Karol Ziarek 318749
Jakub Kieruczenko 318669

POP 23Z

Zadanie 4.

Opracuj i przeprowadź eksperyment, w którym porównasz dowolny wariant ewolucji różnicowej z dowolnym wariantem strategii ewolucyjnej w ramach zadania uczenia sieci neuronowej. Ponadto porównaj działanie metod ewolucyjnych z wybraną metodą gradientową.

1. Opis problemu i jego sposobu rozwiązania

Celem projektu jest zbadanie i porównanie skuteczności klasycznego podejścia gradientowego z alternatywnymi metodami: ewolucją różnicową oraz strategią ewolucyjną w uczeniu sztucznej sieci neuronowej.

Uczenie sieci neuronowej opiera się na dostosowywaniu wag w kolejnych warstwach sieci neuronowej w celu minimalizacji funkcji błędu, co odbywa się na podstawie danych treningowych. W klasycznym podejściu, gdzie wykorzystywana jest metoda gradientu prostego uczenie opiera się na dwu głównych krokach:

- propagacja sygnału w przód - dane przechodzą przez kolejne warstwy neuronów i na podstawie wag i funkcji aktywacji na wyjściu sieci generowany jest wynik,
- propagacja błędu wstecz - obliczony błąd między wynikami (najczęściej MSE) jest propagowany wstecz przez sieć, podczas której obliczane są gradienty funkcji błędu względem wag sieci w celu aktualizacji wag, tak by zminimalizować funkcję błędu.

Takie rozwiązanie wiąże się jednak z potencjalnymi problemami, które możemy wyeliminować implementując do celów uczenia sieci neuronowych algorytmy ewolucyjne. Takie potencjalne problemy to między innymi:

- Zatrzymywanie się w ekstremach lokalnych:
    - Algorytmy ewolucyjne mogą lepiej radzić sobie z poszukiwaniem globalnych minimów funkcji błędu, zwłaszcza w trudnych przestrzeniach rozwiązań (np. zaszumionych, wielomodalnych) co czasem może być problematyczne dla metod gradientowych.
- Odporność na znikający gradient:
    - W przeciwności do metod gradientowych, algorytmy ewolucyjne nie wykorzystują gradientów, więc nie są podatne na zjawisko znikającego gradientu które może wystąpić w głębokich sieciach neuronowych, gdzie gradient może maleć lub rosnąć eksponencjalnie.

Należy jednak pamiętać, że implementacja algorytmów ewolucyjnych do tego typu zadania również wiąże się z pewnymi problemami. Problemy te występują głównie w sferze wydajności i czasu uczenia modeli, co jest spowodowane większą zasobochłonnością obliczeniową w stosunku do metod gradientowych i jest szczególnie widoczne w przypadku

dużych zbiorów danych i skomplikowanych modeli. Oprócz tego problematyczne może być odpowiednie zakodowania wag sieci (osobników algorytmu ewolucyjnego) w postaci genotypu.

Wybrane metody optymalizacji:

1) Ewolucja różnicowa: **SHADE - Success-History based Adaptive Differential Evolution**
   Wykorzystujemy wersję algorytmu ewolucji różnicowej, która opiera się na dynamicznym dostosowywaniu parametrów F i c_r z wykorzystaniem historii sukcesów poprzednich iteracji. W przeciwieństwie do standardowej ewolucji różnicowej pozwala na osiągnięcie dowolnego punktu.

2) Strategia ewolucyjna: **CMA-ES - Covariance Matrix Adaptation Evolution Strategy**
   Adaptacja rozkładu prawdopodobieństwa generacji punktów, opisanego przez wartość oczekiwaną i macierz kowariancji oraz parametr długości kroku. Najważniejsze mechanizmy algorytmu obejmują adaptację macierzy kowariancji i adaptację długości kroku.

3) Metoda **gradientu prostego z propagacją wsteczną**:
   Implementowany, jak opisano wyżej, na zasadzie feedforwardingu i backpropagation. W wersji prostej do obliczenia gradientu funkcji błędu wykorzystywany jest cały zbiór treningowy.

Każdy z algorytmów zostanie zastosowany dla kilku architektur sieci neuronowych o różnej złożoności zaimplementowanych przy pomocy PyTorch.

Potencjalne sieci do testowania:

Prosta Sieć Neuronowa (FNN):

- Warstwa wejściowa: 784 (28x28 pikseli obrazu MNIST)
- Warstwa ukryta: 128 neuronów (lub inna liczba)
- Funkcja aktywacji: ReLU
- Warstwa wyjściowa: 10 neuronów (klasy 0-9)
- Funkcja aktywacji: Softmax

Konwolucyjna Sieć Neuronowa (CNN):

- Warstwa wejściowa: 28x28x1 (1 kanał obrazu)
- Konwolucyjna warstwa: 32 filtry, rozmiar jądra 3x3, funkcja aktywacji ReLU
- Max Pooling: Rozmiar 2x2
- Konwolucyjna warstwa: 64 filtry, rozmiar jądra 3x3, funkcja aktywacji ReLU
- Max Pooling: Rozmiar 2x2
- Warstwa ukryta: 128 neuronów, funkcja aktywacji ReLU
- Warstwa wyjściowa: 10 neuronów, funkcja aktywacji Softmax

Recurrent Neural Network (RNN):

- Warstwa wejściowa: 28x28 (płaski obraz MNIST)
- Warstwa rekurencyjna: LSTM lub GRU z odpowiednią liczbą jednostek
- Warstwa ukryta: 128 neuronów, funkcja aktywacji ReLU
- Warstwa wyjściowa: 10 neuronów, funkcja aktywacji Softmax

Sieć z Warstwą Rekurencyjną i Konwolucyjną (CRNN):

- Warstwa wejściowa: 28x28x1 (obraz MNIST)
- Konwolucyjna warstwa: 32 filtry, rozmiar jądra 3x3, funkcja aktywacji ReLU
- Max Pooling: Rozmiar 2x2
- Warstwa rekurencyjna: LSTM lub GRU
- Warstwa ukryta: 128 neuronów, funkcja aktywacji ReLU
- Warstwa wyjściowa: 10 neuronów, funkcja aktywacji Softmax

2. Planowane eksperymenty numeryczne

Celem przeprowadzanych eksperymentów jest porównanie działania wytrenowanych modeli sieci neuronowych z wykorzystaniem różnych metod uczenia sieci neuronowych. Analizować będziemy jakość działania sieci w kontekście zadania klasyfikacji, główne miary które będziemy wyznaczać dla modeli to: dokładność (accuracy), macierz pomyłek (confusion matrix), precyzja (precision) i czułość (recall).

Eksperymenty będą przeprowadzone w 3 krokach:

1. *Ewaluacja Początkowa:* Ocena jakości wytrenowanych modeli sieci neuronowych dla domyślnych hiperparametrów.
2. *Optymalizacja Parametrów:* Dobór optymalnych hiperparametrów dla każdej metody.
3. *Ewaluacja końcowa:* Porównanie wyników i wyciągnięcie wniosków.

Do przeprowadzeniu planujemy wykorzystać zbiór mnist jako standardowy zbiór w kontekście problemu klasyfikacji, ale dopuszczamy możliwość sprawdzenia działania metod dla innych zbiorów.

3. Technologie, w których realizowany będzie projekt.

Projekt zdecydowaliśmy się realizować w języku **Python**, głównie ze względu na dużą dostępność narzędzi i bibliotek ułatwiających realizację zagadnień z domeny Machine Learning. Dokładnie będziemy korzystać z bibliotek:

- *Jupyter Notebook* - do przeprowadzania eksperymentów i analizy otrzymanych wyników w jednym pliku
- *NumPy* - do operacji numerycznych i manipulacji danymi.
- *pandas* - zarządzanie, manipulacja zbiorami danych
- *PyTorch* - do implementacji sieci neuronowych
- *DEAP* - do implementacji algorytmów ewolucyjnych