Sieci Neuronowe 2

Zadanie 1

Sprawdź możliwości wykorzystania pakietu tensorflow do budowy wielowarstwowej sieci neuronowej. W tym celu:

A. Zaimportuj potrzebne biblioteki

```
import tensorflow as tf
from tensorflow import keras

from matplotlib import pyplot as plt
import numpy as np

import sklearn.metrics
```

B. Załaduj dane ze zbioru MNIST oraz wykorzystaj metodę *normalize,* aby przeprowadzić normalizację danych

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

x_train_normalized = tf.keras.utils.normalize(x_train, axis=-1, order=2)
x_test_normalized = tf.keras.utils.normalize(x_test, axis=-1, order=2)
```

C. Wykorzystując funkcję *plot_random_pics* zobacz jak wyglądają dane w zbiorze MNIST

```
def plot_random_pics(data, nmb_of_pics =10):
   fig, axes1 = plt.subplots(nmb_of_pics, nmb_of_pics,figsize=(8,8))

   for j in range(nmb_of_pics):
       for k in range(nmb_of_pics):
           i = np.random.choice(range(len(data)))
           axes1[j][k].set_axis_off()
           axes1[j][k].imshow(data[i], cmap="binary")
```

D. Zdefiniuj model sieci z jedną warstwą ukrytą z 40-stoma neuronami oraz funkcją aktywacji ReLU. W warstwie wyjściowej wykorzystaj funkcję softmax. Wykorzystaj Sequential API.

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.Dense(..., activation="..."))
model.add(keras.layers.Dense(..., activation="..."))
```

E. Skonfiguruj model wykorzystując metodę model.compile() tak, aby wykorzystywał:
   a. funkcję straty będącą wariantem entropii krzyżowej (ang. cross entropy) - SparseCategoricalCrossentropy
   b. Optymalizator wykorzystujący metodę stochastycznego spadku gradientu

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

F. Przeprowadź trening sieci neuronowej w 3 epokach, wykorzystując 20% danych jako zbiór walidacyjny.

```
history = model.fit(x_train_normalized, y_train,
                    epochs=..., validation_split=..., verbose = 2)
```

G. Przeprowadź ewaluację na zbiorze testowym. Przedstaw macierz pomyłek dla poszczególnych klas.

```
y_test_pred = model.predict(...)

matrix = sklearn.metrics.confusion_matrix(y_test, y_test_pred.argmax(axis=1))
matrix
```

H. Przygotuj kilka własnoręcznie napisanych cyfr w przekształconych do formatu 28 x 28 pikseli. Spróbuj wykorzystać zbudowany model do predykcji. Co zauważyłaś?

```
img=cv2.imread('your_image.bmp', cv2.IMREAD_GRAYSCALE)
img = 255 - img

plt.imshow(img, cmap="binary")

img_normalized = tf.keras.utils.normalize(img, axis=-1, order=2)

np.expand_dims(img, axis=0)
model.predict(np.expand_dims(img, axis=0)).argmax()
```

Zadanie 2

Dla wywołania metody model.fit() z Zadania 1 odpowiedz na poniższe pytania (posiłkując się wiedzą zdobytą we wprowadzeniu oraz dokumentacją na metody https://keras.io/api/models/model_training_apis/#fit-method):

A. Ile wynosi i co oznacza parametr `batch_size`?
B. Co oznacza oraz skąd wzięła się taka wartość liczby “1500” podawana na początku każdej linii logów w procesie uczenia?

Zadanie 3

A) Przeprowadź eksperyment sprawdzający czy w rzeczywistości architektury głębokie sieci powinny być preferowane w stosunku do płytkich. Eksperyment przeprowadź dla liczby warstw od 1 do 9. Skorzystaj ze zbioru MNIST. Procedurę walidacji przeprowadź na próbie testowej dla tego zbioru. Dla każdej architektury (= liczby warstw ukrytych) zapewnij, że w ramach warstw ukrytych będzie się znajdowało łącznie 600 neuronów (z dokładnością do zaokrągleń).

   Pamiętaj o tym, iż z uwagi na losowy sposób inicjalizowania wag, sieci o tej samej architekturze mogą się istotnie różnić w zakresie ich skuteczności (mierzonej np. poprzez accuracy). Aby zniwelować ten błąd wynikający z losowości należałoby przeprowadzić znaczną ilość eksperymentów dla tej samej architektury a wynik uśrednić. Przyjmijmy na potrzeby tego zadania, że

uśrednienie na podstawie 5 powtórzonych procedur trenowania dla tej samej architektury jest wystarczające.

```
nmb_of_tests_per_architecture = ...
nmb_of_neurons_at_disposal = ...

results_for_nmb_of_hidden_layers = {}

for current_nmb_of_hidden_layers in range(1,...):
   current_val_accuracy = 0

   for j in range(nmb_of_tests_per_architecture):
       keras.backend.clear_session()
       del model

       model = keras.models.Sequential()

       model.add(keras.layers.Flatten(input_shape=[...,...]))

       for hidden_layer in range(1, ...):

model.add(keras.layers.Dense(int(nmb_of_neurons_at_disposal/current_nmb_of_hidden_la
yers), activation="relu"))

       model.add(keras.layers.Dense(..., activation="softmax"))

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

       history = model.fit(x_train_normalized, y_train, epochs=4,
validation_data=(x_test_normalized, y_test), verbose=2)

       current_val_accuracy += max(history.history['val_accuracy'])

   current_val_accuracy = current_val_accuracy / ...

   results_for_nmb_of_hidden_layers[current_nmb_of_hidden_layers] =
current_val_accuracy
```

B) Narysuj wykres liniowy zależności liczby warstw ukrytych w stosunku do uśrednionego Accuracy dla danej liczby warstw. Zinterpretuj wykres.

Zadanie 4

Narysuj, wykorzystując funkcje z biblioteki Keras następujące funkcje aktywacji: sigmoid, tangens hiperboliczny, SoftSign, ReLU, SoftPlus, SeLU, ELU. Posłuż się poniższym kodem, aby narysować dwa wykresy w zależności od wartości funkcji aktywacji.

W komentarzu opisz wpływ wyboru poszczególnych grup funkcji aktywacji na proces uczenia metodą gradientową.

```
x = np.linspace(-10,10,200)
a = tf.constant(x, dtype = tf.float32)

y_sigmoid = tf.keras.activations.sigmoid(a).numpy()
y_tanh= ...
y_softsign= ...
y_relu = ...
y_softplus = .
y_selu = ...
y_elu = ...

fig = plt.figure(figsize=(10, 10))

ax = fig.add_subplot(2, 1, 1)

ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.grid()


plt.plot(x, y_tanh, label='tanh')
plt.plot(x, y_sigmoid, label='sigmoid')
plt.plot(x, y_softsign, label='softsign')
ax.legend(loc="upper left")


ax = fig.add_subplot(2, 1, 2)
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.plot(x, y_relu, label='ReLU')
plt.plot(x, y_softplus, label='SoftPlus')
plt.plot(x, y_selu, label='SeLU')
```

```
plt.plot(x, y_elu, label='ELU')
ax.legend(loc="upper left")

ax.grid()

plt.show()
```

Zadanie 5

Zbuduj prostą sieć trójwarstwową z 40 neuronami w warstwie ukrytej. Wykorzystaj dwie funkcje aktywacji: ReLU oraz Sigmoid. Nie normalizuj danych przed procesem uczenia. Przeprowadź dla każdej z nich proces uczenia w standardowych ustawieniach przez 5 epok. Co zaobserwowałeś? Zaobserwuj i skomentuj zdolności uczenia się obu sieci.

Zadanie 6

Dla prostej sieci z poprzedniego zadania dodaj warstwę w postaci `keras.layers.BatchNormalization()` przed warstwą funkcji aktywacji. W tym celu musisz dodać funkcję aktywacji jako kolejną warstwę (wykorzystując https://keras.io/api/layers/activation_layers/relu/). Jak wpływa dodanie Batch Normalization na szybkość i skuteczność trenowania?

Zadanie 7

UWAGA!!! - do uruchomienia tylko na COLAB

Dla zbioru MNIST skonstruuj sieć o następującej po sobie warstwach:

- konwolucyjna z 64 filtrami, rozmiar okna - 7 pikseli, f. aktywacji ReLU

```
keras.layers.Conv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1],
activation='relu', padding="SAME")
```

aby dostosować dane wejściowe przeprowadź następującą modyfikację danych wejściowych:

```
x_train_normalized_conv = x_train_normalized[..., np.newaxis]
```

- `MaxPooling2D` o rozmiarze 2

```
keras.layers.MaxPooling2D(pool_size=2)
```

- konwolucyjna z 128 filtrami, rozmiar okna - 7, f. aktywacji ReLU
- `MaxPooling2D` o rozmiarze 2
- Dense o rozmiarze 128 z f. aktywacji ReLU

Jakie wyniki uzyskałaś? Wykorzystaj wiedzę o “tunowaniu” sieci zdobytą w poprzednich zadaniach, aby spróbować poprzez wprowadzone zmiany albo a) uzyskać lepsze wyniki dla zbioru walidacyjnego albo b) uzyskać podobne wyniki ale skrócić czas procesu trenowania.

Sprawdź czy własne cyfry z Zadania 1 zostają poprawnie zaklasyfikowane. W szczególności zwróć uwagę na przypadki błędnie klasyfikowane w poprzednim eksperymencie.