Temat: Implementacja w pełni połączonej sieci neuronowej do klasyfikacji wielomodalnej

Cel: Celem tej pracy jest stworzenie w pełni połączonej sieci neuronowej (fully connected neural network) w PyTorch, która będzie rozpoznawać różne klasy na podstawie cech z podanego zbioru danych. Studenci nauczą się, jak przygotować dane do treningu, skonfigurować sieć, dobrać hiperparametry, a także przeanalizować i zwizualizować wyniki modelu.

Opis zadania:

1. Dane wejściowe:
    - Wybierz zbiór danych z repozytorium UCI Machine Learning lub Kaggle (lub dostarcz przygotowany zbiór).
    - Przykładowe zbiory danych:
        - Iris: Zbiór z czterema cechami opisującymi kwiaty irysów, z trzema możliwymi klasami do klasyfikacji.
        - Wine: Zbiór opisujący właściwości chemiczne różnych typów wina, z trzema klasami.
        - Digits: Zbiór danych cyfrowych, który można potraktować jako zbiór wielomodalny z 10 klasami.
2. Wymagania sieci neuronowej:
    - Sieć powinna być w pełni połączona, z przynajmniej jedną warstwą ukrytą (hidden layer).
    - Każda warstwa powinna zawierać odpowiednią funkcję aktywacji (np. ReLU).
    - Warstwa wyjściowa powinna mieć tyle neuronów, ile jest klas w zadaniu i używać funkcji aktywacji Softmax do przewidywania prawdopodobieństw klas.
3. Podział danych i preprocessing:
    - Podziel zbiór danych na zestawy treningowy i testowy (np. 80% dla treningu, 20% dla testu).
    - Standaryzuj lub normalizuj dane, aby przyspieszyć trening sieci.
4. Implementacja sieci neuronowej:
    - Zaimplementuj klasę sieci neuronowej w PyTorch, uwzględniając strukturę wielowarstwową.
    - Zdefiniuj funkcję kosztu (np. Cross-Entropy Loss) i optymalizator (np. Adam lub SGD).
    - Przeprowadź proces treningu i walidacji modelu oraz śledź zmianę funkcji kosztu i dokładności.
5. Trening i testowanie:
    - Przeprowadź trening modelu na danych treningowych przez określoną liczbę epok.

- Po zakończeniu treningu, oceń dokładność modelu na zbiorze testowym.

6. Eksperymentowanie z hiperparametrami:
    - Przeprowadź eksperymenty zmieniając liczbę neuronów w warstwach, liczbę warstw, funkcje aktywacji i inne hiperparametry.
    - Zapisz wyniki dla każdej konfiguracji (np. w tabeli) i spróbuj uzyskać model o najwyższej dokładności na zbiorze testowym.
7. Wizualizacja i analiza wyników:
    - Narysuj wykresy pokazujące zmiany funkcji kosztu oraz dokładności na danych treningowych i walidacyjnych w trakcie trenowania.

Raport:

1. Kod źródłowy: Skrypt w języku Python (np. w formacie .py lub notebook .ipynb) zawierający pełną implementację sieci i eksperymentów.
2. Raport (PDF lub .ipynb): Krótki raport (1-2 strony), który zawiera:
    - Opis eksperymentów, użytych hiperparametrów oraz architektury sieci.
    - Wyniki dokładności na zbiorze testowym oraz omówienie wykresów.
    - Wnioski z przeprowadzonych eksperymentów: który model działał najlepiej i dlaczego.

Wytyczne dotyczące oceny:

- Poprawność implementacji (40%): Poprawność kodu, struktura sieci, konfiguracja treningu i walidacji.
- Eksperymenty i analiza (30%): Analiza doboru hiperparametrów na dokładność modelu i próba optymalizacji wyników.
- Raport (20%): Zwięzły, czytelny opis wyników i wniosków.
- Wizualizacje (10%): Wykresy i analiza błędów.

Dodatkowe wskazówki:

- Pamiętaj o regularnym zapisywaniu modelu i wyników, aby móc powrócić do najlepszej wersji.
- Korzystaj z GPU, jeśli jest dostępne, aby przyspieszyć proces trenowania (opcjonalnie).

Powodzenia!