SPRAWOZDANIE

Zajęcia: Uczenie Maszynowe

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 4 Data 11.01.2025 Temat: "5. Implementacja algorytm´ow optymalizacji gradientowej do trenowania modeli. 6. Projektowanie i trening prostych sieci neuronowych w TensorFlow lub PyTorch. 7. Zastosowanie konwolucyjnych sieci neuronowych (CNN) do analizy obrazu" Wariant 10	Anna Więzik Informatyka II stopień, niestacjonarne, 1 semestr, gr.1b

1. Polecenie:

10. **Wariant 10**

- Optymalizuj funkcję $f(x) = \sqrt{|x| + 1} + x^2$ metodą spadku gradientu i zwizualizuj proces.
- Zbuduj sieć neuronową do regresji na zbiorze Boston Housing z przekształconą jedną cechą.
- Zaprojektuj, wytrenuj i przetestuj sieć konwolucyjną na zbiorze CIFAR-100.

Ze wzgledów etycznych Boston Housing został usunięty, użyłam California Housing

2. Link do repozytorium:

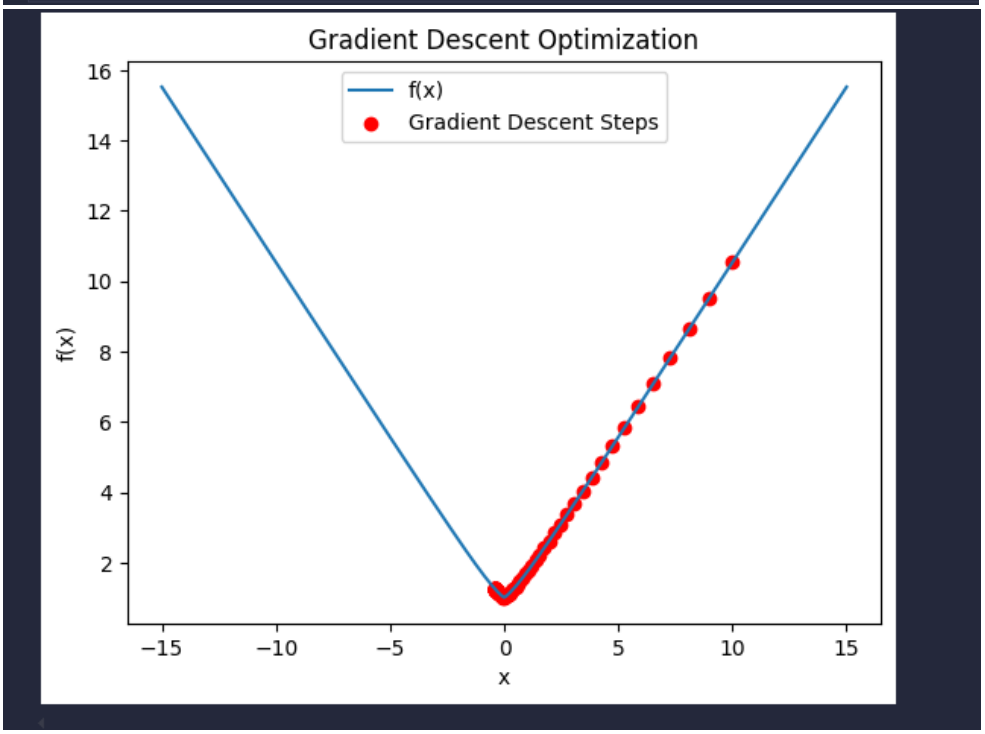
Link: https://github.com/AnaShiro/UM_2024

3. Opis programu opracowanego

1) Zaimplementuj w Pythonie optymalizację funkcji metodą spadku gradientu wraz z wizualizacją.

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

def gradient_descent(f, grad_f, theta_init, learning_rate, iterations):
    theta = theta_init
    history = [theta]
    for i in range(iterations):
        theta -= learning_rate * grad_f(theta)
        history.append(theta)
    return theta, history

# Definicja funkcji i jej pochodnej
f = lambda x: np.sqrt(np.abs(x) + 1 + x**2)
grad_f = lambda x: (0.5 * (2*x + 1/np.sqrt(np.abs(x) + 1 + x**2)))

# Parametry
theta_init = 10.0
learning_rate = 0.1
iterations = 100

# Optymalizacja
optimal_theta, history = gradient_descent(f, grad_f, theta_init, learning_rate, iterations)

# Wizualizacja
x = np.linspace(-15, 15, 400)
y = f(x)

plt.plot(x, y, label='f(x)')
plt.scatter(history, f(np.array(history)), color='red', label='Gradient Descent Steps')
plt.legend()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent Optimization')
plt.show()

print("Optymalne theta:", optimal_theta)
```

```
[1]  ✓ 1.8s
```



2) Stwórz w Pythonie najprostszą sieć neuronową wraz z ewaluacją i prognozowaniem.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential    Import "tensorflow.keras.models" could not be resolved
from tensorflow.keras.layers import Dense    Import "tensorflow.keras.layers" could not be resolved
Qodo Gen: Options | Test this function
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Wczytanie danych California Housing
california = fetch_california_housing()
X, y = california.data, california.target

# Przekształcenie jednej cechy (np. standaryzacja)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Podział na dane treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Budowa modelu
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1)  # Warstwa wyjściowa
])

# Kompilacja
model.compile(optimizer='adam', loss='mse')

# Trening
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)

# Ocena modelu
loss = model.evaluate(X_test, y_test)
print("Strata na danych testowych:", loss)
```

```
Epoch 1/100

  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 1.5865 - val_loss: 0.4615
Epoch 2/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3997 - val_loss: 0.4267
Epoch 3/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.3872 - val_loss: 0.4439
Epoch 4/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.3895 - val_loss: 0.3734
Epoch 5/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3286 - val_loss: 0.3685
Epoch 6/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.3171 - val_loss: 0.3444
Epoch 7/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3217 - val_loss: 0.3446
Epoch 8/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3102 - val_loss: 0.3332
Epoch 9/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2952 - val_loss: 0.3341
Epoch 10/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2892 - val_loss: 0.3311
Epoch 11/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2971 - val_loss: 0.3199
Epoch 12/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.2931 - val_loss: 0.3424
Epoch 13/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.2835 - val_loss: 0.3295
...
Epoch 100/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2081 - val_loss: 0.2691
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2642
Strata na danych testowych: 0.2725612521171757
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

3) Zaprojektuj, wytrenuj i przetestuj sieć konwolucyjną, wykorzystując jeden z dostępnych w Pythonie podstawowych zbiorów danych.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential    Import "tensorflow.keras.models" could not be resolved
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPooling2D    Import "tensorflow.keras.layers" could not be resolved
from tensorflow.keras.datasets import cifar100    Import "tensorflow.keras.datasets" could not be resolved
from tensorflow.keras.utils import to_categorical    Import "tensorflow.keras.utils" could not be resolved

# Wczytanie danych CIFAR-100
(X_train, y_train), (X_test, y_test) = cifar100.load_data()
y_train = to_categorical(y_train, 100)
y_test = to_categorical(y_test, 100)

# Budowa modelu
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(100, activation='softmax')
])

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

# Trening
model.fit(X_train, y_train, epochs=20, batch_size=64, validation_split=0.2)

# Ocena modelu
loss, accuracy = model.evaluate(X_test, y_test)
print("Strata na danych testowych:", loss)
print("Dokładność na danych testowych:", accuracy)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169001437/169001437 ━━━━━━━━━━━━━━━━━━━━ 9s 0us/step


Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 29ms/step - accuracy: 0.0200 - loss: 8.9081 - val_accuracy: 0.0584 - val_loss: 4.2730
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 28ms/step - accuracy: 0.0795 - loss: 4.1209 - val_accuracy: 0.1158 - val_loss: 3.9113
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 23s 32ms/step - accuracy: 0.1401 - loss: 3.7364 - val_accuracy: 0.1450 - val_loss: 3.8003
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 31ms/step - accuracy: 0.2176 - loss: 3.3101 - val_accuracy: 0.1634 - val_loss: 3.6641
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 29ms/step - accuracy: 0.3041 - loss: 2.8781 - val_accuracy: 0.1950 - val_loss: 3.5932
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 30ms/step - accuracy: 0.4101 - loss: 2.3665 - val_accuracy: 0.2132 - val_loss: 3.8154
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 29ms/step - accuracy: 0.5191 - loss: 1.8705 - val_accuracy: 0.2120 - val_loss: 4.1251
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 30ms/step - accuracy: 0.6266 - loss: 1.4187 - val_accuracy: 0.2097 - val_loss: 4.8582
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 29ms/step - accuracy: 0.7117 - loss: 1.0648 - val_accuracy: 0.2094 - val_loss: 5.5740
Epoch 10/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 29ms/step - accuracy: 0.7688 - loss: 0.8525 - val_accuracy: 0.2035 - val_loss: 6.2577
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 29ms/step - accuracy: 0.8066 - loss: 0.6924 - val_accuracy: 0.2073 - val_loss: 6.8779
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 29ms/step - accuracy: 0.8365 - loss: 0.5657 - val_accuracy: 0.2062 - val_loss: 7.6468
Epoch 13/20
...
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 30ms/step - accuracy: 0.9138 - loss: 0.3195 - val_accuracy: 0.1954 - val_loss: 13.3479
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.1965 - loss: 13.0540
Strata na danych testowych: 13.029065132141113
Dokładność na danych testowych: 0.20010000467300415
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

4. Wnioski

Optymalizacja gradientowa to kluczowy mechanizm stosowany w celu minimalizacji funkcji kosztu J(θ) podczas treningu modeli.

Kategorialna krosentropia jest popularną funkcją kosztu wykorzystywaną w zadaniach klasyfikacji wieloklasowej, skutecznie karzącą model za przypisywanie niskiego prawdopodobieństwa prawdziwej klasie.

Sieci neuronowe przekształcają dane wejściowe w wyniki. Architektura modelu obejmuje warstwę wejściową, warstwy ukryte, które przekształcają dane za pomocą funkcji aktywacji, oraz warstwę wyjściową, która generuje wyniki. Konwolucyjne sieci neuronowe (CNN) analizują obrazy, wychwytując lokalne wzorce za pomocą operacji konwolucji.