AKADEMIA GÓRNICZO-HUTNICZA

Dokumentacja do projektu

Biblioteka do modelowania prostych sieci neuronowych

z przedmiotu

Języki Programowania Obiektowego

Elektronika i Telekomunikacja 3. rok

Kacper Łucki

Piątek 13:15

prowadzący: mgr inż. Jakub Zimnol

10.01.2025 r.

1. Wprowadzenie

Projekt przedstawia bibliotekę do implementacji prostych sieci neuronowych, zaprojektowaną do klasyfikacji danych na podstawie cech wejściowych. Biblioteka demonstruje podstawowe koncepcje uczenia maszynowego w języku C++ z wykorzystaniem klas, takich jak Neuron, Layer, oraz NeuralNetwork. Dodatkowo projekt uwzględnia normalizację danych wejściowych oraz obliczenia propagacji w przód i wstecz.

2. Zastosowania projektu

Biblioteka może być wykorzystywana do nauki podstaw sieci neuronowych i algorytmów uczenia maszynowego. Umożliwia szybkie prototypowanie modeli klasyfikacyjnych i prezentację ich działania na rzeczywistych danych. Dzięki wbudowanym funkcjom normalizacji i propagacji pozwala na analizę oraz przetwarzanie danych w małej skali. To narzędzie do zrozumienia i eksperymentowania z podstawowymi koncepcjami sztucznej inteligencji.

3. Zaimplementowane klasy

Klasa Neuron

Reprezentuje pojedynczą jednostkę obliczeniową w sieci neuronowej.

Właściwości:

Wagi połączeń z poprzednią warstwą.

Bias (przesunięcie).

Wartość aktywacji i gradient dla propagacji wstecznej.

Metody:

Funkcje aktywacji (tanh) i jej pochodna.

Ustawianie oraz pobieranie wag, biasów, wartości aktywacji i gradientów.

Klasa Layer

Bazowa klasa dla wszystkich warstw sieci neuronowej. Zawiera interfejs dla propagacji w przód i wstecz.

Klasy dziedziczące:

InputLayer: Warstwa wejściowa odpowiedzialna za przekazywanie danych wejściowych do sieci.

HiddenLayer: Warstwa ukryta wykonująca transformacje na danych wejściowych.

OutputLayer: Warstwa wyjściowa generująca przewidywania oraz obliczająca błąd.

Klasa NeuralNetwork

Zarządza całą siecią neuronową, łącząc warstwy w kompletną strukturę.

Funkcjonalności:

Propagacja w przód: Przetwarzanie danych od warstwy wejściowej do wyjściowej.

Propagacja wstecz: Obliczanie gradientów i aktualizacja wag w celu minimalizacji błędu.

Obliczanie błędów i dokładności.

Moduł utils

Funkcje pomocnicze do przetwarzania danych wejściowych.

Funkcjonalności:

Normalizacja danych wejściowych do zakresu [0, 1].

Ładowanie danych do macierzy.

4. Kompilacja i uruchomienie

Linux

Sklonuj repozytorium:

git clone https://github.com/example/NeuralNetworkJPO.git

cd NeuralNetworkJPO

Zbuduj projekt przy użyciu CMake:

mkdir build

cd build

cmake

make

./NeuralNetworkJPO

Należy upewnić się, że ma się zainstalowany **CMake** (wersja co najmniej **3.10**), jeżeli CMake nie jest zainstalowany:

sudo apt-get update

sudo apt-get install cmake

Należy upewnić się, że ma się zainstalowaną **bibliotkę Eigen3**, jeżeli Eigen3 nie jest zainstalowany:

sudo apt-get update

sudo apt-get install libeigen3-dev

Windows

Zainstaluj kompilator zgodny z C++17 (np. Visual Studio).

Zainstaluj CMake (co najmniej 3.10) ze strony cmake.org.

Pobierz wybraną wersję biblioteki Eigen3 ze strony https://eigen.tuxfamily.org

Rozpakuj w wybranej lokalizacji, najlepiej w C:\ lub C:\Program Files, aby moduły znajdowania CMake mogły łatwo ją wykryć (pamiętaj, aby wyodrębnić wewnętrzny folder i zmienić jego nazwę na Eigen3 lub Eigen).

Wykonaj te same kroki, co w sekcji dotyczącej Linuxa, używając terminala, takiego jak PowerShell, lub GUI CMake

5. Przykład działania

W pliku main.cpp zostały zaimplementowane trzy testy ilustrujące działanie biblioteki:

- **Test XOR**
  Sieć neuronowa została użyta do rozwiązania problemu logicznego XOR, demonstrując zdolność sieci do nauki nieliniowych zależności.

- **Przewidywanie wartości funkcji sinus**
  Model został przeszkolony do aproksymacji funkcji sinus w wybranym zakresie, co pokazuje możliwość modelowania funkcji ciągłych.
- **Analiza danych Iris Dataset**
  Sieć przeprowadza klasyfikację próbek danych z zestawu iris_dataset (pliku iris.csv), co ilustruje praktyczne zastosowanie w analizie danych i rozpoznawaniu wzorców.

6. Test wycieków pamięci

Program wykorzystuje dynamiczne alokowanie pamięci, dlatego celu przetestowania programy pod kątem wycieków pamięci wykorzystano narzędzie Valgrind, a następnie uruchomiono program komendą:

valgrind –leak-check=full ./NeutalNetworkJPO

```
==7751== HEAP SUMMARY:
==7751==     in use at exit: 0 bytes in 0 blocks
==7751==   total heap usage: 29,277,573 allocs, 29,277,573 frees, 906,453,270 bytes allocated
==7751==
==7751== All heap blocks were freed -- no leaks are possible
==7751==
==7751== For lists of detected and suppressed errors, rerun with: -s
==7751== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Zdjęcie 1 Wyniki testu