych w procesie uczenia, oraz mogą być opisane za pomocą równań różniczkowych *cząstkowych* (ang. **PDE**) [1].

1.1 Szczegóły specyfikacji sieci neuronowych

Poniżej przedstawiamy charakterystykę używanych w ramach tego laboratorium sieci neuronowych. Koszt rezydualny zdefiniowany jest następująco:

$$\mathcal{L}_r(\theta) = \frac{1}{N}\sum_{i=1}^{N} ||\frac{d\hat{u}(x)}{dx} - \frac{du(x)}{dx}||^2,$$

gdzie N jest liczbą punktów kolokacyjnych, $\hat{u}(x)$ to rozwiązanie uzyskane z sieci neuronowej, natomiast $u(x)$ to funkcja, która jest niewiadomą w równaniu różniczkowym.

Koszt związany z warunkiem początkowym przyjmuje postać:

$$\mathcal{L}_{IC}(\theta) = ||\hat{u}(0) - u(0)||^2,$$

a funkcja kosztu odpowiednio

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_{IC}(\theta).$$

Nie rozważamy tutaj warunków brzegowych, dlatego nie pojawia się ta składowa we wzorze na funkcję kosztu.

Warstwa wejściowa sieci posiada 1 neuron, reprezentujący zmienną x, warstwa wyjściowa także posiada 1 neuron, reprezentujący zmienną $\hat{u}(x)$.
Uczenie trwa przez 50 000 kroków, używamy algorytmu *Adam* ze stałą uczenia równą 0.001. Jako funkcję aktywacji przyjmujemy *tanh* (tangens hiperboliczny).

2 Opis zadania

Celem zadania jest estymacja rozwiązań podanego równania różniczkowego:

$$\frac{du(x)}{dx} = cos(\omega x)$$

dla dziedziny $-2\pi \leq x \leq 2\pi$ i warunku początkowego $u(0) = 0$.
Występujące w tym równaniu wielkości oznaczają odpowiednio:

- $x, \omega \in \mathbb{R}$, przy czym x to położenie,
- $u(\cdot)$ to funkcja, której postaci szukamy.

Mamy podany warunek początkowy $u(0) = 0$. Najpierw należy dla różnych wartości parametru ω, oraz wartości takich czynników jak:

- liczba warstw ukrytych,
- liczba neuronów w każdej warstwie,
- liczba punktów treningowych oraz
- liczba punktów testowych,

[1] Za angielską wikipedią: `https://en.wikipedia.org/wiki/Physics-informed_neural_networks`

przeprowadzić eksperymenty i przeanalizować uzyskane wyniki.

W dalszej części zadania będziemy porównywać wynik z wybranej sieci z poprzedniego punktu z rozwiązaniami uzyskanymi w inny sposób, np. kiedy pierwszą warstwę ukrytą zainicjalizowano cechami Fouriera.

Dla każdego uzyskanego rezultatu przedstawimy wykresy porównawcze tego rozwiązania wraz z dokładnym rozwiązaniem funkcji $u(x)$, wykresy funkcji błędu oraz wykresy funkcji kosztu w zależności od liczby epok.

3 Opracowanie zadania

Analityczna postać rozwiązania badanego równania z warunkiem początkowym jest następująca:

$$u(x) = \frac{1}{\omega}\sin(\omega x).$$

Korzystając z danych w poleceniu i uwzględniając naszą specyfikację sieci neuronowych, koszt rezydualny jest opisany zależnością:

$$\mathcal{L}_r(\theta) = \frac{1}{N}\sum_{i=1}^{N} ||\frac{d\hat{u}(x)}{dx} - cos(\omega x_i)||^2,$$

natomiast koszt związany z warunkiem początkowym to:

$$\mathcal{L}_{IC}(\theta) = ||\hat{u}(0) - 0||^2 = ||\hat{u}(0)||^2.$$

Aby wykonać to zadanie, korzystamy z lekko zmodyfikowanego kodu z pliku `PINN.py`.

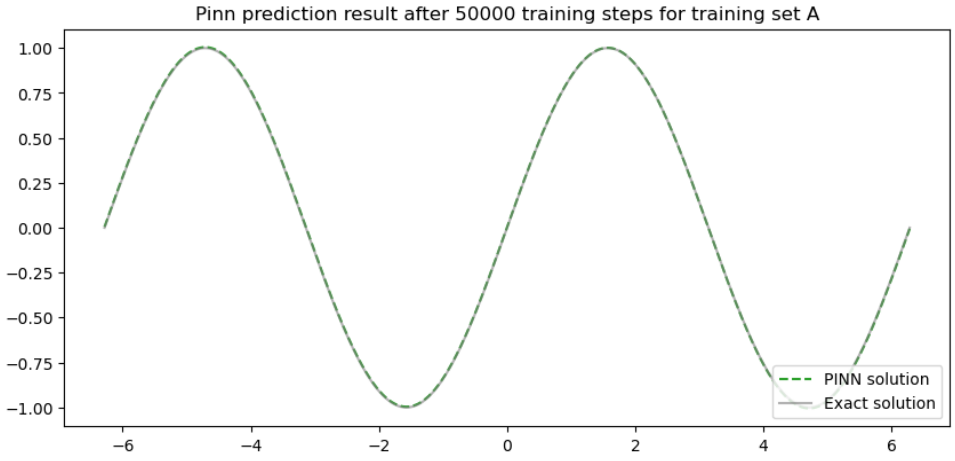
3.1 Eksperymenty z sieciami neuronowymi dla różnych parametrów

W pierwszej części zadania badamy wyniki uzyskane z wykorzystaniem sieci neuronowych dla wartości parametrów tak jak podano poniżej. Dla każdego z tych przypadków przedstawiamy wspomniane w opisie zadania wykresy. Wykresy są przedstawiane dla punktów ze zbioru testowego.
Ponadto, przedstawiamy dokładne wartości funkcji kosztu co 10000 kroków.

3.1.1 Przypadek a

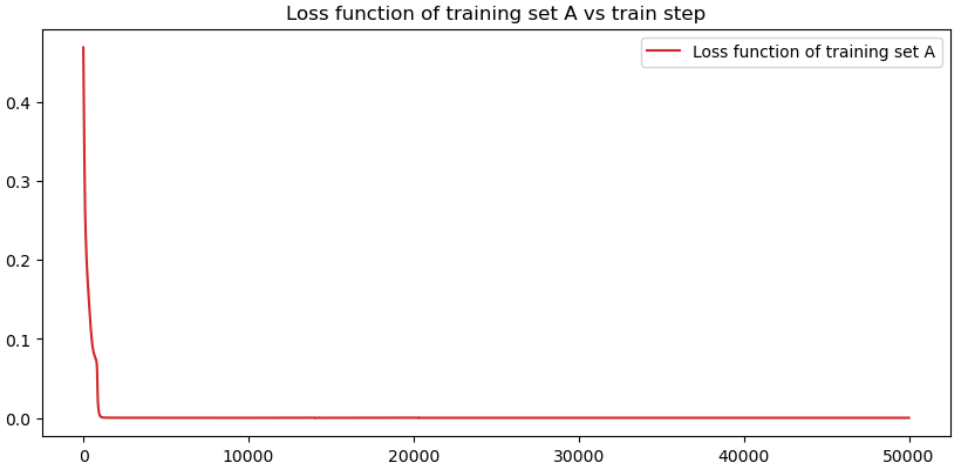
Przyjmujemy $\omega = 1$, 2 warstwy ukryte, 16 neuronów w każdej warstwie, 200 punktów treningowych oraz 1000 punktów testowych.

Najpierw wspólny wykres funkcji $u(x)$ oraz funkcji $\hat{u}(x)$.



Rysunek 1: Wykres rozwiązania uzyskanego przez sieć neuronową w porównaniu do rzeczywistego rozwiązania

Teraz przedstawmy wartości funkcji kosztu dla kolejnych epok podczas trenowania.



Rysunek 2: Wykres funkcji kosztu w zależności od kroku treningowego

Spójrzmy na dokładne wartości funkcji kosztu w wybranych krokach.

Numer epoki	0	10000	20000	30000	40000	50000
Wartość funkcji kosztu	0.469568	$5.598{\cdot}10^{-6}$	$1.338{\cdot}10^{-6}$	$1.477{\cdot}10^{-6}$	$2.059{\cdot}10^{-6}$	$1.090{\cdot}10^{-5}$

Tabela 1: Wartości funkcji kosztu co 10000 kroków

Jak widać, w tym przypadku sieć neuronowa bardzo szybko znajduje optymalne rozwiązanie, dla którego wartość funkcji kosztu utrzymuje się na poziomie rzędu 10^{-6}. Wraz z dalszym trenowaniem sieci neuronowej, wartość funkcji kosztu trochę fluktuuje i na koniec jest nieco większa niż dla poprzednich wybranych kroków.

Poniżej również przedstawiamy wykres funkcji błędu (różnicy bezwzględnej otrzymanej aproksymacji i dokładnego rozwiązania równania).

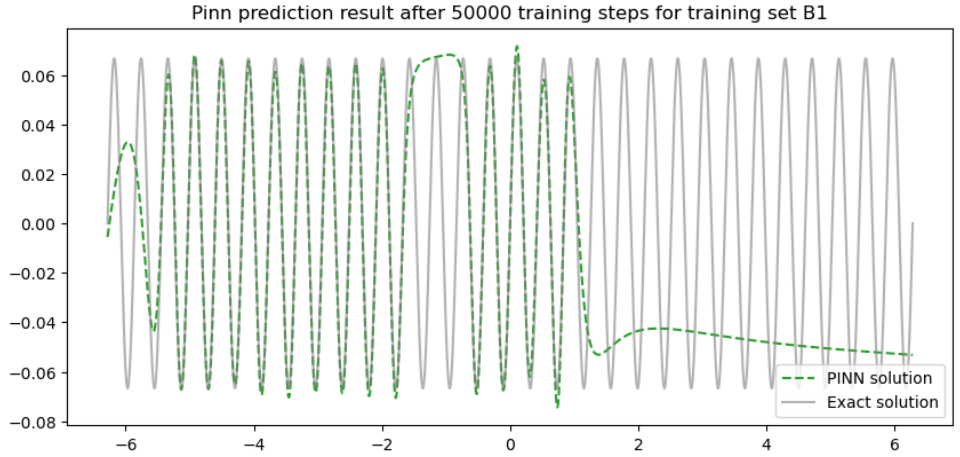
Rysunek 3: Funkcja błędu dla przypadku a

Jak zobaczymy w kolejnych podpunktach, aproksymacje naszego równania otrzymane za pomocą sieci często mają ten problem, że im dalej od warunku początkowego (w naszym przypadku $x = 0$), tym bardziej tracą precyzję. Także i w tym przypadku funkcja błędu potwierdza to spostrzeżenie swoim charakterystycznym wyglądem (coś przypominającego literę "V").

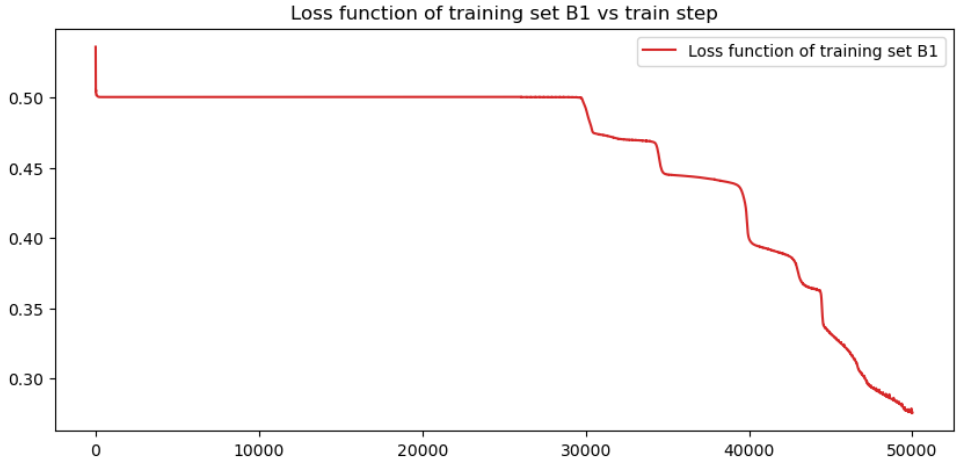
3.1.2 Przypadek $b1$

Przyjmujemy $\omega = 15$, 2 warstwy ukryte, 16 neuronów w każdej warstwie, 3000 punktów treningowych oraz 5000 punktów testowych.

W tym przypadku, podobnie jak i w następnych, przedstawiamy wykresy w tej samej kolejności.



Rysunek 4: Wykres rozwiązania uzyskanego przez sieć neuronową w porównaniu do rzeczywistego rozwiązania



Rysunek 5: Wykres funkcji kosztu w zależności od kroku treningowego

Już patrząc na wykresy widzimy, że 50000 kroków treningowych nie wystarczyło, aby sieć dawała dokładną predykcję. Potwierdzamy nasze wnioski wartościami w tabeli.

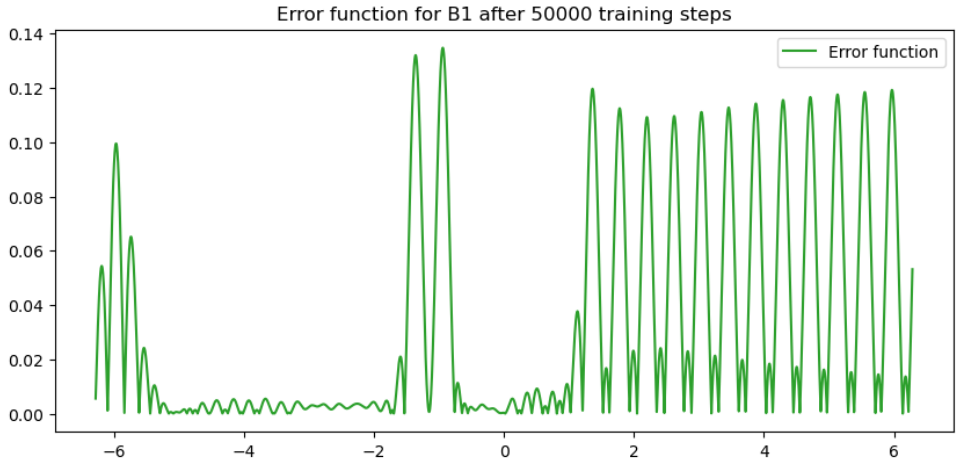
Numer epoki	0	10000	20000	30000	40000	50000
Wartość funkcji kosztu	0.535859	0.500145	0.500119	0.492264	0.399439	0.275739

Tabela 2: Wartości funkcji kosztu co 10000 kroków

Wartości funkcji na pierwszym wykresie pokrywają się jedynie mniej więcej na przedziałach $[-5;-2]$ i $[-1;1]$, natomiast na pozostałych częściach dziedziny funkcji rozwiązanie jest bardzo niedokładne.

Wartość funkcji kosztu praktycznie się nie zmienia przez początkowe 30000 kroków, dopiero później stopniowo maleje, a po kroku nr 40000 tempo spadku nieco wzrasta. Być może dla np. 70000 kroków rozwiązanie byłoby już dokładne? Jednakże, to rozważania na odrębny temat, gdyż w naszym modelu przyjmujemy 50000 kroków treningowych.

Poniżej przedstawiamy wykres funkcji błędu.



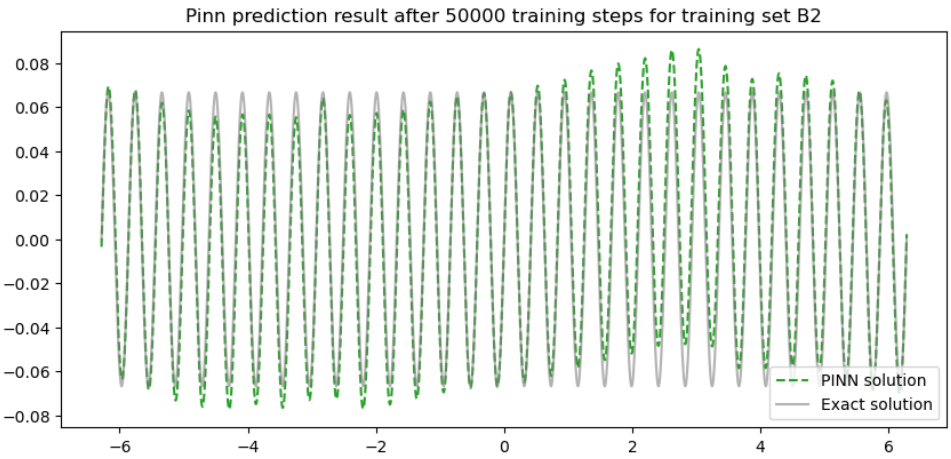
Rysunek 6: Funkcja błędu dla przypadku $b1$

Funkcja błędu tym razem znacząco się różni w porównaniu do przypadku a. Spostrzegamy oscylujące zachowanie błędu, przypominające funkcje trygonometryczne.

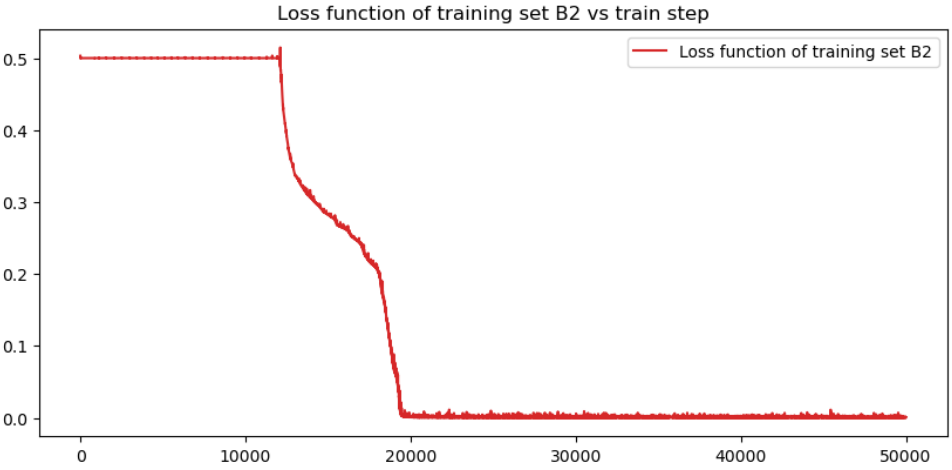
3.1.3 Przypadek $b2$

Przyjmujemy $\omega = 15$, 4 warstwy ukryte, 64 neurony w każdej warstwie, 3000 punktów treningowych oraz 5000 punktów testowych.

Najpierw przedstawiamy wykresy.



Rysunek 7: Wykres rozwiązania uzyskanego przez sieć neuronową w porównaniu do rzeczywistego rozwiązania



Rysunek 8: Wykres funkcji kosztu w zależności od kroku treningowego

Tym razem patrząc na wykresy stwierdzamy, że sieć dobrze poradziła sobie z zadaniem znalezienia funkcji, bowiem wykresy w górnej części praktycznie sie pokrywają, a wykres funkcji kosztu w dolnej części sugeruje małe wartości tej funkcji.

Numer epoki	0	10000	20000	30000	40000	50000
Wartość funkcji kosztu	0.503338	0.500161	0.001639	0.000292	0.000715	0.000806

Tabela 3: Wartości funkcji kosztu co 10000 kroków

Tym razem wartość funkcji kosztu zaczyna maleć po mniej więcej kroku nr 10000 i dzieje się to stopniowo do kroku 20000, później przyjmuje ona wartości rzędu 10^{-3}, co wydaje się być akceptowalną wartością.

Zwiększenie liczby warstw ukrytych i liczby neuronów w każdej warstwie zapewne znacząco wpłyneło na uzyskanie dokładniejszego rozwiązania, natomiast czas obliczeń zauważalnie się wydłuża.

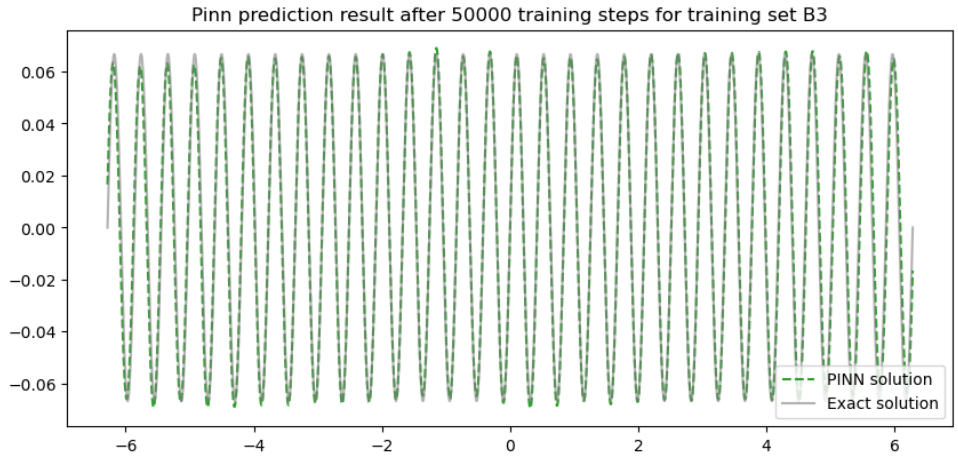
Poniżej przedstawiamy wykres funkcji błędu.

Rysunek 9: Funkcja błędu dla przypadku $b2$

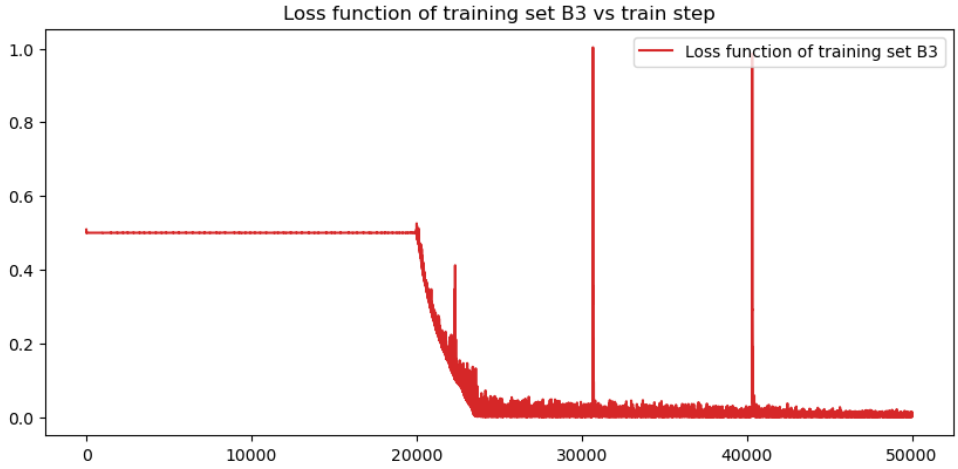
3.1.4 Przypadek $b3$

Przyjmujemy $\omega = 15$, 5 warstw ukrytych, 128 neuronów w każdej warstwie, 3000 punktów treningowych oraz 5000 punktów testowych.

Najpierw przedstawiamy wykresy.



Rysunek 10: Wykres rozwiązania uzyskanego przez sieć neuronową w porównaniu do rzeczywistego rozwiązania



Rysunek 11: Wykres funkcji kosztu w zależności od kroku treningowego

Teraz również sieć dobrze poradziła sobie z zadaniem znalezienia funkcji, lecz zauważamy ciekawą anomalię w okolicach kroków 30000 oraz 40000, kiedy to wartość funkcji kosztu została wyznaczona jako ≈ 1, czyli około dwukrotnie więcej niż na samym początku!

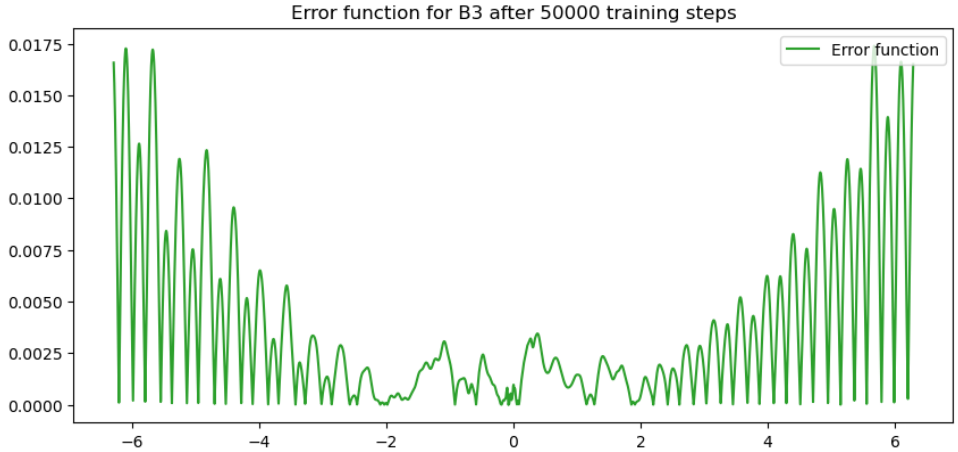
Numer epoki	0	10000	20000	30000	40000	50000
Wartość funkcji kosztu	0.508880	0.500166	0.498143	0.010984	0.004050	0.006046

Tabela 4: Wartości funkcji kosztu co 10000 kroków

Jak można zauważyć, bazując na drugim wykresie oraz wartościach w tabeli, wartości funkcji kosztu maleją po kroku nr 20000. Zachowana jest tendencja malejąca poza wspomnianą anomalią oraz pomniejszą po kroku nr 25000, kiedy również wartość funkcji kosztu gwałtownie wzrosła.

Paradoksalnie, mimo dalszego zwiększania liczby warstw ukrytych i liczby neuronów w każdej warstwie, uzyskane wyniki są gorszej jakości niż w poprzednim przypadku, natomiast czas obliczeń jest jeszcze dłuższy.

W potwierdzeniu powyższej analizy przedstawiamy wykres funkcji błędu.



Rysunek 12: Funkcja błędu dla przypadku $b3$

Po przeanalizowaniu czterech przypadków dla podstawowego wariantu sieci neuronowej, wyciągamy wnioski, że przyjęcie wartości parametru $\omega = 15$ znacząco utrudniło zadanie znalezienia funkcji.

Gdy wartości badanych parametrów sieci neuronowej (liczba warstw ukrytych i liczba neuronów w każdej warstwie) pozostały takie same jak dla przypadku $\omega = 1$, to sieć neuronowa znalazła bardzo niedokładne rozwiązanie, nieakceptowalne w warunkach praktycznych. Dopiero ich zwiększenie doprowadziło do sensownego rozwiązania, które i tak

nie było aż tak dokładne jak to dla $\omega = 1$.

Należy jednak wziąć pod uwagę fakt, że dla $\omega = 15$ funkcja ma znacznie bardziej zmienny charakter, więc jej znalezienie w dokładnym stopniu jest o wiele trudniejszym rozwiązaniem.

3.2 Porównanie wybranego wyniku z rozwiązaniami uzyskanymi z innych podejść

3.2.1 Przypadek $\hat{u}(x, \theta) = \tanh(\omega x) \cdot NN(x, \theta)$

Do porównania jako reprezentanta sieci neuronowych w wariancie podstawowym wybieramy pierwszą sieć z poprzedniego podpunktu - ze względu na prostotę i wysoką jakość rozwiązania. Porównujemy rozwiązanie uzyskane z pomocą tej sieci z takim, w którym przyjęto, że szukane rozwiązanie ma postać:

$$\hat{u}(x, \theta) = \tanh(\omega x) \cdot NN(x, \theta),$$

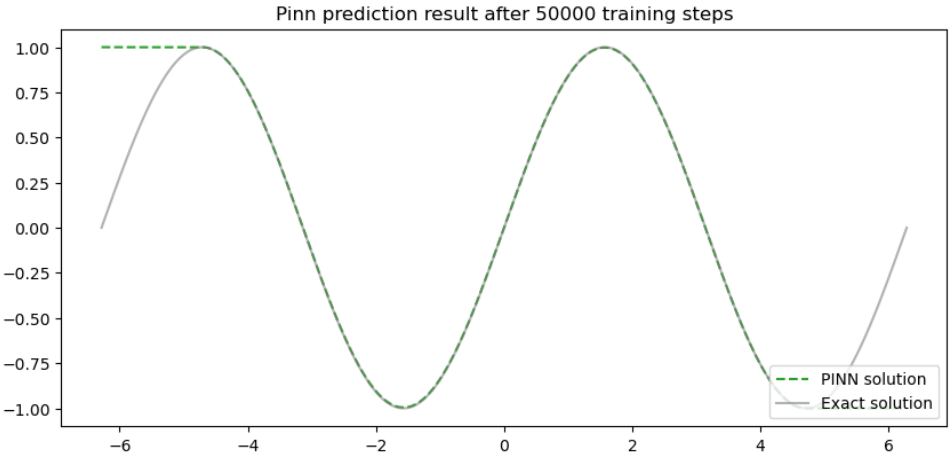
gdzie NN oznacza pierwotne rozwiązanie z sieci neuronowej. Zauważmy, że taka postać rozwiązania gwarantuje spełnienie warunku $\hat{u}(0) = 0$, nie wprowadzając składnika $\mathcal{L}_{IC}$ do funkcji kosztu.

Przedstawiamy najpierw wartości funkcji kosztu w wybranych krokach, tak jak to zrobiliśmy w poprzednim punkcie.

Numer epoki	0	10000	20000	30000	40000	50000
Wartość funkcji kosztu	0.498186	0.129484	0.129396	0.129387	0.129383	0.129382

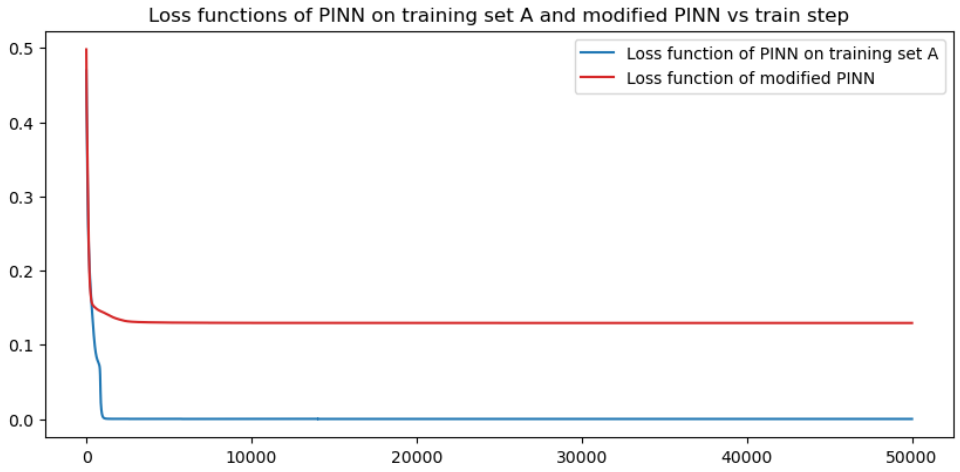
Tabela 5: Wartości funkcji kosztu co 10000 kroków

Zobaczmy jeszcze wykresy uzyskanego rozwiązania i funkcji kosztu.



Rysunek 13: Wykres porównawczy rozwiązania uzyskanego przez sieć z rzeczywistym rozwiązaniem

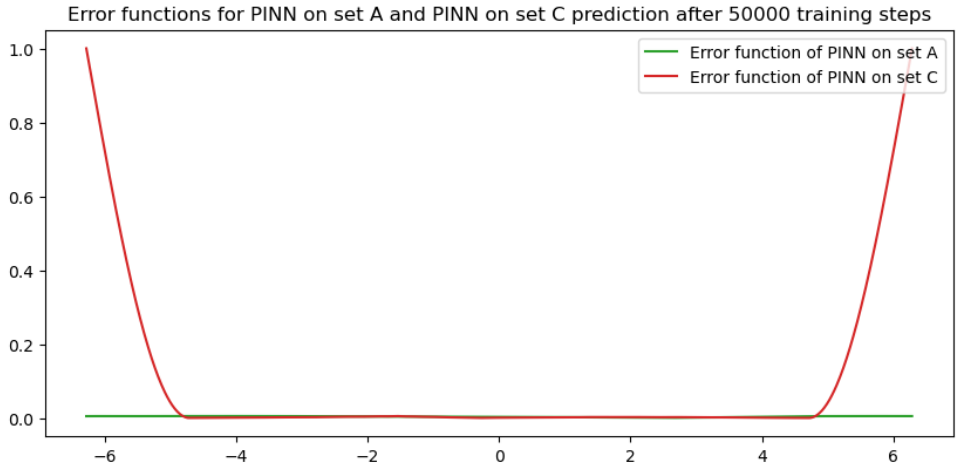
Wykres funkcji kosztu tym razem porównamy z przypadkiem a i przeanalizujemy różnice.



Rysunek 14: Porównanie funkcji kosztu z podpunktu a i c w zależności od epoki

Można zauważyć, że funkcja kosztu bardzo szybko osiąga pewną wartość, wynoszącą ≈ 0.13, natomiast później utrzymuje się na takim poziomie - bardzo wolno maleje. Jeżeli spojrzymy na wykres uzyskanego rozwiązania, to widać, że na krańcach przedziałów, na pewnych odcinkach obliczona funkcja przyjmuje wartość stale równą 1, co jest zjawiskiem zdecydowanie niepożądanym. Ciężko powiedzieć czemu tak sie dzieje, ponieważ nawet własności funkcji *tanh* nie sugerowałyby takiego zachowania.

Bez wątpienia właśnie to zjawisko wpływa na wartość funkcji kosztu. Porównując funkcje kosztu sieci z podpunktu a ze zmodyfikowanym wariantem, widzimy, że pierwotnie nie mieliśmy takiego problemu z funkcją kosztu utrzymującą się w pewnej odległości od $y = 0$. Jest szansa, że dzieje się to dlatego, że zakładamy (odpowiednio modyfikując sieć), że nasze rozwiązanie będzie miało postać $\hat{u}(x, \theta) = \tanh(\omega x) \cdot NN(x, \theta)$, co jest niezbyt korzystne, gdy rzeczywistym rozwiązaniem jest sinusoida. Chociaż warto zauważyć, że ta modyfikacja ma inne ciekawe zalety związane z funkcją błędu, do której przechodzimy.



Rysunek 15: Porównanie funkcji błędu z podpunktu a i c

Dopiero w tym momencie jesteśmy w stanie zauważyć niektóre potencjalne zalety badanej przez nas metody. Założenie, że funkcja jest iloczynem tangensa hiperbolicznego i pewnego rozwiązania sieci daje nam aproksymację, której błąd jest funkcją dużo bardziej przewidywalną. Stosując tę metodę jesteśmy w stanie otrzymać aproksymacje z błędem bardziej jednorodnym na interesującym nas wykresie. Mamy przez to gwarancję, że otrzymana funkcja będzie zachowywała się podobnie do rzeczywistego rozwiązania, zachowując (przynajmniej w większym stopniu w porównaniu do zwykłej sieci z podpuntu a) cechy przebiegu szukanego rozwiązania.

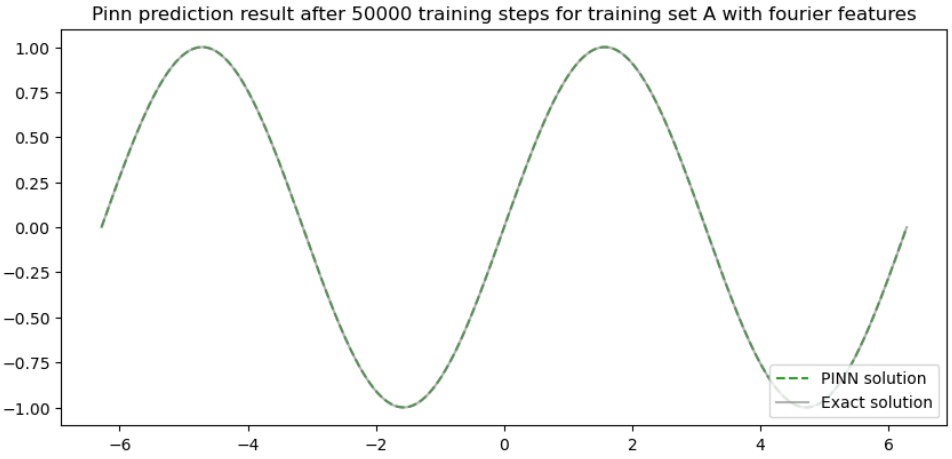
3.2.2 Cechy Fouriera

Na koniec dokonujemy porównania pierwotnej sieci neuronowej z rozwiązaniem, w którym pierwszą warstwę ukrytą zainicjalizowano cechami Fouriera, tzn. jest ona przedstawiona następująco:

$$\gamma(x) = [\sin(2^0\pi x), \cos(2^0\pi x), ..., \sin(2^{L-1}\pi x), \cos(2^{L-1}\pi x)],$$

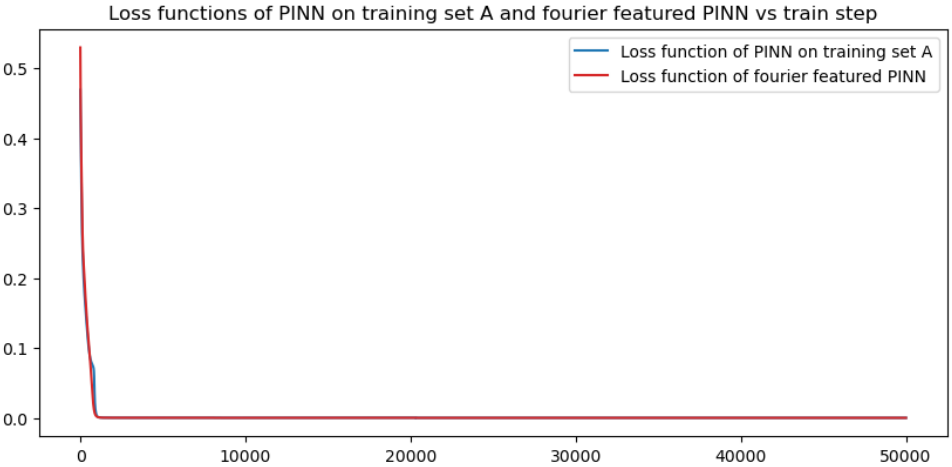
gdzie L należy dobrać tak, aby nie zmieniać szerokości warstwy ukrytej. W naszym przypadku szerokość warstwy ukrytej to 16, więc jako że każde L wystepuje dwa razy (w funkcji *sin* i *cos*), to dobieramy $L = 8$.

Poniżej przedstawiamy wykres uzyskanej aproksymacji po 50 000 krokach trenowania.

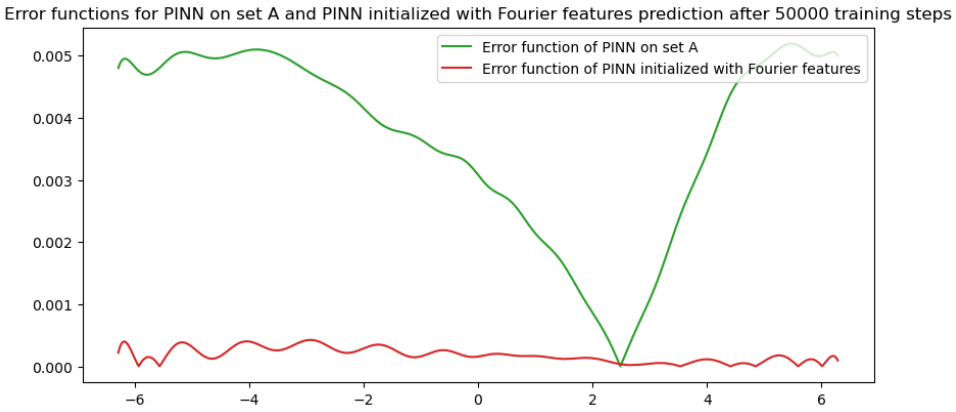


Rysunek 16: Wykres porównawczy rozwiązania uzyskanego przez sieć z rzeczywistym rozwiązaniem

Dwa kolejne wykresy przedstawiają porównanie odpowiednich wartości z przypadkiem a.



Rysunek 17: Porównanie funkcji kosztu z podpunktu a i d w zależności od epoki



Rysunek 18: Porównanie funkcji błędu z podpunktu a i d w zależności od epoki

Widzimy, że na oko aproksymacja otrzymana z użyciem cech Fouriera niczym się nie różni od zwykłej sieci z przypadku a. Porównanie funkcji kosztu również wskazuje na to, że te dwie metody zachowują się dosyć podobnie. Natomiast porównanie wykresów funkcji błędu pozwala zauważyć kluczową różnicę w przypadku zastosowania cech Fouriera - funkcja w podpunkcie d ma dużo bardziej równomierny błąd.

Wyjaśnieniem tego zjawiska mogłoby być to, że MLP korzystające z cech Fouriera są szczególnie efektywne w celach reprezentacji funkcji o wysokiej częstotliwości z dziedziną małowymiarową [2] (w naszym przypadku wymiar dziedziny to 1, więc istotnie mały).

Poniżej przedstawiamy jeszcze tabelę wartości funkcji kosztu.

Numer epoki	0	10000	20000	30000	40000	50000
Wartość funkcji kosztu	0.530086	$2.111 \cdot 10^{-5}$	$3.457 \cdot 10^{-6}$	$1.280 \cdot 10^{-6}$	$6.067 \cdot 10^{-7}$	$3.603 \cdot 10^{-7}$

Tabela 6: Wartości funkcji kosztu co 10000 kroków

4 Podsumowanie i wnioski

Pierwsza część zadania polegała na przetestowaniu działania sieci neuronowych dla prostego równania różniczkowego $\frac{du(x)}{dx} = cos(\omega x)$ na dziedzinie $[-2\pi, 2\pi]$ oraz z warunkiem początkowym $u(0) = 0$.
Przyjęliśmy ustalone wartości takich parametrów, jak **liczba kroków treningowych** (50000), **algorytm optymalizacyjny** (Adam), **stałą uczenia** (0.001) oraz **funkcję aktywacji** ($tanh$).

Z kolei zmienialiśmy liczbę warstw ukrytych sieci neuronowej oraz liczbę neuronów w każdej warstwie, rozważaliśmy także przypadki dla różnych wartości ω. Dla wartości $\omega = 1$ prosta sieć neuronowa dawała satysfakcjonujący rezultat, natomiast dla przypadku $\omega = 15$ testowaliśmy trzy sieci, gdzie pierwsza miała takie same parametry jak dla $\omega = 1$, a każda kolejna miała zwiększane wspomniane parametry.

Druga i trzecia sieć pozwoliły uzyskać dość dokładne rozwiązania, chociaż wciąż były one gorszej jakości niż to dla $\omega = 1$, ponadto ostatnia sieć dała gorszy wynik niż druga, mimo większego kosztu obliczeniowego, który (intuicyjnie) powinien przełożyć się na dokładniejsze rozwiązanie.

Dlatego kiedy w dalszej części zadania potrzebowaliśmy wybrać jedną z wcześniej badanych sieci neuronowych w celach porównania, wybraliśmy tę pierwszą, bowiem dawała ona najdokładniejsze rozwiązanie. Najpierw porównaliśmy ją z siecią, w której przyjęto jako rozwiązanie rezultat uzyskany z pierwotnej sieci neuronowej przemnożony przez funkcję $\tanh(\omega x)$. Pozwala to z funkcji kosztu usunąć składnik dotyczący warunku początkowego, gdyż $\hat{u}(0) = 0$ w każdym kroku.

Również zaobserwowaliśmy dużo bardziej przewidywalne zachowanie funkcji błędu w porównaniu do przypadku a, co sugeruje, że moglibyśmy stosować daną metodę dla aproksymacji pewnego, interesującego nas przedziału dziedziny, żeby dostać rozwiązanie,

które zachowuje się bardziej "naturalnie"(nie posiada gwałtownym zmian ruchu oddalających go od rzeczywistego rozwiązania).

Badając sieć z warstwą neuronów zainicjalizowaną cechami Fouriera, otrzymaliśmy zbieżność (funkcji kosztu) oraz funkcję końcową bardzo podobną do przypadku a. Natomiast funkcja błędu badanej metody okazała się być bardziej zbliżona do zera na całej dziedzinie. Nie zaobserwowaliśmy wykresu w postaci litery "V", tak jak w przypadku a, tylko równomierne oscylacje o małych wartościach.

Również zastosowania cech Fouriera opisane w danej publikacji [2] mogłyby sugerować o zwiększonej efektywności cech Fouriera w przypadkach dla większych ω. Gdyż sieci z warstwą neuronową zainicjalizowaną cechami Fouriera wykazują się dużo szybszą zbieżnością funkcji kosztu w porównaniu do zwykłych MLP. A właśnie w przypadkach dla dużego ω, czas uczenia sieci rósł eksponencjalnie, a dokładność często tak przewidywalna, jak w prostszych przypadkach.

Literatura

[1] Materiały pomocnicze do laboratorium zamieszczone na platformie Teams w katalogu *lab10*.

[2] Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains : `https://bmild.github.io/fourfeat/`.