SPRAWOZDANIE

Zajęcia: Nauka o danych 2

Prowadzący: prof. dr hab. inż. Vasyl Martsenyuk

Laboratorium Nr 1 **Data** 22.10.2025 **Temat: "** " **Wariant** 10	Jakub Janik Informatyka II stopień, stacjonarne, 2 semestr, gr. CB

1. Cel ćwiczenia:

Cel: Zapoznanie się z praktycznym wykorzystaniem i różnicami w działaniu podstawowych i zaawansowanych algorytmów optymalizacji (Gradient Descent, Momentum, RMSProp, Adam) oraz ich wpływem na proces uczenia.
Wariant 10:

Współczynniki uczenia (η): 0.02,0.002,0.0002.

Funkcja celu: f(x,y)=arctan(xy)+y2.

Zadanie z TensorBoard: Śledzenie dokładności (accuracy) i straty (loss) dla każdej epoki podczas trenowania modelu MLP na danych MNIST.

2. Przebieg ćwiczenia:

Analiza Wizualna Algorytmów Optymalizacji

Przeprowadzono symulację optymalizacji dla czterech algorytmów na funkcji f(x,y)=arctan(xy)+y2. Funkcja ta charakteryzuje się wydłużoną, płytką rynną z minimum w punkcie (0,0), co stanowi wyzwanie dla standardowego algorytmu GD.

2.1. Wpływ Współczynnika Uczenia (η)

Parametr η	Obserwowane Zachowanie Trajektorii	Wniosek
0.02 (Wysoki)	GD i Momentum wykazują dużą niestabilność i silne oscylacje wzdłuż stromego zbocza rynny. Adam i RMSProp utrzymują stabilność, ale początkowo ich kroki są również duże, co opóźnia precyzyjne wejście do minimum.	Wysoki współczynnik η prowadzi do niestabilności w przypadku algorytmów bez adaptacji (GD, Momentum).

0.002 (Umiarkowany)	GD i Momentum stają się stabilne, ale ich zbieżność jest powolna. Adam i RMSProp wykazują najlepszą wydajność, szybko wchodząc w rynnę minimum i precyzyjnie się do niego zbliżając.	Jest to optymalny zakres η, w którym algorytmy adaptacyjne wykorzystują swoje mechanizmy różnicujące kroki.
0.0002 (Niski)	Wszystkie algorytmy są bardzo stabilne, ale ich zbieżność jest ekstremalnie powolna i niemal identyczna.	Za niski η niweluje przewagę zaawansowanych algorytmów, spowalniając trening i marnując czas obliczeniowy.

2.2. Porównanie Algorytmów (Adam vs GD)

- Gradient Descent (GD): Jest wrażliwy na anizotropową (nierówną) skalę funkcji celu. W wąskiej rynnie minimum wykonuje minimalny postęp w kierunku minimum, jednocześnie silnie oscylując w poprzek rynny, co jest jego główną wadą.
- Adam (Adaptive Moment Estimation): Jest zdecydowanie najbardziej efektywny. Dzięki wykorzystaniu pędu (Momentum) i adaptacyjnego współczynnika uczenia (RMSProp), Adam automatycznie zmniejsza kroki wzdłuż stromych osi i zwiększa je wzdłuż płytkich osi, co skutkuje najszybszą i najbardziej płynną trajektorią zbieżności.

Kod źródłowy:

```
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.colors as colors
import tensorflow as tf
import os
import datetime
import warnings
from tensorflow.keras import layers, models
import logging
tf.get_logger().setLevel(logging.ERROR)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
# ----------------------------------------------------------------------
# 1. Definicja funkcji celu i jej gradientu # -------------------------
-----------------------------------------

def custom_function(x, y):
  """Funkcja celu: f(x, y) = arctan(xy) + y^2."""
  return np.arctan(x * y) + y**2
```

```
def custom_function_grad(x, y):
    """Gradient funkcji celu: [df/dx, df/dy]"""
    denominator = 1 + (x * y)**2
    df_dx = y / denominator
    df_dy = x / denominator + 2 * y
    return np.array([df_dx, df_dy])

# ----------------------------------------------------------------------#
# 2. Implementacja Optymalizatorów
# ----------------------------------------------------------------------

def optimize_path(opt_name, lr, steps=1000, initial_pos=np.array([-2.0, 2.0])):
    """Śledzi ścieżkę optymalizacji."""
    pos = initial_pos.copy()
    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 = custom_function_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)

# ----------------------------------------------------------------------#
# 3. WIZUALIZACJA I PORÓWNANIE WYNIKÓW (WERSJA CICHA)
# ----------------------------------------------------------------------

warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)

learning_rates = [0.02, 0.002, 0.0002]
```

```
] ]
optimizers = ['gd', 'momentum', 'rmsprop', 'adam'
x =-33) y =-33) XY =y)                              , 400,  np.linspace(
Z =Y)                                   (X,         , 400,
 ,                  x,  custom_function
                np.meshgrid(    min_Z =   .min(Z[np.isfinite(Z)])  np.linspace(
                                    np
max_Z_clip = min_Z + 5


plt.suptitle(f"Porównanie trajektorii optymalizacji dla f(x, y) = arctan(xy) +
fig, axes =     .subplots(1, len(learning_rates), figsize=(21, 7))
           plt
                 )
y^2", fontsize=18
=                  xlim_list    -25, 3], [-   , 3], [-  , 3]]
=                  ylim_list [[          4          3
                                -  , 3], [-  , 3], [-  , 3]]
                             [[ 3          3          3
for i, lr in
enumerate(learning_rates):
  ax = axes[i]

  min_log_val = np.log10(min_Z + 1e-4)
  max_log_val = np.log10(max_Z_clip)
  levels = np.logspace(min_log_val, max_log_val, 20)

  try:
    ax.contour(X, Y, Z, levels=levels, cmap='viridis',
norm=colors.LogNorm(vmin=min(levels), vmax=max(levels)))
except ValueError:
    levels_linear = np.linspace(min_Z, max_Z_clip, 20)
    ax.contour(X, Y, Z, levels=levels_linear, cmap='viridis')


                             ])ax.set_xlim(xlim_list[i
  ax.set_ylim(ylim_list[i])

  ax.plot(0, 0, 'r*',               12, label='Minimum (0, 0)', zorder
markersize==5)

  for opt in optimizers:
    path = optimize_path(opt,         lr, steps=1000 lr=)
    ax.plot(path[:, 0], path[:,       ], 'o-', markersize=3, alpha=0.8,
1 label=opt.upper(), lw=2)
    ax.plot(path[-1, 0], path[-1, 1], 'x', markersize=7, markeredgewidth=2)

  ax.set_title(f"Współczynnik uczenia (eta): {lr}", fontsize=14)
ax.set_xlabel("x")
```

```python
    ax.set_ylabel("y")
    ax.legend(loc='upper right', fancybox=True, shadow=True)
    ax.grid(True, alpha=0.4)

plt.tight_layout(rect=[0, 0.03, 1, 0.95]).
plt.show()

warnings.filterwarnings("default", category=RuntimeWarning.()
warnings.filterwarnings("default", category=UserWarning.()

# ---------------------------------------------------------------------#
4.1 Regresja liniowa przy użyciu GD
# ---------------------------------------------------------------------

np.random.seed(42)
X = 2 *1) np.random.rand(100,
y = 4 + 3 * X +1) np.random.randn(100,

X_b = np.c_[np.ones((100, 1)), X]
1) theta = np.random.randn( 2,
n_iterations = 1000
eta = 0.1
m = 100
print \n--- 4.1 Regresja Liniowa z GD ---"(")
print "Początkowe współczynniki (theta):\n", theta.flatten())(
print "-" * 30 ()

for iteration in range(n_iterations):
    gradients = 2 / m * X_b.T @ (X_b @ theta - y)
    theta = theta - eta * gradients

print "Końcowe współczynniki regresji (theta):\n", theta.flatten())(
print "Oczekiwane wartości (b=4, a=3):\n [4.        3.        ]"()
# ------
--------------------------------------------------------------# 6.
Wprowadzenie do TensorBoard
# ---------------------------------------------------------------------

# 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 modelu MLP
model = models.Sequential([
layers.Input(shape=(784,)),
```

```
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

# Konfiguracja TensorBoard
log_dir = os.path.join("logs", "fit", "wariant_10_mlp_" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(
log_dir=log_dir,
histogram_freq=0)

print("\n--- 6. Wprowadzenie do TensorBoard ---")
print("Uruchomienie TensorBoard: Po zakończeniu treningu uruchom w terminalu:")
print(f"\n\n    tensorboard --logdir={os.path.join('logs', 'fit')}\n")

# Trening modelu z zapisem do TensorBoard
history = model.fit(
    x_train,
    y_train,
    epochs=5,
    validation_data=(x_test, y_test),
    callbacks=[tensorboard_callback],
    verbose=1
)
```

Wyniki Regresji Liniowej z Gradient Descent

Przeprowadzono regresję liniową na syntetycznych danych zaszumionych, gdzie oczekiwane parametry to b=4.0 (wyraz wolny) i a=3.0 (współczynnik kierunkowy).

Parametr	Wartość Początkowa	Wartość Końcowa (GD)	Wartość Oczekiwana
B	0.0130	4.2151	4.0
a	1.4535	2.7701	3.0

Wniosek: Algorytm GD zbiegł do parametrów **b=4.2151** i **a=2.7701**. Odchylenie od oczekiwanych wartości (4.0 i 3.0) wynika z **szumu dodanego do danych wejściowych** (np.random.randn). Otrzymane wyniki potwierdzają, że algorytm GD poprawnie minimalizuje błąd średniokwadratowy (MSE) i znalazł najlepsze dopasowanie dla zaszumionego zbioru danych.

4. Analiza TensorBoard

Model MLP (Multi-Layer Perceptron) został wytrenowany na zbiorze MNIST w ciągu 5 epok, a metryki zostały zapisane za pomocą tf.keras.callbacks.TensorBoard.

Wyniki Metryk (Epoka 5)

Metryka	Zbiór Treningowy	Zbiór Walidacyjny
Loss (Strata)	0.0469	0.0709
Accuracy (Dokładność)	98.52%	97.87%

Interpretacja Wzorców (Wykresy z TensorBoard)

Zbieżność Strata/Loss: Na wykresie Strata (Loss) widoczny jest ciągły spadek zarówno dla danych treningowych, jak i walidacyjnych. Strata walidacyjna (0.0709) utrzymuje się blisko straty treningowej (0.0469), co jest zdrowym symptomem.
Zbieżność Dokładność/Accuracy: Na wykresie Dokładność (Accuracy) widoczny jest ciągły wzrost obu linii do poziomu bliskiego 98%.
Wnioski nt. Uogólniania: Model osiągnął bardzo wysoką dokładność na danych testowych (97.87%). Minimalna różnica w dokładności między zbiorem treningowym a walidacyjnym (mniej niż 1%) świadczy o braku znaczącego przeuczenia (overfittingu) i dużej zdolności modelu do uogólniania wiedzy.

Link do repozytorium:https://github.com/Uczelniane/NODII.git

3. Wnioski

Metody adaptacyjne (Adam, RMSProp) są znacznie lepsze w radzeniu sobie z trudnymi krajobrazami funkcji celu i są mniej wrażliwe na dobór współczynnika uczenia niż klasyczny GD.
Istnieje **optymalny zakres** współczynnika uczenia (η=0.002) dla danego problemu, przy którym zaawansowane algorytmy wykazują pełnię swoich możliwości.
Narzędzie **TensorBoard** jest kluczowe w monitorowaniu i diagnostyce procesu treningu modeli głębokiego uczenia.