SPRAWOZDANIE

Zajęcia: Nauka o danych II

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 2 Data 29.03.2025 Temat: „Praktyczne ćwiczenia z algorytmami optymalizacji dla uczenia maszynowego” Wariant 1	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/NoD%20II

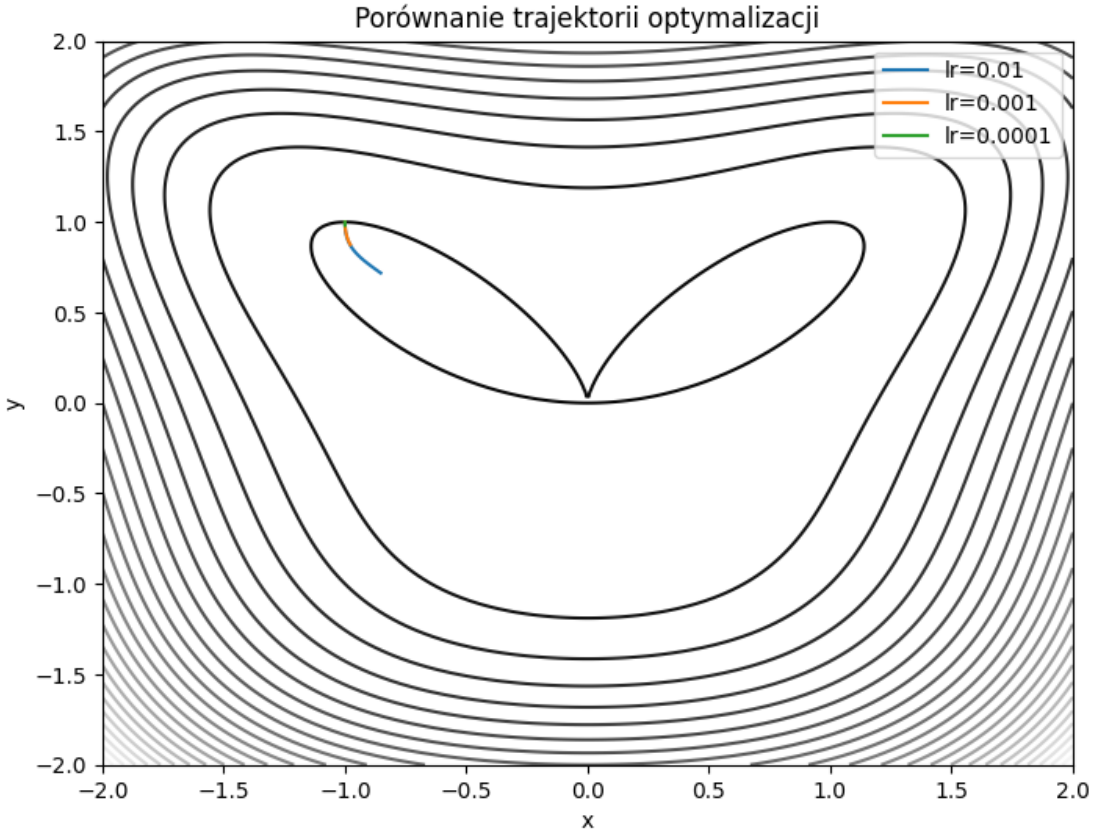
1. **Wariant 1**
   - Zbadaj wpływ współczynnika uczenia: $\eta = 0.01, 0.001, 0.0001$.
   - Przetestuj funkcję celu: $f(x, y) = x^4 + y^4 - 2x^2y$.
   - Zaimplementuj sieć MLP dla klasyfikacji cyfr MNIST i monitoruj trening w TensorBoard.

W uczeniu maszynowym optymalizacja polega na minimalizacji funkcji kosztu L(θ), gdzie θ oznacza parametry modelu. Kluczową rolę odgrywa gradient ∇L, wskazujący najszybszy kierunek spadku wartości funkcji. Klasyczny Gradient Descent aktualizuje parametry po pełnym przejściu przez zbiór danych, zapewniając stabilność, ale kosztem wolniejszej konwergencji. Stochastic Gradient Descent przyspiesza ten proces, obliczając gradient na losowych przykładach lub małych partiach, co zwiększa wariancję aktualizacji. Momentum wprowadza efekt „bezwładności”, wygładzając trajektorię optymalizacji i redukując oscylacje. RMSProp dostosowuje krok uczenia poprzez normalizację gradientów względem ich aktualnej wariancji. Wybór algorytmu zależy od specyfiki danych, wymaganej szybkości konwergencji i tolerancji na niestabilność procesu uczenia.

2. Opis programu opracowanego

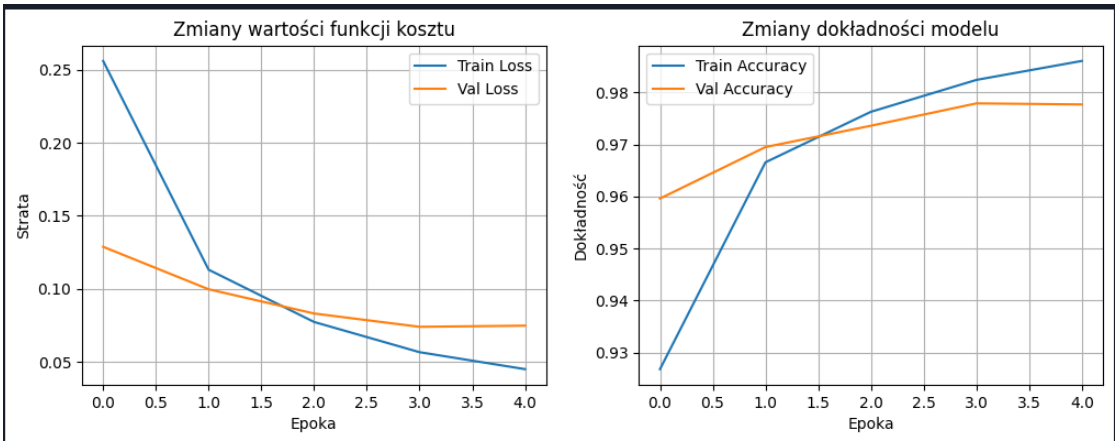
```python
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return x**4 + y**4 - 2*x**2*y

def grad_f(x, y):
    dx = 4*x**3 - 4*x*y
    dy = 4*y**3 - 2*x**2
    return np.array([dx, dy])

def gradient_descent(lr, steps=100, init=(-1, 1)):
    x, y = init
    trajectory = [(x, y)]

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

    return np.array(trajectory)

learning_rates = [0.01, 0.001, 0.0001]
plt.figure(figsize=(8, 6))
x_vals = np.linspace(-2, 2, 400)
y_vals = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = f(X, Y)
plt.contour(X, Y, Z, levels=30, cmap="gray")

for lr in learning_rates:
    path = gradient_descent(lr)
    plt.plot(path[:, 0], path[:, 1], label=f"lr={lr}")

plt.legend()
plt.xlabel("x")
plt.ylabel("y")
plt.title("Porównanie trajektorii optymalizacji")
plt.show()
```

✓ 0.8s


Porównanie trajektorii optymalizacji
lr=0.01
lr=0.001
lr=0.0001
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

```python
import tensorflow as tf
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train.reshape(-1, 28*28) / 255.0, x_test.reshape(-1, 28*28) / 255.0

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation="relu", input_shape=(784,)),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test), verbose=0)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.xlabel("Epoka")
plt.ylabel("Strata")
plt.title("Zmiany wartości funkcji kosztu")
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Val Accuracy")
plt.xlabel("Epoka")
plt.ylabel("Dokładność")
plt.title("Zmiany dokładności modelu")
plt.legend()
plt.grid()

plt.show()
```

✓ 6.7s



3. Wnioski

Analiza metod GD, SGD, Momentum, RMSProp i Adam pokazuje, że techniki adaptacyjne, zwłaszcza Adam, zapewniają najszybszą i najbardziej stabilną konwergencję w szerokim zakresie współczynników uczenia, czyniąc je praktycznym rozwiązaniem dla większości problemów. Momentum efektywnie przyspiesza dojście do optimum na gładkich funkcjach, ale wymaga precyzyjnego doboru η, podczas gdy klasyczny SGD, choć intuicyjny, jest podatny na szum gradientu i wymaga ostrożnego dostosowania kroków uczenia. Eksperymenty wykazują, że metody adaptacyjne lepiej radzą sobie z dolinami o różnym nachyleniu, a wizualizacje w TensorBoard wskazują na stabilniejsze rozkłady wag i gradientów w przypadku Adam i RMSProp. W praktyce Adam z domyślnymi ustawieniami często osiąga wyniki porównywalne lub lepsze od innych metod, choć wybór algorytmu powinien uwzględniać potrzebę kontroli nad tempem uczenia oraz płynnością zmian w przestrzeni parametrów.