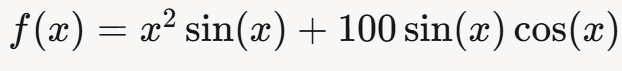
**Dokumentacja zadania 5: Aproksymacja funkcji za pomocą perceptronu wielowarstwowego**

**Artem Kukushkin, 317140, WSI**

**Cel zadania**

Zadanie polega na zaimplementowaniu perceptronu wielowarstwowego (MLP) do aproksymacji funkcji

 , na przedziale [−10,10]. Celem jest zbadanie:

1. Wpływu liczby neuronów i warstw na jakość aproksymacji (mierzona błędem średniokwadratowym MSE).
2. Różnic w jakości aproksymacji w zależności od metody optymalizacji: gradientowej (Adam) oraz ewolucyjnej (Differential Evolution, DE).

**Implementacja**

Implementacja została wykonana w języku Python z wykorzystaniem bibliotek:

**NumPy**: do operacji na danych i generowania danych treningowych/testowych.

**scikit-learn**: do implementacji MLP (MLPRegressor) z optymalizatorem Adam.

**SciPy**: do optymalizacji ewolucyjnej za pomocą metody Differential Evolution (differential\_evolution).

**Matplotlib**: do wizualizacji wyników.

**Dane**

Zbiór treningowy: 1000 równomiernie rozłożonych punktów x∈[−10,10].

Zbiór testowy: 200 równomiernie rozłożonych punktów x∈[−10,10].

Dane zostały wygenerowane za pomocą funkcji np.linspace, a wartości f(x).

**Architektura MLP**

**Aktywacja**: ReLU (Rectified Linear Unit), wybrana ze względu na jej prostotę i skuteczność w modelach nieliniowych.

**Konfiguracje sieci**:

(10,): 1 warstwa, 10 neuronów.

(20,): 1 warstwa, 20 neuronów.

(50,): 1 warstwa, 50 neuronów.

(20, 10): 2 warstwy, 20 i 10 neuronów.

(20, 20): 2 warstwy, 20 neuronów każda.

**Metody optymalizacji**

1. **Adam**:

Optymalizator gradientowy zaimplementowany w MLPRegressor.

Parametry: max\_iter=1000, random\_state=42 (dla powtarzalności).

Szybki i efektywny dla mniejszych sieci, ale może utknąć w minimach lokalnych.

1. **Differential Evolution (DE)**:

Metoda ewolucyjna z biblioteki SciPy.

Parametry: maxiter=50, popsize=15, seed=42.

Globalna metoda optymalizacji, wolniejsza, ale potencjalnie skuteczniejsza dla złożonych problemów.

**Metryka oceny**

**MSE (Mean Squared Error)**: Błąd średniokwadratowy na zbiorze testowym jako miara jakości aproksymacji.

**Czas treningu**: Mierzony dla obu metod optymalizacji, aby ocenić ich wydajność.

**Wyniki**

Tabela wyników

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Konfiguracja** | **MSE (Adam)** | **Czas (Adam)** | **MSE (DE)** | **Czas (DE)** |
| (10,) | 1.68e+03 | 1.31s | 1.69e+03 | 7.39s |
| (20,) | 1.68e+03 | 1.25s | 1.68e+03 | 12.29s |
| (50,) | 1.64e+03 | 4.71s | 1.70e+03 | 42.66s |
| (20, 10) | 1.14e+03 | 1.92s | 1.69e+03 | 83.03s |
| (20, 20) | 1.12e+03 | 2.06s | 1.69e+03 | 216.10s |

**Wizualizacja**

Wyniki aproksymacji przedstawiono na wykresach:

A graph of a graph of a graph

AI-generated content may be incorrect.

Każdy wykres pokazuje funkcję prawdziwą (czarna linia) oraz aproksymacje uzyskane za pomocą Adam (linia przerywana, niebieska) i DE (linia kropkowana, pomarańczowa).

Tytuł wykresu zawiera konfigurację sieci oraz wartości MSE dla obu metod.

**Analiza wyników**

1. **Wpływ liczby neuronów i warstw**:

**Jednowarstwowe sieci**:

(10,) i (20,): MSE dla Adam i DE jest podobne (~1.68e+03), co wskazuje na niewystarczającą zdolność modelu do uchwycenia złożoności funkcji.

(50,): Lepsze MSE dla Adam (1.64e+03), ale DE pogarsza się (1.70e+03), sugerując, że większa liczba neuronów nie zawsze pomaga DE bez odpowiedniego strojenia.

**Dwu-warstwowe sieci**:

(20, 10) i (20, 20): Znacznie lepsze MSE dla Adam (1.14e+03 i 1.12e+03), co pokazuje, że dodatkowe warstwy lepiej modelują nieliniowość funkcji.

DE pozostaje słabe (1.69e+03), co może wynikać z niewystarczającej liczby iteracji lub trudności w znalezieniu dobrego rozwiązania globalnego.

**Wnioski**: Zwiększenie liczby warstw (np. do 2) poprawia jakość aproksymacji dla Adam, ale liczba neuronów w jednej warstwie (np. 50) nie przynosi dużych korzyści.

1. **Porównanie metod optymalizacji**:

**Adam**: Szybszy (1.25–4.71s) i osiąga lepsze wyniki dla bardziej złożonych sieci (MSE 1.12e+03 dla (20, 20)). Gradientowe podejście dobrze radzi sobie z gładkimi funkcjami.

**DE**: Znacznie wolniejszy (7.39–216.10s), szczególnie dla większych sieci, ze względu na dużą liczbę wag (np. 450 dla (20, 20)). MSE jest konsekwentnie gorsze (~1.69e+03), co wskazuje na konieczność większej liczby iteracji lub lepszego strojenia.

**Wnioski**: Adam jest bardziej efektywny czasowo i skuteczny dla tej funkcji. DE wymaga optymalizacji parametrów (np. większej liczby iteracji lub równoległości) i jest bardziej odpowiedni dla problemów z wieloma minimami lokalnymi.

1. **Czas wykonania**:

Adam: Czas rośnie z liczbą neuronów i warstw (od 1.25s dla (20,) do 4.71s dla (50,)), ale pozostaje szybki.

DE: Czas rośnie dramatycznie z liczbą wag (np. 216.10s dla (20, 20)), co czyni metodę niepraktyczną dla większych sieci bez optymalizacji.

**Wnioski i rekomendacje**

1. **Jakość aproksymacji**:

Dwu-warstwowe sieci (np. (20, 20)) z Adam osiągają najlepsze wyniki (MSE 1.12e+03), co pokazuje, że głębsze architektury lepiej modelują złożoną funkcję.

DE nie osiąga dobrych wyników w obecnym ustawieniu i wymaga dalszego strojenia.

1. **Wydajność**:

Adam jest znacznie szybszy i bardziej odpowiedni do tego zadania.

DE jest zbyt wolne dla większych sieci, ale można je przyspieszyć przez:

Zmniejszenie maxiter i popsize.

Włączenie równoległości (workers=-1).

Zmniejszenie liczby punktów treningowych.