SPRAWOZDANIE

Zajęcia: Nauka o danych II

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 4 Data 20.04.2025 Temat: Praktyczne ćwiczenia z algorytmami optymalizacji dla uczenia maszynowego Wariant 6	Imię Nazwisko Hubert Mentel Informatyka II stopień, niestacjonarne, 2 semestr, gr.1a

1. Zadania:

Zadanie 1.

Celem ćwiczenia jest poznanie podstawowych algorytmów optymalizacji stosowanych w uczeniu maszynowym oraz ich praktyczne wykorzystanie przy trenowaniu modeli.

Zadanie 2.

6. **Wariant 6**

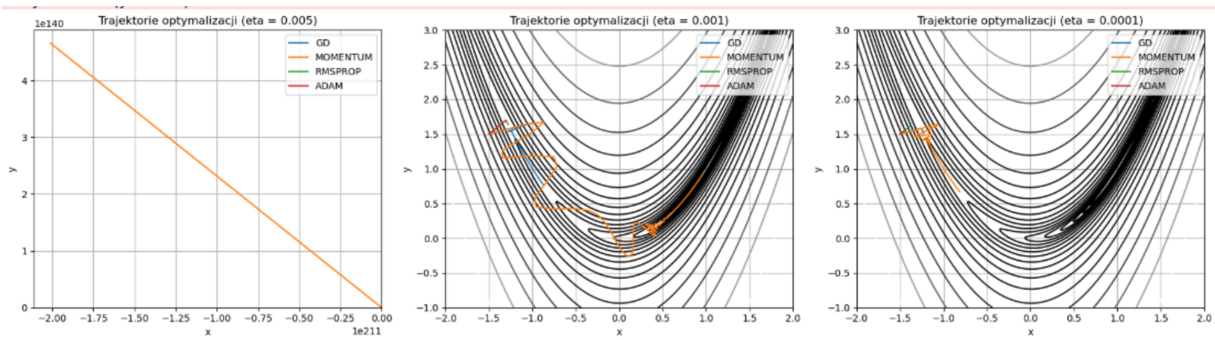
- Testuj $\eta = 0.005, 0.001, 0.0001$.
- Użyj funkcji: $f(x, y) = \log(1 + x^2 + y^2)$.
- Monitoruj zmiany wag w TensorBoard dla 3-warstwowej sieci.

Pliki dostępne są pod linkiem:
https://github.com/HubiPX/NOD/tree/master/NOD2/Zadanie%204

2. Opis programu opracowanego (kody źródłowe, zrzuty ekranu)

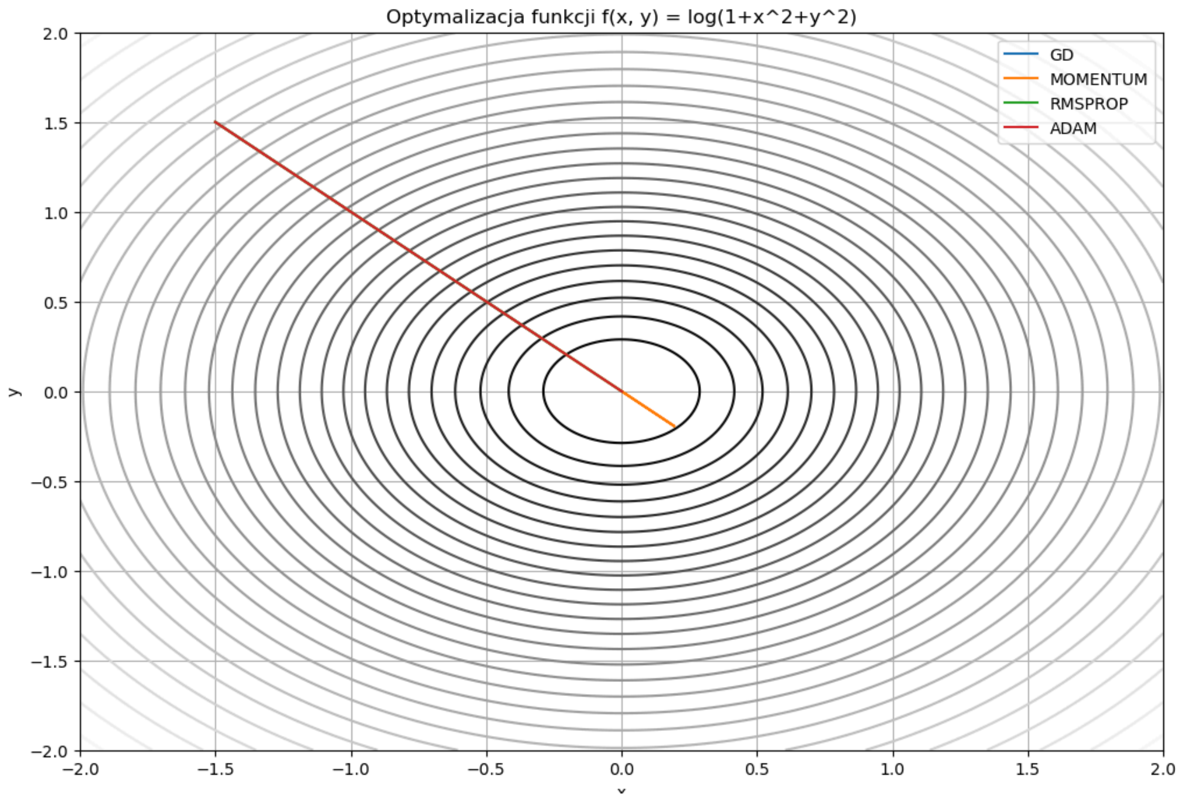
[2]:

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

# Funkcja Rosenbrocka i jej gradient
def rosenbrock(x, y):
    return (1 - x)**2 + 100 * (y - x**2)**2

def rosenbrock_grad(x, y):
    dx = -2 * (1 - x) - 400 * x * (y - x**2)
    dy = 200 * (y - x**2)
    return np.array([dx, dy])

# Funkcja optymalizacji
def optimize_path(opt_name, lr=0.001, steps=500):
    pos = np.array([-1.5, 1.5])
    path = [pos.copy()]
    v, s = np.zeros(2), np.zeros(2)
    beta1, beta2 = 0.9, 0.999
    eps = 1e-8
    for t in range(1, steps + 1):
        grad = rosenbrock_grad(*pos)
        if opt_name == 'gd':
            pos -= lr * grad
        elif opt_name == 'momentum':
            v = beta1 * v + lr * grad
            pos -= v
        elif opt_name == 'rmsprop':
            s = beta2 * s + (1 - beta2) * grad**2
            pos -= lr / (np.sqrt(s) + eps) * grad
        elif opt_name == 'adam':
            v = beta1 * v + (1 - beta1) * grad
            s = beta2 * s + (1 - beta2) * grad**2
            v_corr = v / (1 - beta1**t)
            s_corr = s / (1 - beta2**t)
            pos -= lr * v_corr / (np.sqrt(s_corr) + eps)
        path.append(pos.copy())
    return np.array(path)

# Zakres siatki
x = np.linspace(-2, 2, 400)
y = np.linspace(-1, 3, 400)
X, Y = np.meshgrid(x, y)
Z = rosenbrock(X, Y)

# Wartości learning rate do przetestowania
etas = [0.005, 0.001, 0.0001]

# Rysowanie trzech wykresów
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for i, eta in enumerate(etas):
    ax = axes[i]
    ax.contour(X, Y, Z, levels=np.logspace(-1, 3, 20), cmap='gray')
    for opt in ['gd', 'momentum', 'rmsprop', 'adam']:
        path = optimize_path(opt, lr=eta)
        ax.plot(path[:, 0], path[:, 1], label=opt.upper())
    ax.set_title(f"Trajektorie optymalizacji (eta = {eta})")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend()
    ax.grid()

plt.tight_layout()
plt.show()
```



[3]:

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

# Funkcja celu
def f(x, y):
    return np.log(1+x**2+y**2)

# Gradient funkcji celu (przybliżony numerycznie)
def f_grad(x, y, h=1e-5):
    fx = (f(x + h, y) - f(x - h, y)) / (2 * h)
    fy = (f(x, y + h) - f(x, y - h)) / (2 * h)
    return np.array([fx, fy])

# Funkcja optymalizacji
def optimize_path(opt_name, lr=0.01, steps=200):
    pos = np.array([-1.5, 1.5])
    path = [pos.copy()]
    v, s = np.zeros(2), np.zeros(2)
    beta1, beta2 = 0.9, 0.999
    eps = 1e-8

    for t in range(1, steps + 1):
        grad = f_grad(*pos)
        if opt_name == 'gd':
            pos -= lr * grad
        elif opt_name == 'momentum':
            v = beta1 * v + lr * grad
            pos -= v
        elif opt_name == 'rmsprop':
            s = beta2 * s + (1 - beta2) * grad**2
            pos -= lr / (np.sqrt(s) + eps) * grad
        elif opt_name == 'adam':
            v = beta1 * v + (1 - beta1) * grad
            s = beta2 * s + (1 - beta2) * grad**2
            v_corr = v / (1 - beta1**t)
            s_corr = s / (1 - beta2**t)
            pos -= lr * v_corr / (np.sqrt(s_corr) + eps)
        path.append(pos.copy())

    return np.array(path)

# Rysowanie wykresu
x = np.linspace(-2, 2, 400)
y = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

plt.figure(figsize=(12, 8))
plt.contour(X, Y, Z, levels=30, cmap='gray')

# Uruchomienie optymalizacji dla każdej metody
for opt in ['gd', 'momentum', 'rmsprop', 'adam']:
    path = optimize_path(opt, lr=0.01)
    plt.plot(path[:, 0], path[:, 1], label=opt.upper())

plt.title("Optymalizacja funkcji f(x, y) = min(x, y)^2 + max(x, y)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.show()
```



[4]:

```python
import tensorflow as tf
from tensorflow.keras import layers, models

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

# 2. Definicja 3-warstwowej sieci MLP
model = models.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# 3. Kompilacja modelu
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# 4. Callback TensorBoard z zapisem histogramów wag
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir='logs/3layer_mlp',
    histogram_freq=1,       # zapis histogramów wag co epokę
    write_graph=True,
    write_images=False
)

# 5. Trening modelu z TensorBoard
model.fit(
    x_train, y_train,
    epochs=5,
    validation_data=(x_test, y_test),
    callbacks=[tensorboard_callback]
)
```

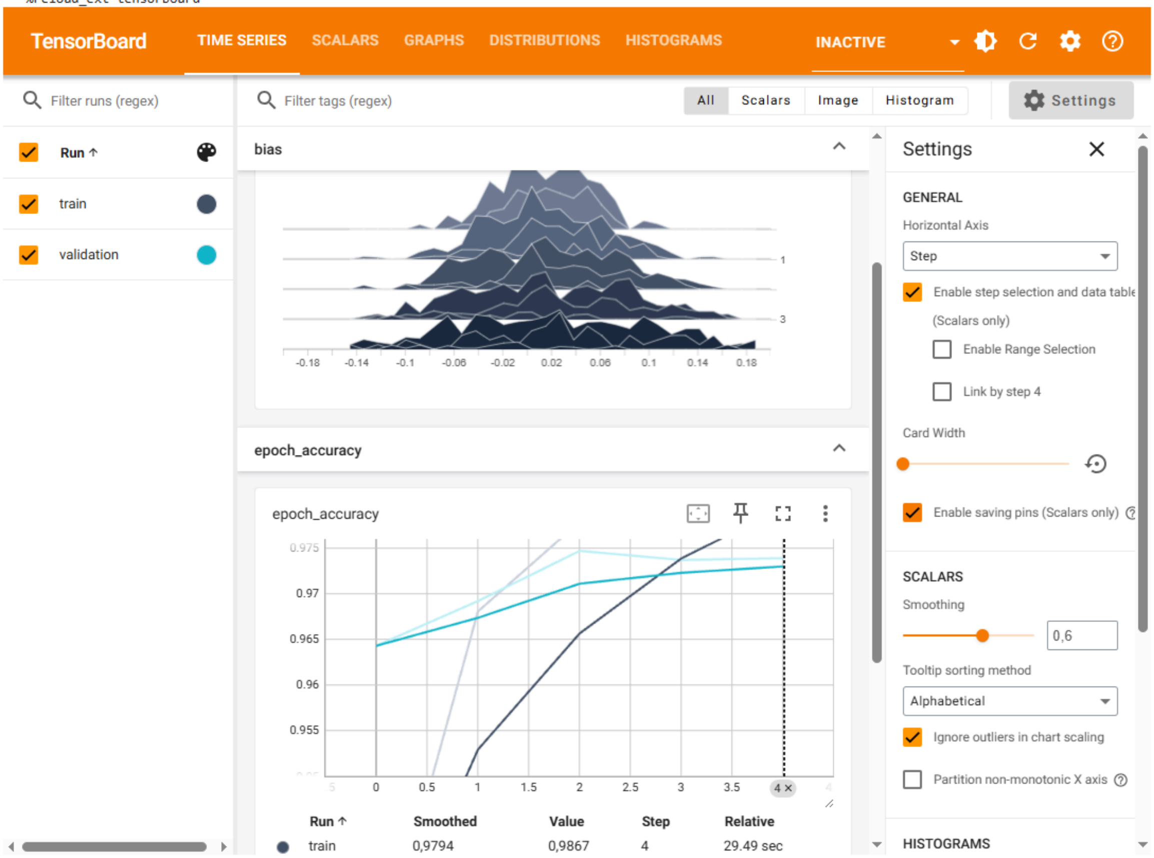
```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8689 - loss: 0.4471 - val_accuracy: 0.9643 - val_loss: 0.1190
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9670 - loss: 0.1049 - val_accuracy: 0.9692 - val_loss: 0.0967
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9787 - loss: 0.0679 - val_accuracy: 0.9747 - val_loss: 0.0844
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9844 - loss: 0.0482 - val_accuracy: 0.9737 - val_loss: 0.0843
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9885 - loss: 0.0378 - val_accuracy: 0.9739 - val_loss: 0.0804
```

[4]:

```
<keras.src.callbacks.history.History at 0x1f0d4e32210>
```

```
[8]: %load_ext tensorboard
     %tensorboard --logdir logs/3layer_mlp
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```



3. Wnioski

W ramach wersji 6 przeprowadzono testy dla trzech różnych wartości współczynnika uczenia (η = 0.005, 0.001, 0.0001) oraz funkcji celu $f(x, y) = \log(1 + x^2 + y^2)$. Zastosowanie tej funkcji pozwoliło zaobserwować, jak optymalizatory radzą sobie z łagodnym, wypukłym krajobrazem. W szczególności metody adaptacyjne, takie jak Adam i RMSProp, wykazały lepszą zbieżność przy większych wartościach η, natomiast dla bardzo małego learning rate proces uczenia był znacznie wolniejszy. Do trenowania 3-warstwowej sieci neuronowej zastosowano TensorBoard, co umożliwiło monitorowanie zmian wag w czasie. Narzędzie to okazało się pomocne w analizie dynamiki uczenia i rozkładów wag w kolejnych epokach, co ułatwiło ocenę skuteczności optymalizacji.