Artur Gęsiarz,
Czerwiec 14, 2024

Laboratorium nr. 10
MOwNiT – Równania różniczkowe – spectral bias

1. Treść zadań

1.1. Zadanie pierwsze

Dane jest równanie różniczkowe zwyczajne

$$\frac{du(x)}{dx} = \cos(w * x)\ dla\ x \in\ \Omega\ (1)$$

gdzie,
$x, w, u\ x \in R$
$x\ to\ położenie$
Ω to dziedzina, na której rozwiązujemy równanie, $\Omega = \{\mathrm{x} \mid -2\pi \leq \mathrm{x} \leq 2\pi\}$
$u(.)$ to to funkcja, której postaci szukamy

Warunek początkowy zdefiniowany jest następująco

$$u(0) = 0\ (2)$$

Analityczna postać rozwiązania równania (1) z warunkiem początkowym (2) jest następująca:

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

Rozwiąż powyższe zagadnienie początkowe (1,2). Do rozwiązania użyj sieci neuronowych PINN (ang. Physics-informed Neural Network) [1]. Można wykorzystać szablon w pytorch-u lub biblioteke DeepXDE [2].
Koszt rezydualny zdefiniowany jest nastepujaco:

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

gdzie N jest liczba punktów kolokacyjnych.
Koszt zwiazany z warunkiem poczatkowym przyjmuje postac:

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

Funkcja kosztu zdefiniowana jest nastepujaco:

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

Warstwa wejsciowa sieci posiada 1 neuron, reprezentujacy zmienna x, Warstwa wyjsciowa takze posiada 1 neuron, reprezentujacy zmienna ^u(x). Uczenie trwa przez 50 000 kroków algorytmem Adam ze stała uczenia równa 0.001. Jako funkcje aktywacji przyjmij tangens hiperboliczny, tanh

(a) Przypadek $\omega = 1$
Ustal następujące wartości:
– 2 warstwy ukryte, 16 neuronów w kazdej warstwie
– liczba punktów treningowych: 200
– liczba punktów testowych: 1000

(b) Przypadek $\omega = 15$
Ustal nastepujace wartosci:
– liczba punktów treningowych: 3000
– liczba punktów testowych: 5000

Eksperymenty przeprowadz z trzema architekturami sieci:
– 2 warstwy ukryte, 16 neuronów w kazdej warstwie
– 4 warstwy ukryte, 64 neurony w kazdej warstwie
– 5 warstw ukrytych, 128 neuronów w kazdej warstwie

(c) Dla wybranej przez siebie sieci porównaj wynik z rozwiązaniem, w którym przyjęto, ze szukane rozwiazanie (ansatz) ma postać:

$$\hat{u}(x;\theta) = \tanh(\omega x) * NN(x;\theta)$$

Taka postać rozwiązania gwarantuje spełnienie warunku $\hat{u}(0) = 0$ bez wprowadzania Składnika $\mathcal{L}_{IC}$ do funkcji kosztu.

(d) Porównaj pierwotny wynik z rozwiązaniem, w którym pierwsza warstwę ukryta zainicjalizowano cechami Fouriera:

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

Dobierz L tak, aby nie zmieniać szerokości warstwy ukrytej

Dla każdego z powyższych przypadków stwórz następujące wykresy:

– Wykres funkcji u(x), tj. dokładnego rozwiązania oraz wykres funkcji ^u(x), tj. rozwiązania znalezionego przez siec neuronowa

– Wykres funkcji błędu.

Stwórz także wykres funkcji kosztu w zależności od liczby epok.

2. Rozwiązanie zadania

2.1. Implementacja zadania pierwszego

2.1.1. Definicja analitycznego rozwiązania

```
def exact_solution(x, w):
    return (1/w) * torch.sin(w * x)
```

2.1.2. Definicja sieci neuronowej

```
class FCN(nn.Module):
    def __init__(self, N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS):
        super().__init__()
        activation = nn.Tanh

        # warstwa poczatkowa
        self.fcs = nn.Sequential(*[
                        nn.Linear(N_INPUT, N_HIDDEN),
                        activation()])
        # warstwa ukrytna
        self.fch = nn.Sequential(*[
                        nn.Sequential(*[
                            nn.Linear(N_HIDDEN, N_HIDDEN),
                            activation()]) for _ in range(N_LAYERS-1)])

        # warstwa wyjsciowa
        self.fce = nn.Linear(N_HIDDEN, N_OUTPUT)
    def forward(self, x):
        x = self.fcs(x)
        x = self.fch(x)
        x = self.fce(x)
        return x
```

2.1.3. Funkcja licząca koszt warunku początkowego

```
def calculate_cond_start_cost(model, x_boundary):
    u0 = model(x_boundary)
    loss_ic = u0 ** 2
    return  loss_ic
```

2.1.4. Funkcja obliczająca koszt rezydualny

```
def calculate_residual_cost(model, x_physics, w):
    u = model(x_physics)
    du_dx = torch.autograd.grad(u, x_physics, torch.ones_like(u), create_graph=True)[0]
    residual = du_dx - torch.cos(w * x_physics)
    loss_r = torch.mean(residual**2)
    return loss_r
```

2.1.5. Funkcja obliczająca koszt totalny

```
def calculate_total_cost(model, x_boundary, x_physics, w):
    return calculate_cond_start_cost(model, x_boundary) + calculate_residual_cost(model, x_physics, w)
```

2.1.6. Funkcja treningowa PINN

```python
def train_PINN(model, x_boundary, x_physics, cost_fun, w, epochs=50000, lr=0.001):
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    losses = []

    for i in range(epochs):
        optimizer.zero_grad()

        loss = cost_fun(model, x_boundary, x_physics, w)
        losses.append(loss.item())

        loss.backward()
        optimizer.step()

        if i % 5000 == 0:
            print(f'Epoch {i}, Loss: {loss.item()}')

    return model, losses
```

2.1.7. Funkcja do rysowania wyników

```python
def plot_results(x_test, u_exact, u_pred, losses, title):

    plt.plot(x_test, u_exact, label='Exact solution')
    plt.plot(x_test, u_pred, '--', label='PINN solution')
    plt.xlabel('x')
    plt.ylabel('u(x)')
    plt.title(title + ': Solution')
    plt.legend()
    plt.show()

    plt.plot(x_test, abs((u_exact - u_pred)))
    plt.xlabel('x')
    plt.ylabel('error')
    plt.title(title + ': Error function')
    plt.show()

    plt.plot(losses)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(title + ': Training Loss')
    plt.show()
```

2.1.8. Stałe w naszym modelu

```python
N_INPUT = 1
N_OUTPUT = 1
LR = 0.001
EPOCHS = 50000
```

2.1.9. Przypadek $\omega = 1$

2.1.9.1. Parametry naszego modelu

```python
N_LAYERS = 2
N_HIDDEN = 16
TRAINING_POINTS = 200
TESTING_POINTS = 1000
OMEGA = 1
```

2.1.9.2. Definicja modelu

```
model_a = FCN(N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS)
```

2.1.9.3. Definicja punktów treningowych i testowych

```
x_boundary_a = torch.tensor([[0.0]], requires_grad=True)
x_physics_a = torch.linspace(-2 * np.pi, 2 * np.pi, TRAINING_POINTS).view(-1, 1).requires_grad_(True)
x_test_a = torch.linspace(-2 * np.pi, 2 * np.pi, TESTING_POINTS).view(-1, 1)
u_exact_a = exact_solution(x_test_a, OMEGA)
```

2.1.9.4. Trening modelu

```
print(f'Training for w = {OMEGA}, Layers = {N_LAYERS}, Neurons = {N_HIDDEN}\n')
model_a ,losses_a = train_PINN(model_a, x_boundary_a, x_physics_a, calculate_total_cost, OMEGA, EPOCHS, LR)
```

2.1.9.5. Przewidywanie wartości

```
u_pred_a = model_load_a(x_test_a).detach().numpy()
u_exact_a = u_exact_a.numpy()
```

2.1.9.6. Rysowanie wyników

```
plot_results(x_test_a, u_exact_a, u_pred_a, losses_a, f'w = {OMEGA}, Layers = {N_LAYERS}, Neurons =
 {N_HIDDEN}')
```

2.1.10. Przypadek $\omega = 15$

2.1.10.1. Parametry naszego modelu

```
TRAINING_POINTS = 3000
TESTING_POINTS = 5000
OMEGA = 15
ARCH = [(2,16), (4,64), (5,128)]
```

2.1.10.2. Definicja modelu, definicja punktów treningowych i testowych, …, wykres wyników

```
for N_LAYERS, N_HIDDEN in ARCH:

    # Definicja modelu
    model_b = FCN(N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS)

    # Definicja punktów treningowych i testowych
    x_boundary_b = torch.tensor([[0.0]], requires_grad=True)
    x_physics_b = torch.linspace(-2 * np.pi, 2 * np.pi, TRAINING_POINTS).view(-1, 1).requires_grad_(True)
    x_test_b = torch.linspace(-2 * np.pi, 2 * np.pi, TESTING_POINTS).view(-1, 1)
    u_exact_b = exact_solution(x_test_b, OMEGA)

    # Trening
    print(f'Training for w = {OMEGA}, Layers = {N_LAYERS}, Neurons = {N_HIDDEN}\n')
    model_b, losses_b = train_PINN(model_b, x_boundary_b, x_physics_b, calculate_total_cost,OMEGA, EPOCHS, LR

    # Zapis
    torch.save(model_b, f"models/model_b_{N_LAYERS}_{N_HIDDEN}.pth")

    # Odczyt
    model_load_b = torch.load(f"models/model_b_{N_LAYERS}_{N_HIDDEN}.pth")

    # Przewidywanie wartości
    u_pred_b = model_load_b(x_test_b).detach().numpy()
    u_exact_b = u_exact_b.numpy()

    # Rysowanie wykresow
    plot_results(x_test_b, u_exact_b, u_pred_b, losses_b, f'w = {OMEGA}, Layers = {N_LAYERS}, Neurons =
     {N_HIDDEN}')
```

2.1.11. Porównanie wyniku wybranej sieci z rozwiązaniem w którym przyjęto, ze szukane rozwiązanie ma konkretną postać

2.1.11.1. Funkcja obliczająca koszt totalny

```
def calculate_total_cost_anastaz(model, _, x_physics, w):
    u = torch.tanh(w * x_physics) * model(x_physics)
    u_x = torch.autograd.grad(u, x_physics, torch.ones_like(u), create_graph=True)[0]
    loss_r = torch.mean(torch.pow((u_x - torch.cos(w * x_physics)),2))
    return loss_r
```

2.1.11.2. Parametry naszego modelu

```
N_LAYERS = 5
N_HIDDEN = 128
TRAINING_POINTS = 3000
TESTING_POINTS = 5000
OMEGA = 15
```

2.1.11.3. Definicja modelu

```
model_c = FCN(N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS)
```

2.1.11.4. Definicja punktów treningowych i testowych

```
x_physics_c = torch.linspace(-2 * np.pi, 2 * np.pi, TRAINING_POINTS).view(-1, 1).requires_grad_(True)
x_test_c = torch.linspace(-2 * np.pi, 2 * np.pi, TESTING_POINTS).view(-1, 1)
u_exact_c = exact_solution(x_test_c, OMEGA)
```

2.1.11.5. Trening modelu

```
print(f'ansatz - Training for w = {OMEGA}, Layers = {N_LAYERS}, Neurons = {N_HIDDEN}\n')
model_c, losses_c = train_PINN(model_c, _, x_physics_c, calculate_total_cost_anastaz, OMEGA, EPOCHS, LR)
```

2.1.11.6. Przewidywanie wartości

```
u_pred_c = model_c_load(x_test_c).detach().numpy()
u_exact_c = u_exact_c.numpy()
```

2.1.11.7. Rysowanie wyników

```
plot_results(x_test_c, u_exact_c, u_pred_c, losses_c, f'ansatz - w = {OMEGA}, Layers = {N_LAYERS}, Neurons =
 {N_HIDDEN}')
```

2.1.12. Porównanie pierwotnego wyniku z rozwiązaniem, w którym pierwszą warstwę ukrytą zainicjalizowano cechami Fouriera

2.1.12.1. Warstwa Fouriera

```
class FourierLayer(nn.Module):
    def __init__(self, N_INPUT, N_HIDDEN):
        super(FourierLayer, self).__init__()
        self.N_HIDDEN = N_HIDDEN
        self.L = N_HIDDEN // 2
        self.linear = nn.Linear(2 * self.L, N_HIDDEN)

    def forward(self, x):
        # Tworzymy cechy Fouriera
        features = []
        for l in range(1, self.L + 1):
            features.append(torch.sin((2 ** l) * np.pi * x))
            features.append(torch.cos((2 ** l) * np.pi * x))
        features = torch.cat(features, dim=-1)
        return self.linear(features)
```

2.1.12.2. Model Fouriera

```python
class FourierFCN(nn.Module):
    "Defines a fully-connected network in PyTorch"
    def __init__(self, N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS):
        super(FourierFCN, self).__init__()

        self.fcs = FourierLayer(N_INPUT, N_HIDDEN)
        activation = nn.Tanh

        self.fch = nn.Sequential(*[
            nn.Sequential(nn.Linear(N_HIDDEN, N_HIDDEN), activation())
            for _ in range(N_LAYERS - 1)
        ])

        self.fce = nn.Linear(N_HIDDEN, N_OUTPUT)

    def forward(self, x):
        x = self.fcs(x)
        x = self.fch(x)
        x = self.fce(x)
        return x
```

2.1.12.3. Parametry naszego modelu

```python
N_HIDDEN = 16
N_LAYERS = 2
TRAINING_POINTS = 3000
TESTING_POINTS = 5000
OMEGA = 15
```

2.1.12.4. Definicja modelu

```python
model_d = FourierFCN(N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS)
```

2.1.12.5. Definicja punktów treningowych i testowych

```python
x_boundary_d = torch.tensor([[0.0]], requires_grad=True)
x_physics_d = torch.linspace(-2 * np.pi, 2 * np.pi, TRAINING_POINTS).view(-1, 1).requires_grad_(True)
x_test_d = torch.linspace(-2 * np.pi, 2 * np.pi, TESTING_POINTS).view(-1, 1)
u_exact_d = exact_solution(x_test_d, OMEGA)
```

2.1.12.6. Trening modelu

```python
print(f'fourier - Training for w = {OMEGA}, Layers = {N_LAYERS}, Neurons = {N_HIDDEN}\n')
model_d, losses_d = train_PINN(model_d, x_boundary_d, x_physics_d, calculate_total_cost, OMEGA, EPOCHS, LR)
```

2.1.12.7. Przewidywanie wartości

```python
u_pred_d = model_d_load(x_test_d).detach().numpy()
u_exact_d = u_exact_d.numpy()
```

2.1.12.8. Rysowanie wyników

```python
plot_results(x_test_d, u_exact_d, u_pred_d, losses_d, f'fourier - w = {OMEGA}, Layers = {N_LAYERS}, Neurons = {N_HIDDEN}')
```

3. Tabele

3.1. Tabela wyników uczenia się dla przypadku $\omega = 1$

3.1.1. Zawierająca: 2 warstwy ukryte, 16 neuronów w każdej warstwie

Epoch	Loss
0	$0.58 * 10^0$
5000	$1.47 * 10^{-5}$
10000	$8.96 * 10^{-6}$
15000	$6.76 * 10^{-6}$
20000	$5.62 * 10^{-6}$
25000	$4.73 * 10^{-6}$
30000	$4.30 * 10^{-6}$
35000	$1.83 * 10^{-5}$
40000	$4.39 * 10^{-5}$
45000	$2.57 * 10^{-6}$

Tabela 1. Tabela wyników kosztu dla $\omega = 1$, 2 warstwy ukryte, 16 neuronów w każdej warstwie

3.2. Tabela wyników uczenia się dla przypadku $\omega = 15$

3.2.1. Zawierający: 2 warstwy ukryte, 16 neuronów w każdej warstwie

Epoch	Loss
0	0.50
5000	0.5
10000	0.5
15000	0.37
20000	0.33
25000	0.3
30000	0.28
35000	0.27
40000	0.26
45000	0.26

Tabela 2. Tabela wyników kosztu dla $\omega = 15$, 2 warstwy ukryte, 16 neuronów w każdej warstwie

3.2.2. Zawierający: 4 warstwy ukryte, 64 neuronów w każdej warstwie

Epoch	Loss
0	0.50
5000	0.5
10000	0.5
15000	0.5
20000	0.36
25000	0.22
30000	0.19
35000	0.15
40000	0.13
45000	0.12

Tabela 3. Tabela wyników kosztu dla $\omega = 15$, 4 warstwy ukryte, 64 neuronów w każdej warstwie

3.2.3. Zawierający: 5 warstwy ukryte, 128 neuronów w każdej warstwie

Epoch	Loss
0	0.50
5000	0.5
10000	0.5
15000	0.5
20000	0.5
25000	0.5
30000	0.5
35000	0.31
40000	0.01
45000	0.001

Tabela 4. Tabela wyników kosztu dla $\omega = 15$, 5 warstwy ukryte, 128 neuronów w każdej warstwie

3.3. Tabela wyników uczenia się dla przypadku wyniku wybranej sieci z rozwiązaniem, w którym przyjęto, ze szukane rozwiązanie ma konkretną postać

3.3.1. Zawierający: 5 warstwy ukryte, 128 neuronów w każdej warstwie

Epoch	Loss
0	0.49
5000	0.27
10000	0.07
15000	0.005
20000	0.009
25000	0.003
30000	0.0005
35000	0.0009
40000	0.005
45000	0.0006

Tabela 5. Tabela wyników kosztu dla $\omega = 15$, 5 warstwy ukryte, 128 neuronów w każdej warstwie

3.4. Tabela wyników uczenia się w którym pierwszą warstwę ukrytą zainicjalizowano cechami Fouriera

3.4.1. Zawierający: 2 warstwy ukryte, 16 neuronów w każdej warstwie

Epoch	Loss
0	2060
5000	0.5
10000	0.49
15000	0.34
20000	0.33
25000	0.32
30000	0.31
35000	0.30
40000	0.309
45000	0.30

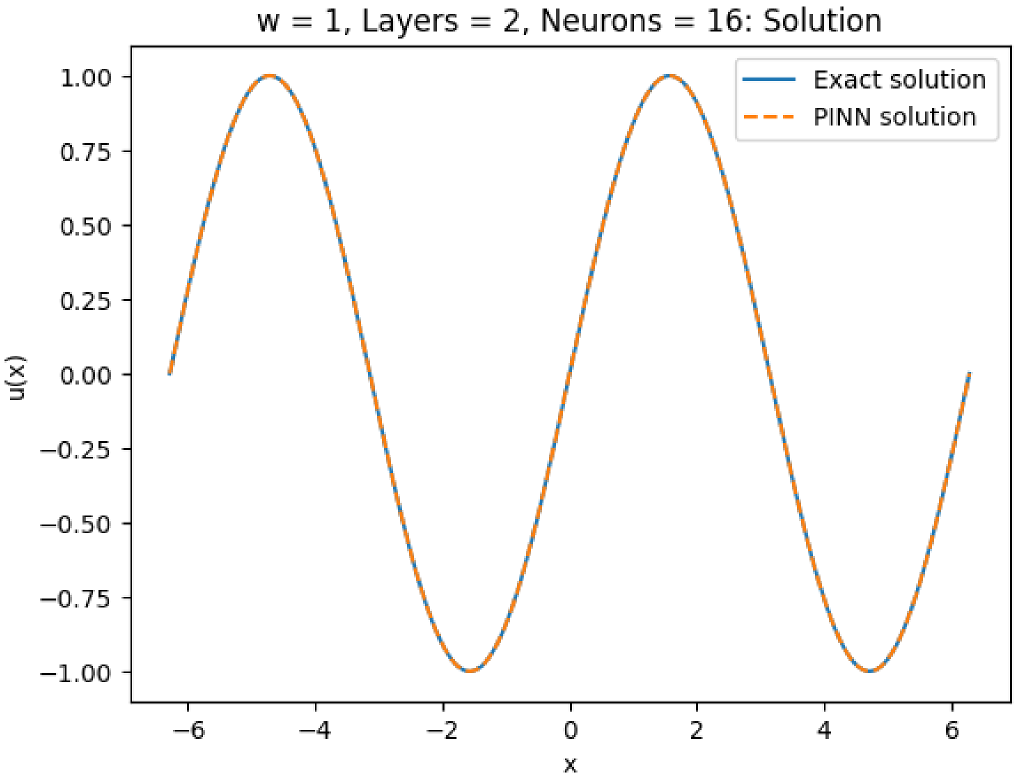
Tabela 6. Tabela wyników kosztu dla $\omega = 15$, 2 warstwy ukryte, 16 neuronów w każdej warstwie

4. Wykresy

4.1. Wykres dla przypadku $\omega = 1$

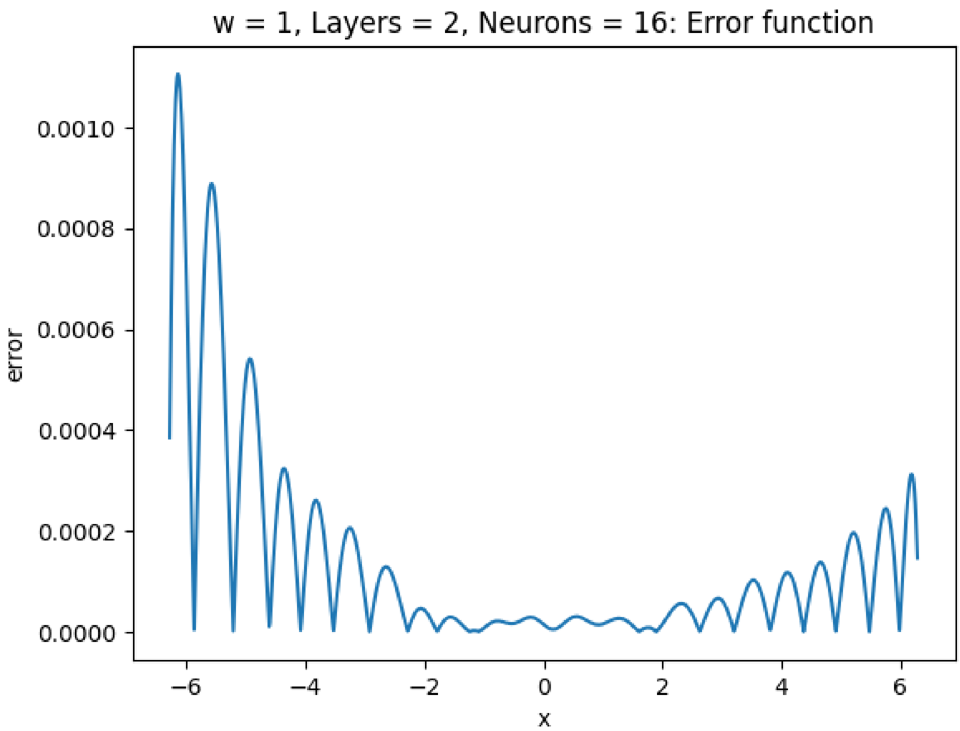
4.1.1. Zawierający: 2 warstwy ukryte, 16 neuronów w każdej warstwie

4.1.1.1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$



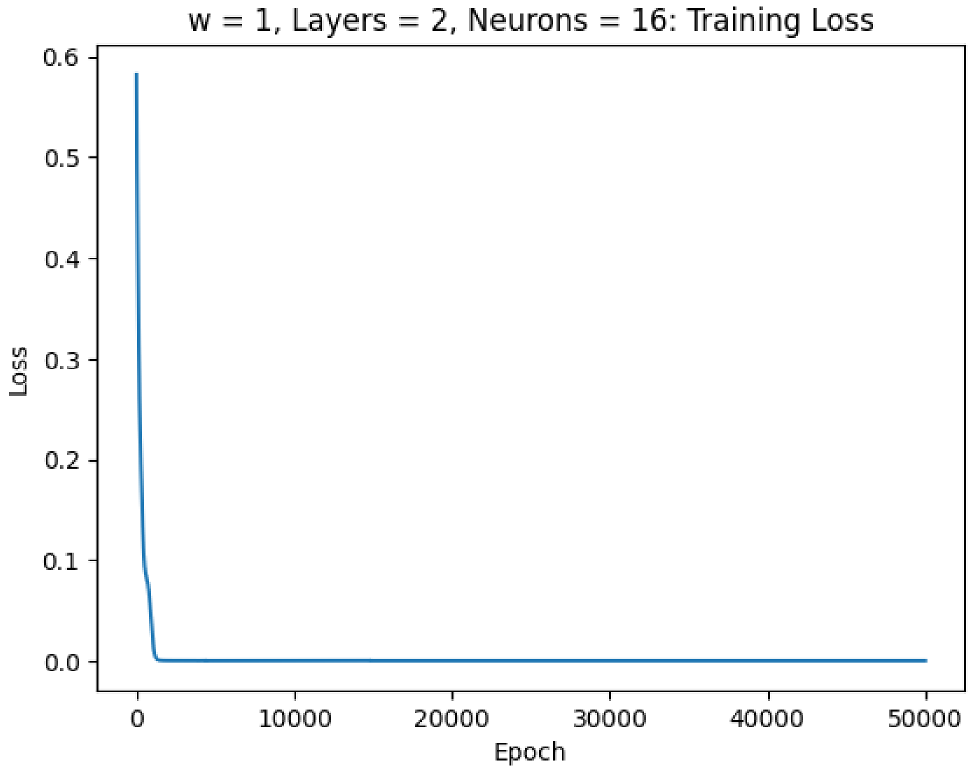
Wykres 1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$ dla $\omega = 1$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

4.1.1.2. Wykres funkcji błędu



Wykres 2. Wykres funkcji *błędu* dla $\omega = 1$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

4.1.1.3. Wykres funkcji kosztu w zależności od liczby epok

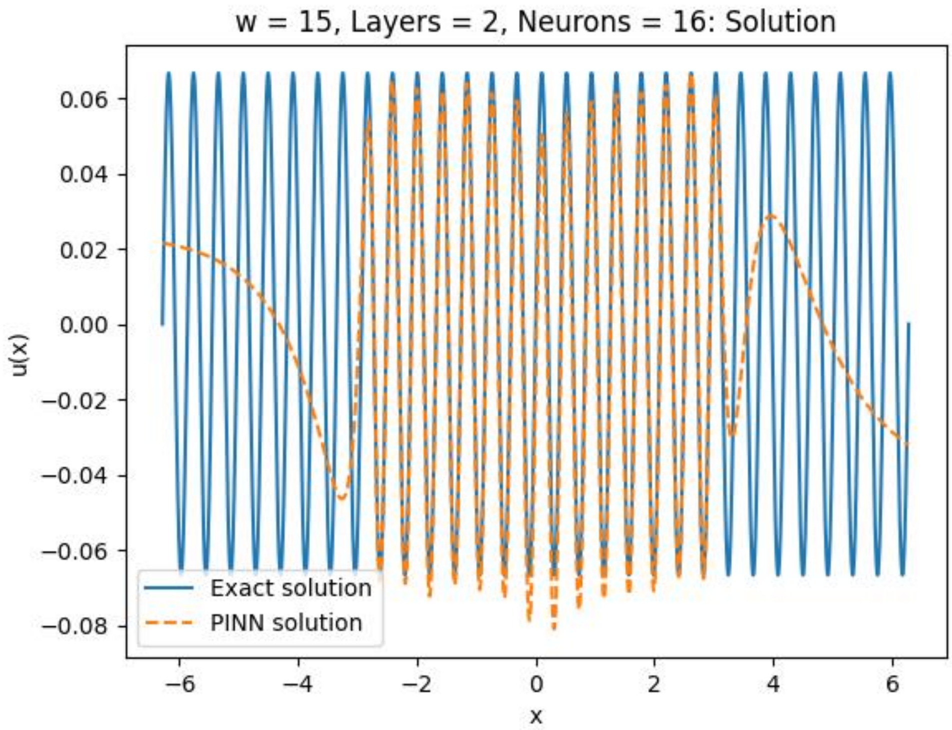


Wykres 3. Wykres funkcji kosztu dla ω=1, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

4.2. Wykres dla przypadku $\omega = 15$

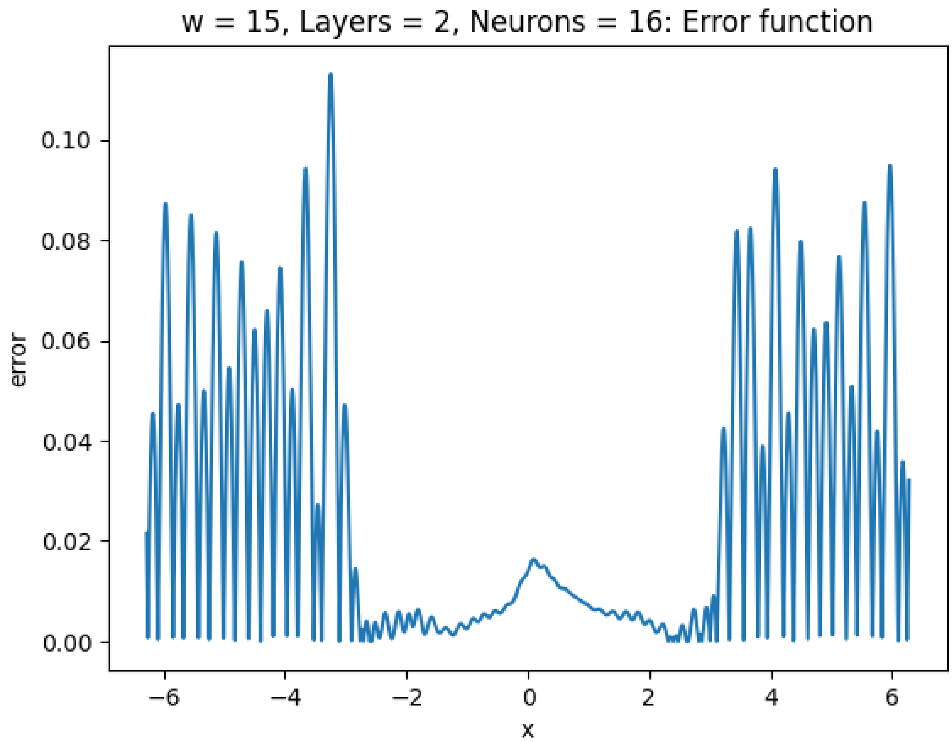
4.2.1. Zawierający: 2 warstwy ukryte, 16 neuronów w każdej warstwie

4.2.1.1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$



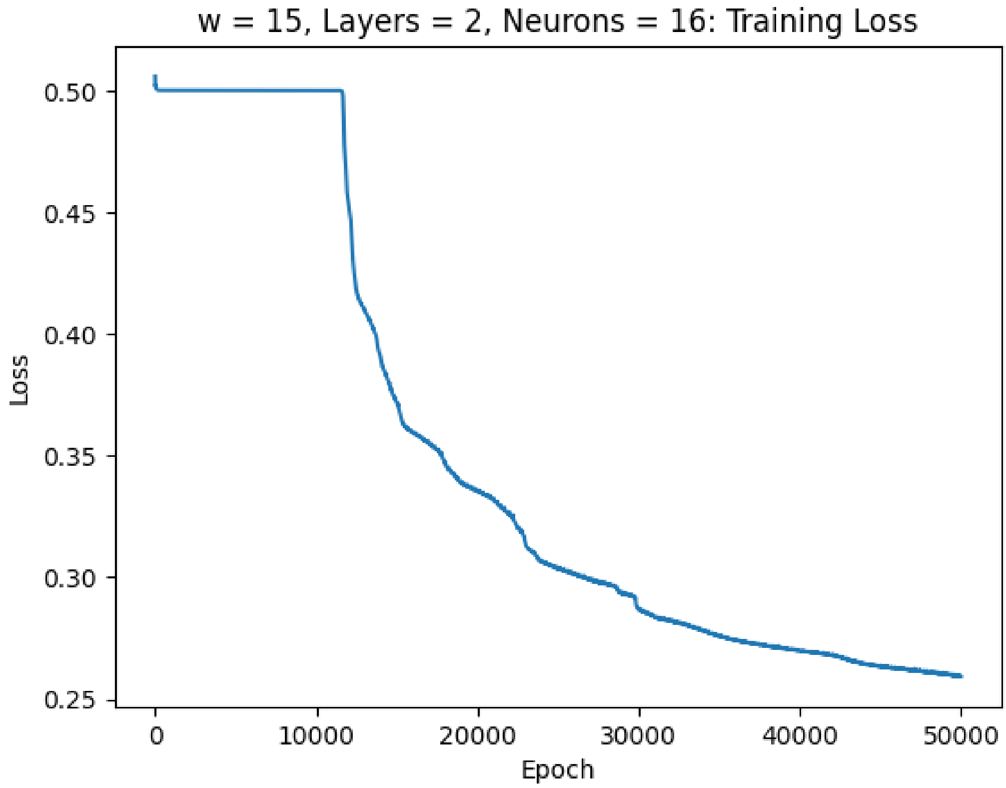
Wykres 4. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$ dla $\omega = 15$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

4.2.1.2. Wykres funkcji błędu



Wykres 5. Wykres funkcji *błędu* dla $\omega = 15$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

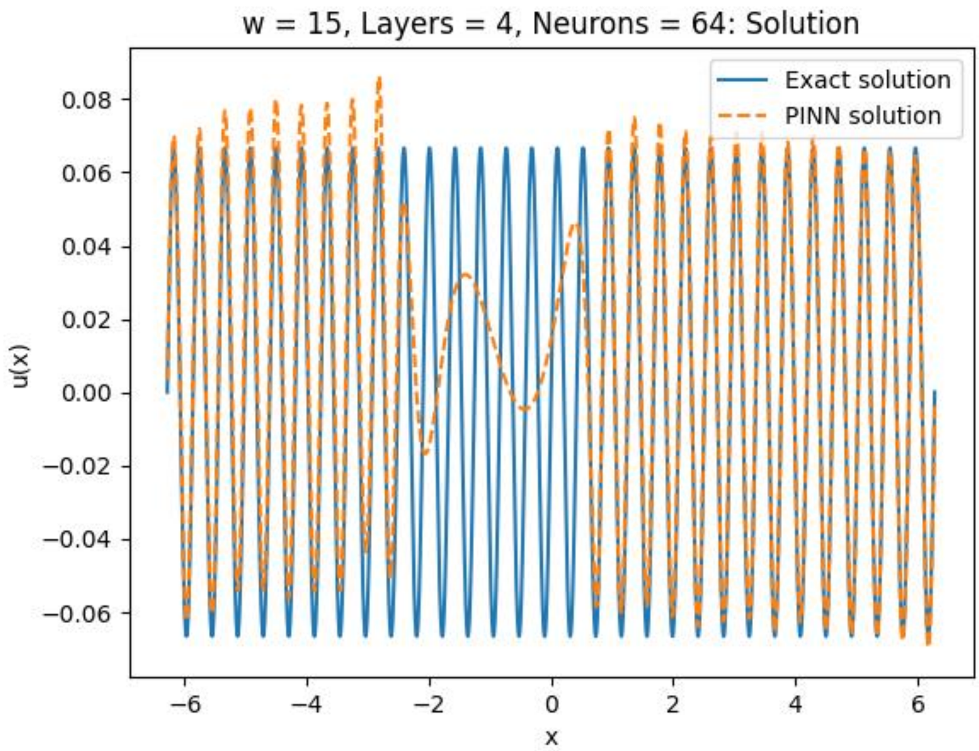
4.2.1.3. Wykres funkcji kosztu w zależności od liczby epok



Wykres 6. Wykres funkcji kosztu dla ω=15, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

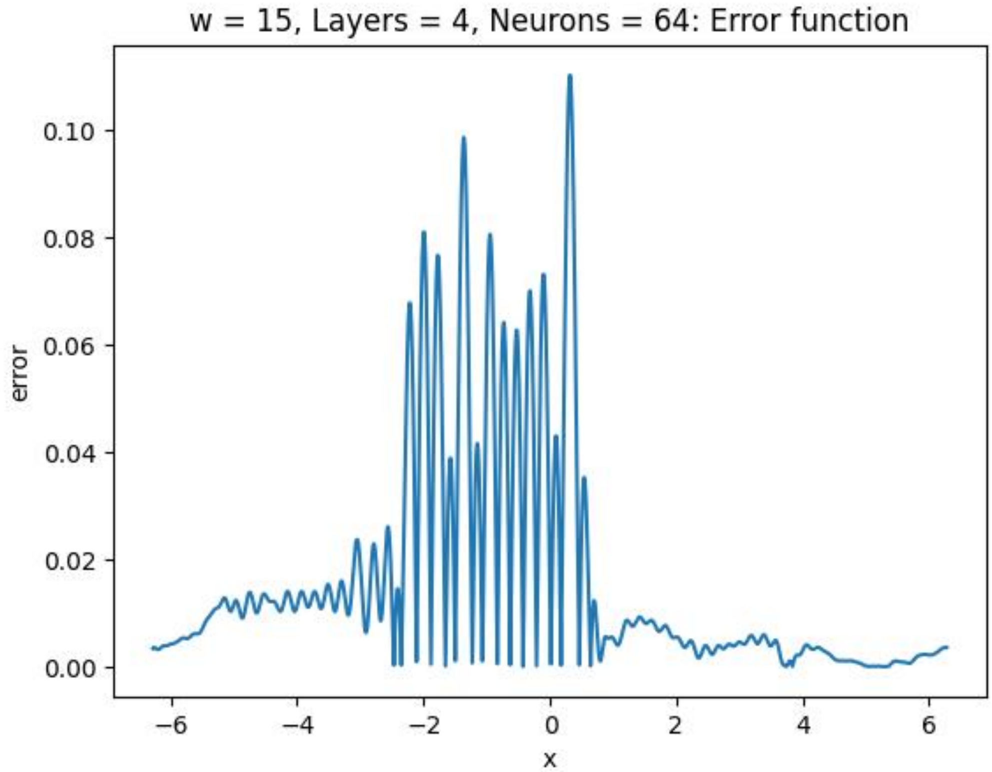
4.2.2. Zawierający: 4 warstwy ukryte, 64 neuronów w każdej warstwie

4.2.2.1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$



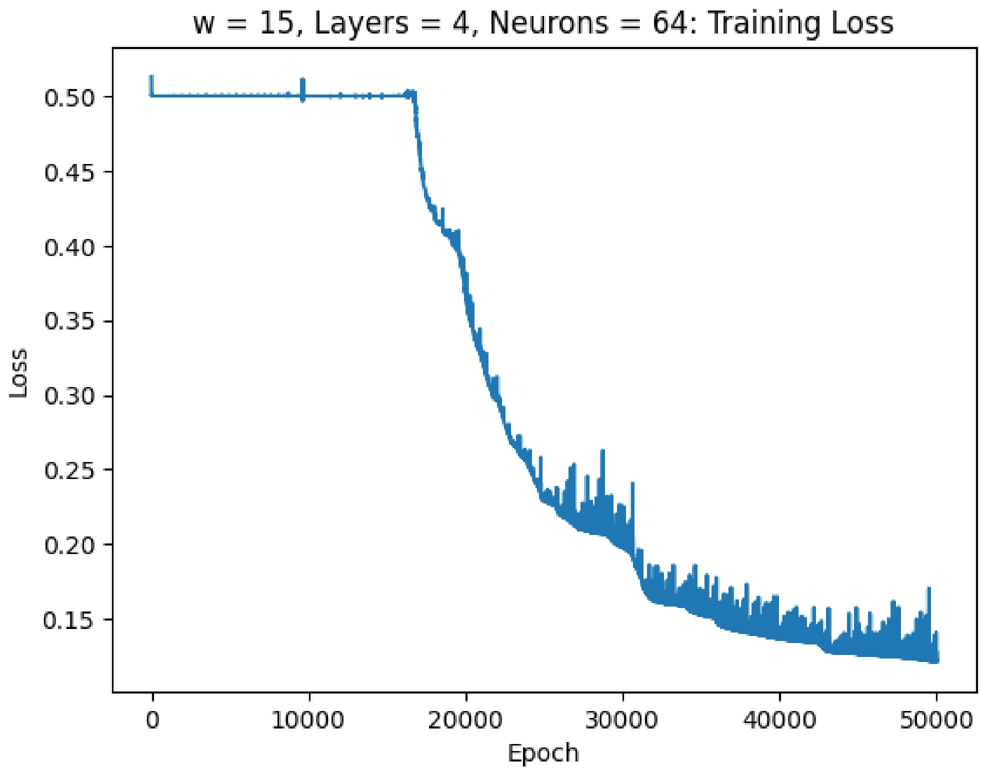
Wykres 7. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$ dla $\omega = 15$, 4 - warstw ukrytych, 64 - neuronów w każdej warstwie

4.2.2.2. Wykres funkcji błędu



Wykres 8. Wykres funkcji błędu dla $\omega = 15$, 4 - warstw ukrytych, 64 - neuronów w każdej warstwie

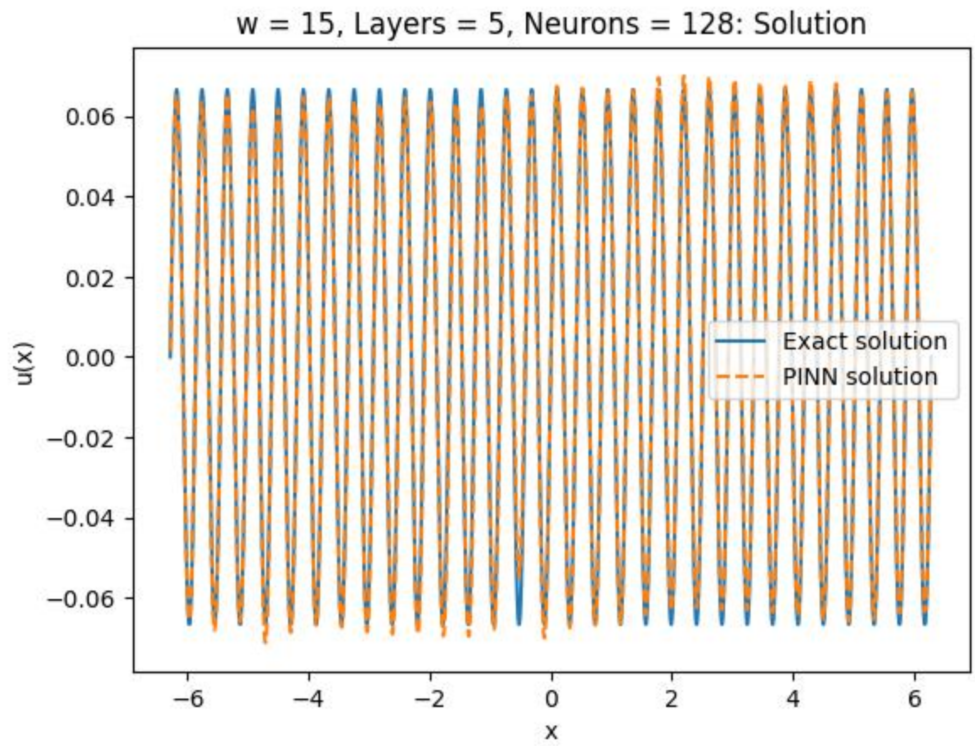
4.2.2.3. Wykres funkcji kosztu w zależności od liczby epok



Wykres 9. Wykres kosztu dla $\omega = 15$, 4 - warstw ukrytych, 64 - neuronów w każdej warstwie

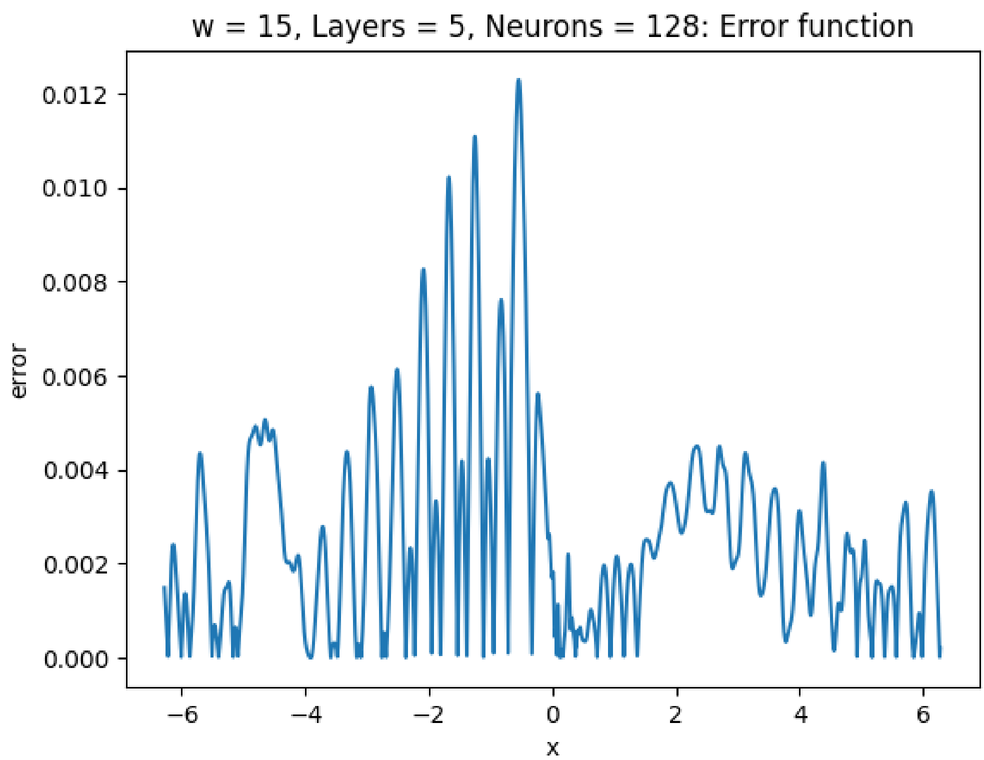
4.2.3. Zawierający: 5 warstw ukrytych, 128 neuronów w każdej warstwie

4.2.3.1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$



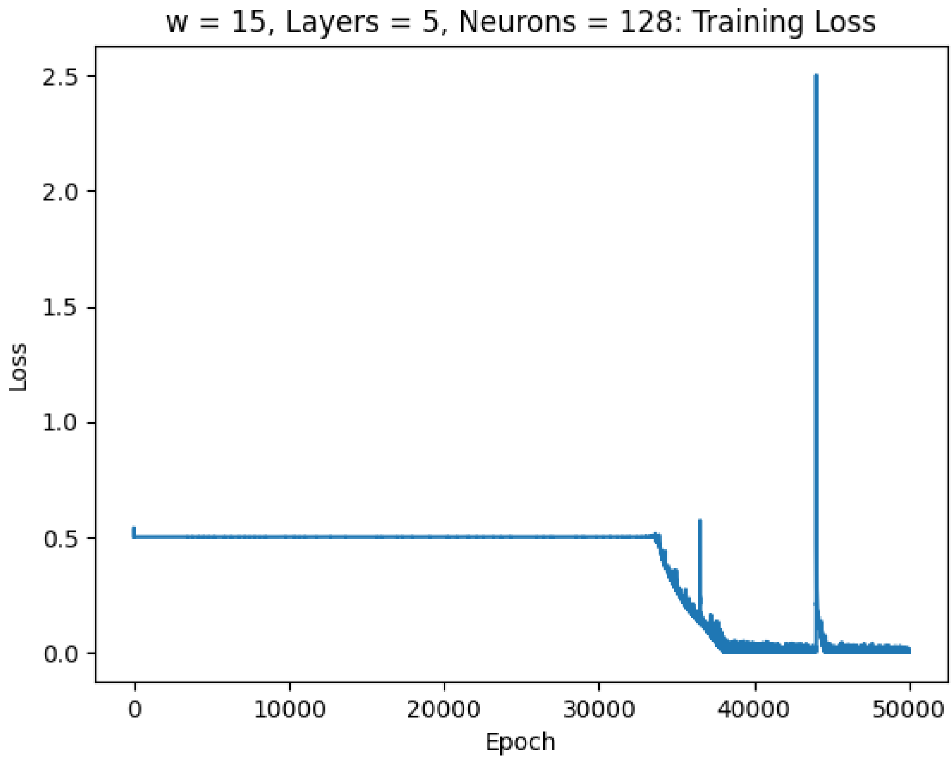
Wykres 10. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$ dla $\omega = 15$, 5 - warstw ukrytych, 128 - neuronów w każdej warstwie

4.2.3.2. Wykres funkcji błędu



Wykres 11. Wykres funkcji $błędu$ dla $\omega = 15$, 5 - warstw ukrytych, 128 - neuronów w każdej warstwie

4.2.3.3. Wykres funkcji kosztu w zależności od liczby epok

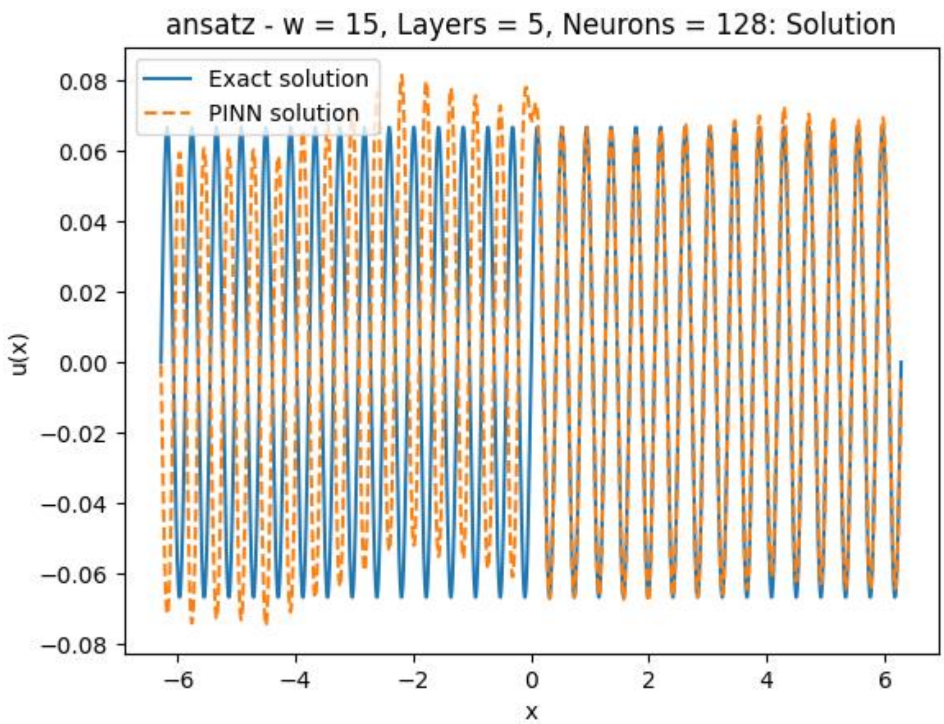


Wykres 12. Wykres funkcji $kosztu$ dla $\omega = 15$, 5 - warstw ukrytych, 128 - neuronów w każdej warstwie

4.3. Wykres dla wyniku wybranej sieci z rozwiązaniem w którym przyjęto, ze szukane rozwiązanie ma konkretną postać

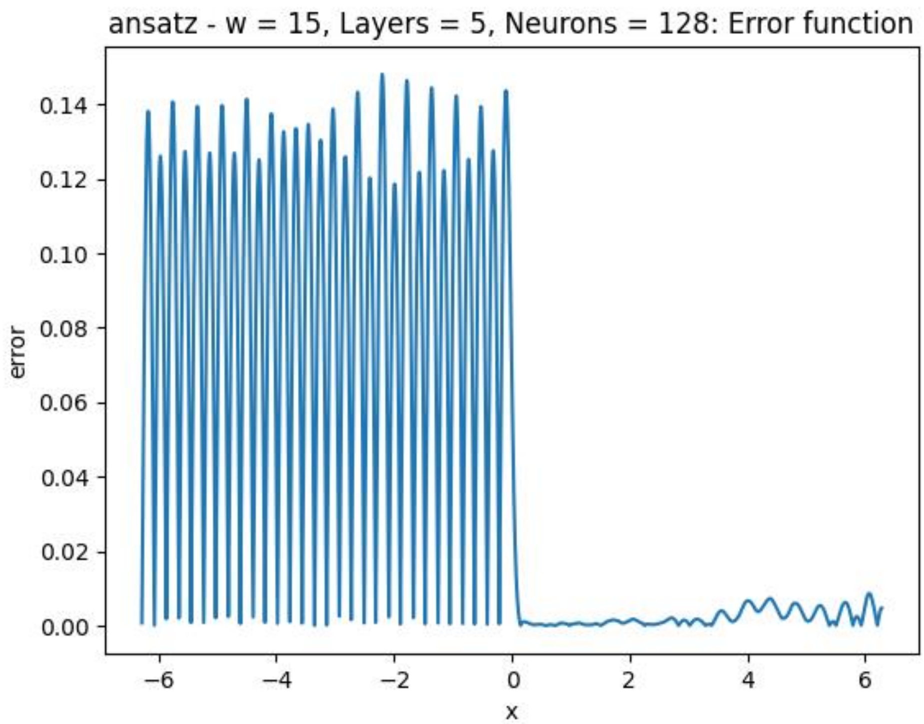
4.3.1. Zawierający: 5 warstw ukrytych, 128 neuronów w każdej warstwie, $\omega = 15$

4.3.1.1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$



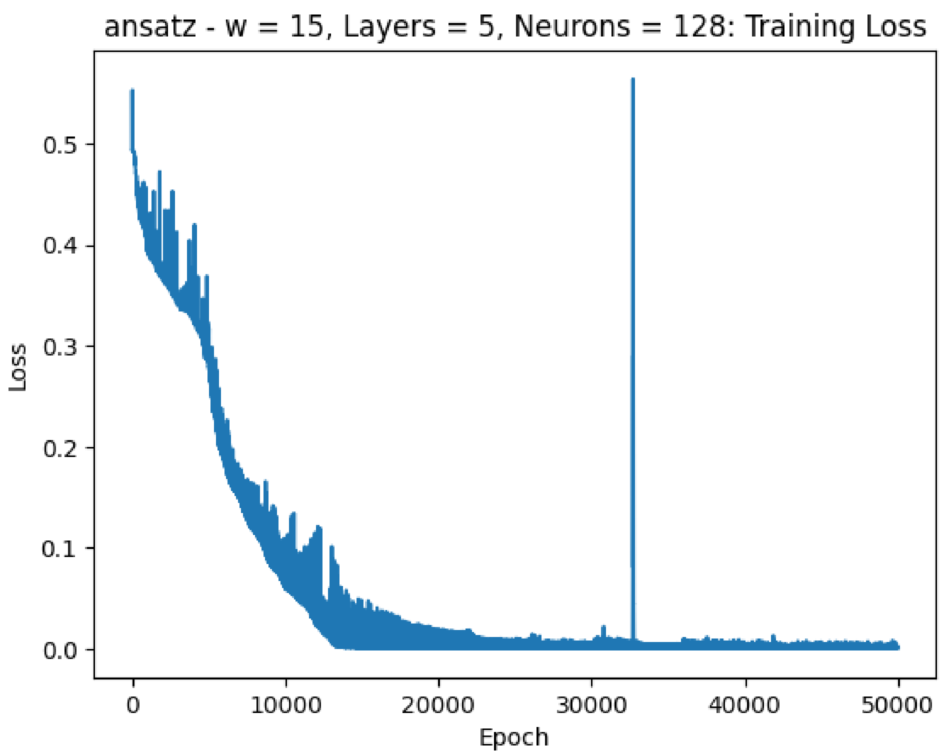
Wykres 13. Wykres funkcji anastaz $u(x)$, wraz z $\hat{u}(x)$ dla $\omega = 15$, 5 - warstw ukrytych, 128 - neuronów w każdej warstwie

4.3.1.2. Wykres funkcji błędu



Wykres 14. Wykres funkcji anastaz *błedu* dla $\omega = 15$, 5 - warstw ukrytych, 128 - neuronów w każdej warstwie

4.3.1.3. Wykres funkcji kosztu w zależności od liczby epok

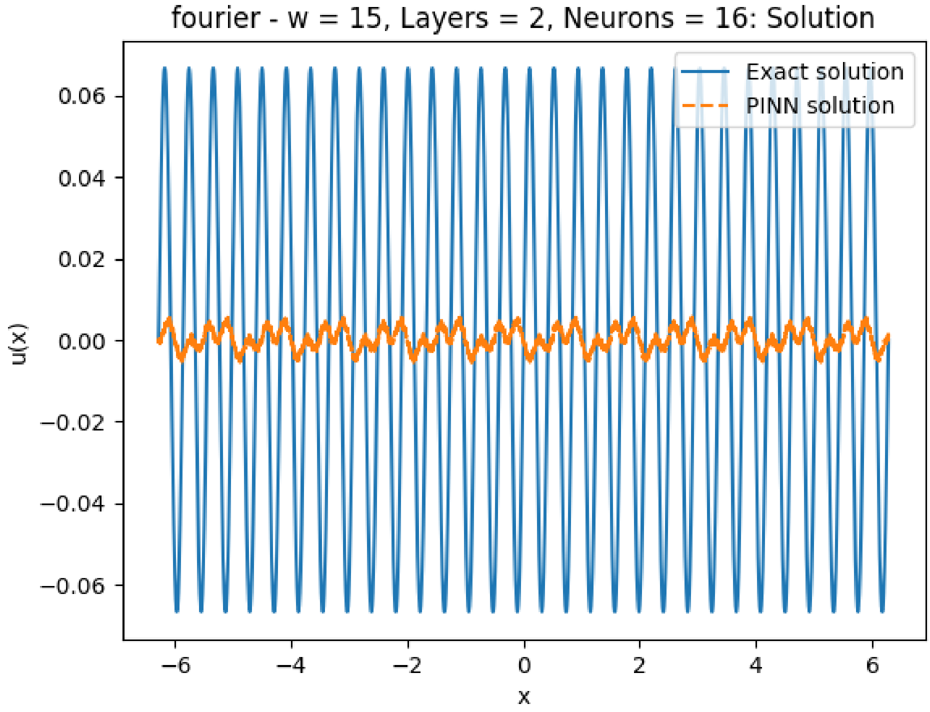


Wykres 15. Wykres funkcij kosztu dla $\omega = 15$, 5 - warstw ukrytych, 128 - neuronów w każdej warstwie

4.4. Wykres w którym pierwszą warstwę ukrytą zainicjalizowano cechami Fouriera

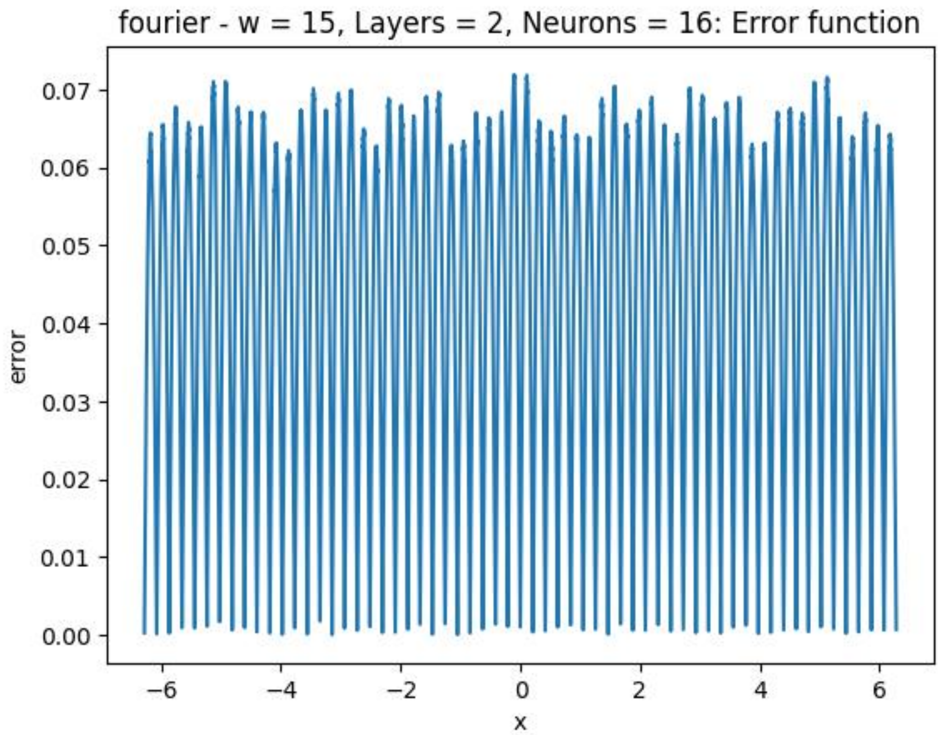
4.4.1. Zawierający: 2 warstwy ukryte, 16 neuronów w każdej warstwie, $\omega = 15$

4.4.1.1. Wykres funkcji $u(x)$, wraz z $\hat{u}(x)$



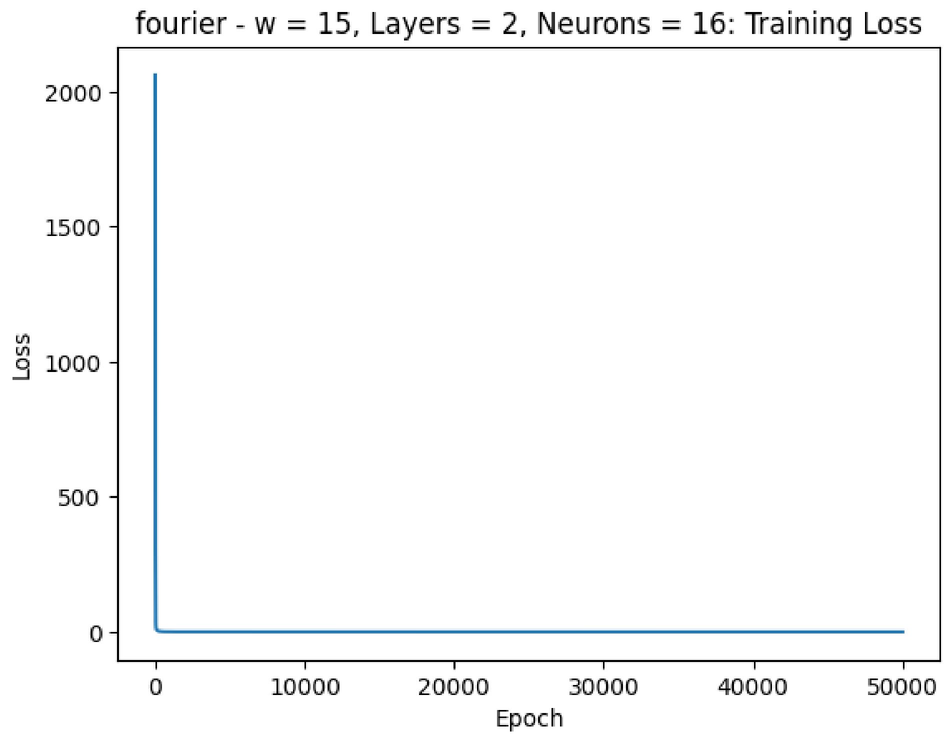
Wykres 16. Wykres funkcji fouriera $u(x)$, wraz z $\hat{u}(x)$ dla $\omega = 15$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

4.4.1.2. Wykres funkcji błędu



Wykres 17. Wykres funkcji błędu dla $\omega = 15$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

4.4.1.3. Wykres funkcji kosztu w zależności od liczby epok



Wykres 18. Wykres funkcji fouriera $kosztu$ dla $\omega = 15$, 2 - warstw ukrytych, 16 - neuronów w każdej warstwie

5. Wnioski
   Porównując wszystkie przypadki najlepiej sobie radził przypadek $\omega = 1$, dla 2 warstw ukrytych, 16 neuronów w każdej warstwie, ponieważ wykres funkcji prawdziwej i funkcji znalezionej dla tych parametrów idealnie się pokrywał z funkcją co jest widoczne na wykresie nr 1.

   Inne przypadki też radziły sobie dobrze kolejnym który wręcz wszystko pokrywał był przypadek $\omega = 15$ dla 5 warstw ukrytych, 128 neuronów w każdej warstwie. Ten przypadek chociaż poradził sobie wyśmienicie to jego wadą było to że ten przypadek był bardzo kosztowny obliczeniowo – wynik można zobaczyć na wykresie nr 10.

   Kolejnym przypadkiem, który poradził sobie bardzo dobrze był ten w którym przyjęto za szukany rozwiązanie wzór: $\hat{u}(x;\theta) = \tanh(\omega x) * NN(x;\theta)$. Wynik można zobaczyć na wykresie nr 13.

   Wykres, w którym pierwszą warstwę ukrytą zainicjalizowano cechami Fouriera poradził sobie dobrze choć mógł lepiej, widać to na wykresie nr. 16.

6. Bibliografia
   Wykład MOwNiT - prowadzony przez dr. Inż. K. Rycerz
   Prezentacje – dr. Inż. M. Kuta

7. Dodatkowe informacje
   Rozwiązanie zadania znajduje się w pliku ex1.ipynb