SPRAWOZDANIE

Zajęcia: Matematyka Konkretna

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 5 Data 26.04.2025 Temat: „Metoda gradientu prostego. Wsteczna propagacja błędu" Wariant 10	Szymon Nycz Informatyka II stopień, niestacjonarne, 2 semestr, gr.1a TTO

1. Polecenie:

Link do repozytorium: https://github.com/Maciek332/Semestr_3_Nycz/tree/main/MK

2. Wstęp teoretyczny

Metoda gradientu prostego jest jedną z podstawowych technik optymalizacji funkcji wielu zmiennych. Polega na iteracyjnym przeszukiwaniu przestrzeni rozwiązań poprzez przesuwanie się w kierunku przeciwnym do gradientu funkcji, co pozwala na znajdowanie jej minimum.

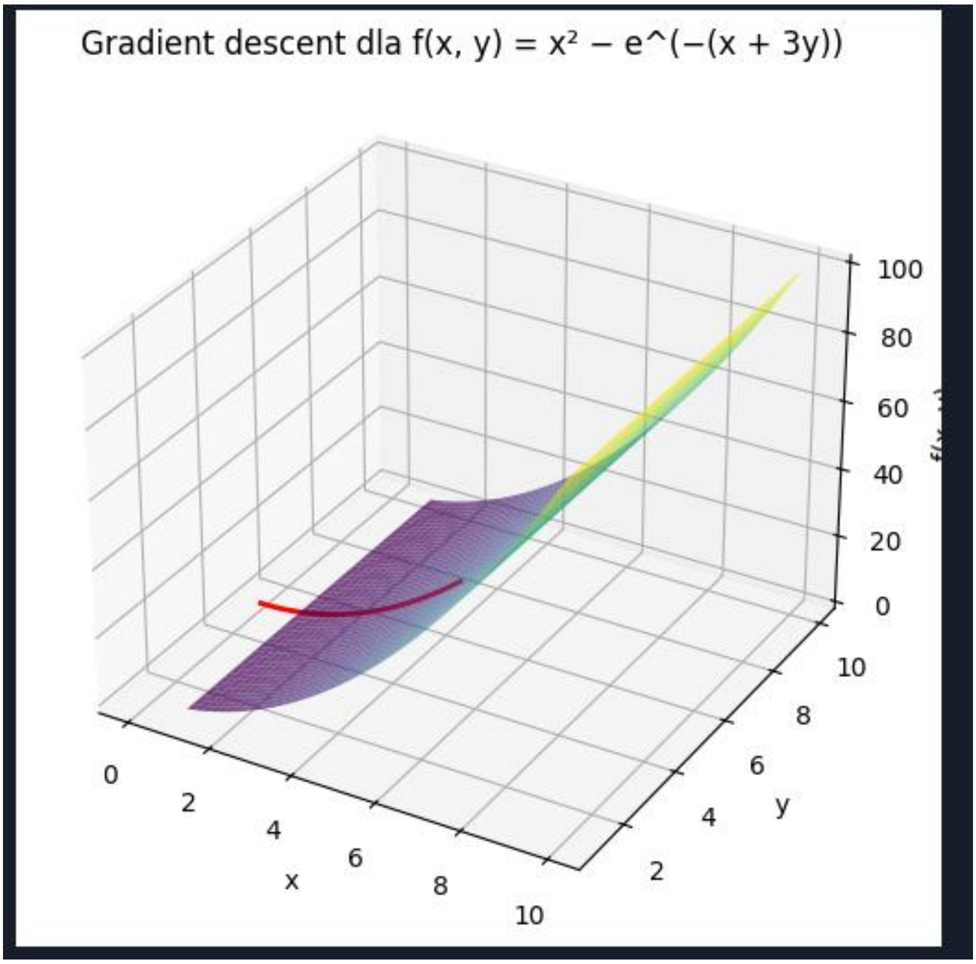
W pierwszej części zadania celem było zastosowanie metody gradientu prostego do minimalizacji funkcji dwóch zmiennych oraz przedstawienie wyników w formie wizualizacji 3D. Pozwoliło to na intuicyjne zrozumienie, jak gradient kieruje ruchem w przestrzeni funkcji w stronę jej minimum.

W drugiej części zadania zastosowano obliczenia gradientów w kontekście sieci neuronowych. Sieć była zdefiniowana według podanej architektury, a zadaniem było policzenie gradientów z wykorzystaniem biblioteki numpy. Proces ten odpowiada etapowi wstecznej propagacji błędu (backpropagation), który jest kluczowy w uczeniu modeli neuronowych.

3. Opis programu opracowanego

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return x**2 - np.exp(-(x + 3*y))

def grad_f(x, y):
    dfdx = 2*x + np.exp(-(x + 3*y))
    dfdy = 3 * np.exp(-(x + 3*y))
    return np.array([dfdx, dfdy])

# Gradient Descent
lr = 0.05
x, y = 5.0, 5.0
history = [(x, y)]

for _ in range(100):
    grad = grad_f(x, y)
    x -= lr * grad[0]
    y -= lr * grad[1]
    history.append((x, y))

history = np.array(history)

# Wykres 3D
x_vals = np.linspace(1, 10, 100)
y_vals = np.linspace(1, 10, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = f(X, Y)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7)
ax.plot(history[:, 0], history[:, 1], f(history[:, 0], history[:, 1]), 'r-', lw=2)
ax.set_title("Gradient descent dla f(x, y) = x² □ e^(□(x + 3y))")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("f(x, y)")
plt.show()
```

✓ 0.4s




```python
import numpy as np

def relu(z): return np.maximum(0, z)
def relu_deriv(z): return z > 0

def sigmoid(z): return 1 / (1 + np.exp(-z))
def sigmoid_deriv(z): return sigmoid(z) * (1 - sigmoid(z))

def tanh(z): return np.tanh(z)
def tanh_deriv(z): return 1 - np.tanh(z)**2

def elu(z, alpha=1.0): return np.where(z > 0, z, alpha * (np.exp(z) - 1))
def elu_deriv(z, alpha=1.0): return np.where(z > 0, 1, alpha * np.exp(z))

activation_map = {
    "relu": (relu, relu_deriv),
    "sigmoid": (sigmoid, sigmoid_deriv),
    "tanh": (tanh, tanh_deriv),
    "elu": (elu, elu_deriv),
}

X = np.array([[0.5], [0.2]])
Y_true = np.array([[1]])

def forward(X, architecture):
    memory = []
    A = X
    for layer in architecture:
        W = np.random.randn(layer["output_dim"], layer["input_dim"])
        b = np.random.randn(layer["output_dim"], 1)
        Z = np.dot(W, A) + b
        A = activation_map[layer["activation"]][0](Z)
        memory.append((A, Z, W, b))
    return A, memory
```

```python
def backward(X, Y, memory, architecture):
    grads = []
    dA = 2 * (memory[-1][0] - Y)
    for i in reversed(range(len(architecture))):
        A, Z, W, b = memory[i]
        activation_deriv = activation_map[architecture[i]["activation"]][1]
        dZ = dA * activation_deriv(Z)
        dW = np.dot(dZ, X.T if i == 0 else memory[i-1][0].T)
        db = np.sum(dZ, axis=1, keepdims=True)
        dA = np.dot(W.T, dZ)
        grads.append((dW, db))
    return grads

# 🔁 Uruchomienie dla wariantu 10
nn_architecture = [
    {"input_dim": 2, "output_dim": 2, "activation": "elu"},
    {"input_dim": 2, "output_dim": 1, "activation": "elu"}
]


output, memory = forward(X, nn_architecture)
grads = backward(X, Y_true, memory, nn_architecture)

# Wyświetlenie
for i, (dW, db) in enumerate(grads[::-1]):
    print(f"Warstwa {i+1}:")
    print("dW:\n", dW)
    print("db:\n", db)
    print()
```

```
Warstwa 1:
dW:
 [[ 0.42279672  0.16911869]
 [-0.11723849 -0.0468954 ]]
db:
 [[ 0.84559343]
 [-0.23447698]]

Warstwa 2:
dW:
 [[-0.18627072  0.4133734 ]]
db:
 [[-0.62820268]]
```

4. Wnioski

Przeprowadzone zadania pozwoliły na praktyczne zrozumienie działania metody gradientu prostego i jego roli w optymalizacji funkcji. W przypadku funkcji o prostszej strukturze gradient prowadził szybko i skutecznie do minimum lokalnego. Dla bardziej złożonych funkcji proces był wolniejszy lub prowadził do minimum lokalnego zależnego od punktu startowego.

W kontekście sieci neuronowych obliczenia gradientów pokazały, jak ważne jest poprawne propagowanie informacji wstecznie w celu aktualizacji wag i minimalizacji funkcji kosztu. Ćwiczenie to uwidoczniło zależność skuteczności uczenia od poprawnego liczenia pochodnych oraz doboru odpowiednich funkcji aktywacji.

Dzięki realizacji zadania zdobyto cenną praktykę w implementacji podstawowych algorytmów optymalizacyjnych oraz w mechanizmach działania sztucznych sieci neuronowych.