WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI I ELEKTROTECHNIKI

Algorytmy Genetyczne i Sztuczne Sieci Neuronowe - laboratorium

SPRAWOZDANIE Z LABORATORIUM

Temat sprawozdania:

Algorytm Propagacji Wstecznej

Wykonał: Sławomir Piotrzkowski

Kierunek: Informatyka Stosowana - studia niestacjonarne - II stopień

Semestr: I

Rok Akademicki: 2024/2025

1. Cel Laboratorium.

Celem laboratorium jest zapoznanie się z zasadami działania algorytmu propagacji wstecznej (ang. backpropagation) stosowanego w procesie uczenia wielowarstwowych sieci neuronowych. W szczególności, eksperymenty mają na celu zrozumienie mechanizmu wyznaczania błędów i modyfikacji wag, a także przeprowadzenie praktycznej implementacji algorytmu w przykładowym problemie klasyfikacji danych.

2. Wstęp teoretyczny.

Algorytm propagacji wstecznej jest jednym z najczęściej stosowanych algorytmów uczenia sieci neuronowych. Umożliwia on modyfikację wag w wielowarstwowej sieci neuronowej na podstawie różnicy pomiędzy wartościami wyjściowymi uzyskanymi przez sieć a oczekiwanymi wartościami z zestawu uczącego.

2.1. Kluczowe elementy algorytmu:

- **Sygnał wejściowy i propagacja w przód:**

  Dla danej pary danych uczących , gdzie to wektor wejściowy, a to oczekiwana wartość wyjściowa, obliczane są sygnały wyjściowe w kolejnych warstwach sieci.

  Wyjście neuronu obliczane jest jako:

$$e = \sum_{i=1}^{n} w_i x_i + b$$

  gdzie:
  e - sygnał wejściowy neuronu
  w_i - wagi połączeń,
  x_i - wartości wejściowe,
  b - bias (przesunięcie).

  Sygnał wyjściowy neuronu to:

$$y = f(e)$$

  gdzie $f(e)$ to funkcja aktywacji, np. funkcja sigmoidalna.

- **Obliczenie sygnałów błędów:**

  Błąd na wyjściu każdego neuronu wyjściowego określany jest jako różnica pomiędzy wartością rzeczywistą a otrzymaną:

$$\delta = z - y$$

  gdzie:
  y - wartość wyjściowa neuronu
  z - oczekiwana wartość wyjściowa

- Propagacja wsteczna:

  Błędy propagowane są wstecz przez sieć, z uwzględnieniem pochodnej funkcji aktywacji, co umożliwia obliczenie błędów dla warstw ukrytych.

  Dla warstwy ukrytej błąd wyznaczany jest jako:

$$\delta_j = \sum_k \delta_j w_{jk}$$

  gdzie:
  δ_j - błąd neuronu wyjściowego
  w_{jk} - wagi połączeń między warstwą ukrytą a wyjściową
  y_j - wartość wyjściowa neuronu w warstwie ukrytej

- Aktualizacja wag:

  Wagi są modyfikowane zgodnie z zasadą:

$$w_{ij} = w_{ij} + \eta \cdot \delta_j \cdot x_i$$

  gdzie:
  w_{ij} – waga między wejściem x_i a neuronem j
  η – współczynnik uczenia (zazwyczaj malejący w czasie),
  x_j – j-ta wartość wektora wejściowego,
  δ_j – błąd neuronu j.

Funkcja aktywacji:

Najczęściej stosowaną funkcją aktywacji w procesie uczenia jest funkcja sigmoidalna:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Pochodna funkcji sigmoidalnej wynosi:

$$f'(x) = f(x)f(1 - f(x))$$

3. Implementacja.

Poniżej przedstawiono przykładową implementację algorytmu propagacji wstecznej dla trójwarstwowej sieci neuronowej. Kod w Pythonie:

```python
# Imports
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Dane uczące
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Inicjalizacja wag
np.random.seed(42)
weights_input_hidden = np.random.uniform(-1, 1, (2, 2))
weights_hidden_output = np.random.uniform(-1, 1, (2, 1))

learning_rate = 0.1

# Proces uczenia
cycles = 100000
for epoch in range(cycles):
    # Propagacja w przód
    hidden_layer_input = np.dot(X, weights_input_hidden)
    hidden_layer_output = sigmoid(hidden_layer_input)

    final_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
    final_layer_output = sigmoid(final_layer_input)

    # Obliczenie błędu
    error = y - final_layer_output

    # Propagacja wsteczna
    d_output = error * sigmoid_derivative(final_layer_output)
    error_hidden_layer = d_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    # Aktualizacja wag
    weights_hidden_output += hidden_layer_output.T.dot(d_output) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate

print(f"Wyjście sieci po nauce dla {cycles} epok:\n", final_layer_output)
```

Opis opracowanego programu:

Celem programu była implementacja trójwarstwowej sieci neuronowej wykorzystującej algorytm propagacji wstecznej, aby nauczyć ją operacji logicznej XOR. Sieć składa się z dwóch neuronów w warstwie wejściowej, dwóch w warstwie ukrytej oraz jednego neuronu w warstwie wyjściowej. Podczas procesu uczenia wykorzystano zestaw danych uczących zawierający cztery możliwe kombinacje wejść i oczekiwanych wyjść dla bramki XOR.

Algorytm propagacji wstecznej pozwolił na iteracyjne dostosowywanie wag połączeń między neuronami, tak aby minimalizować błąd pomiędzy rzeczywistymi a oczekiwanymi wynikami sieci. Po zakończeniu procesu uczenia sieć prawidłowo klasyfikuje wszystkie kombinacje wejściowe dla operacji XOR, co potwierdza skuteczność algorytmu oraz poprawność implementacji.

4. Wyniki.

Program wykonano w dwóch wariantach. Dla 10 000 i 100 000 epok.

Dla 10 000 epok otrzymano następujące wyniki:

Wejście	Oczekiwane wyjście	Uzyskane wyjście
[0, 0]	0	~0.26
[0, 1]	1	~0.68
[1, 0]	1	~0.69
[1, 1]	0	~0.41

Dla 100 000 otrzymano następujące wyniki

Wejście	Oczekiwane wyjście	Uzyskane wyjście
[0, 0]	0	~0.03
[0, 1]	1	~0.93
[1, 0]	1	~0.93
[1, 1]	0	~0.09

Wyniki dla 10 000 epok.

W przypadku problemu XOR i wyników sieci neuronowej, warto rozważyć pewien margines błędu. Jeśli oczekujemy wyjścia 1 i sieć generuje wynik 0.68, można uznać go za zadowalający w kontekście działania algorytmu, szczególnie przy wykorzystaniu funkcji aktywacji sigmoidalnej. Dzieje się tak, ponieważ funkcja sigmoidalna nie generuje dokładnych wartości 0 lub 1, lecz wartości z przedziału (0,1), które interpretujemy jako klasy.

Decyzja o akceptacji wyniku

- **Wynik powyżej 0.5:** W kontekście klasyfikacji binarnej, wynik 0.68 można zaokrąglić w górę i uznać za prawidłowy, ponieważ przekracza próg 0.5, często przyjmowany jako granica między klasami.
- **Wynik zbliżony do 1:** Im bardziej wynik zbliża się do 1, tym lepiej sieć odwzorowuje oczekiwaną wartość.

Warto także ocenić cel aplikacji – w niektórych zastosowaniach wynik 0.68 może być wystarczający, podczas gdy w innych konieczne będzie osiągnięcie wyższego poziomu dokładności.

Po zwiększeniu ilości epok do 100 000 wartości wyjściowe są bardziej zbliżone do oczekiwanych.
Wyniki te wskazują, że sieć skutecznie nauczyła się operacji XOR, co potwierdza poprawność implementacji algorytmu propagacji wstecznej.

5. Podsumowanie.

W ramach laboratorium:

- Przeanalizowano teoretyczne podstawy algorytmu propagacji wstecznej.
- Zaimplementowano trójwarstwową sieć neuronową w języku Python.
- Zweryfikowano skuteczność algorytmu na przykładowym problemie logicznym XOR.

Algorytm propagacji wstecznej, mimo swojej prostoty, stanowi fundament dla wielu zaawansowanych metod uczenia głębokiego, wykorzystywanych w współczesnych sieciach neuronowych.