Лабораторна робота
“MNIST”

Група ТК-41
Соловей Євгеній

2025

Зміст

Зміст **2**
Мета роботи **3**
Опис датасету 3
Архітектура нейронної мережі 3
Додаток **5**
Приклад виводу: 7

github: https://github.com/TheYev/KNU/tree/main/NHCI/lab2

Мета роботи

Метою даної роботи є створення, навчання та оцінка нейронної мережі для класифікації рукописних цифр на основі датасету MNIST. Для реалізації використовується бібліотека Keras, яка є високорівневим API для TensorFlow.

Опис датасету

У роботі використовується датасет MNIST, який складається з 70 000 зображень рукописних цифр розміром 28×28 пікселів у градаціях сірого:

- 60 000 зображень у тренувальній вибірці
- 10 000 зображень у тестовій вибірці
- Кожне зображення подано у вигляді 784 пікселів (28×28) та має мітку — від 0 до 9.

Архітектура нейронної мережі

Архітектура нейронної мережі складається з кількох щільно з'єднаних (Dense) шарів. Перший шар містить 512 нейронів і приймає на вхід вектор з 784 ознак (розгортка зображення 28x28 пікселів у плоский вектор). Активаційною функцією на цьому шарі є ReLU (Rectified Linear Unit), яка добре працює для задач класифікації та дозволяє уникнути проблеми зникаючого градієнта.

Після кожного шару з нейронами додається Dropout-шар із ймовірністю 0.2. Dropout вимикає випадкову частину нейронів під час навчання, що допомагає уникнути перенавчання (overfitting) та покращує здатність моделі до узагальнення.

Після другого прихованого шару знову додається ReLU-активація, і ще один Dropout для регуляризації. Завершується мережа вихідним шаром із 10 нейронів, де кожен відповідає одному з можливих класів (цифри від 0 до 9). На цьому шарі використовується функція активації softmax, яка перетворює

вихідні значення в ймовірнісне розподілення — модель визначає ймовірність належності зображення до кожного класу.

Для навчання моделі було використано оптимізатор Adam, що є адаптивним методом градієнтного спуску і дозволяє швидко досягати гарної збіжності. В якості функції втрат застосовується категоріальна кросентропія, яка добре підходить для багатокласової класифікації. Навчання відбувалося протягом десяти епох з розміром пакета даних 128.

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.8613 - loss: 0.4574
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.9669 - loss: 0.1094
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9759 - loss: 0.0724
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9834 - loss: 0.0522
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.9864 - loss: 0.0407
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9867 - loss: 0.0390
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9880 - loss: 0.0346
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9906 - loss: 0.0266
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.9923 - loss: 0.0256
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9930 - loss: 0.0203
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9763 - loss: 0.0963
```

Після навчання модель показала високу точність на тестовій вибірці. Це свідчить про її здатність узагальнювати інформацію з навчального набору і правильно класифікувати нові зображення. Така архітектура є достатньо простою, але вже демонструє високі результати на базових задачах машинного навчання.

Додаток

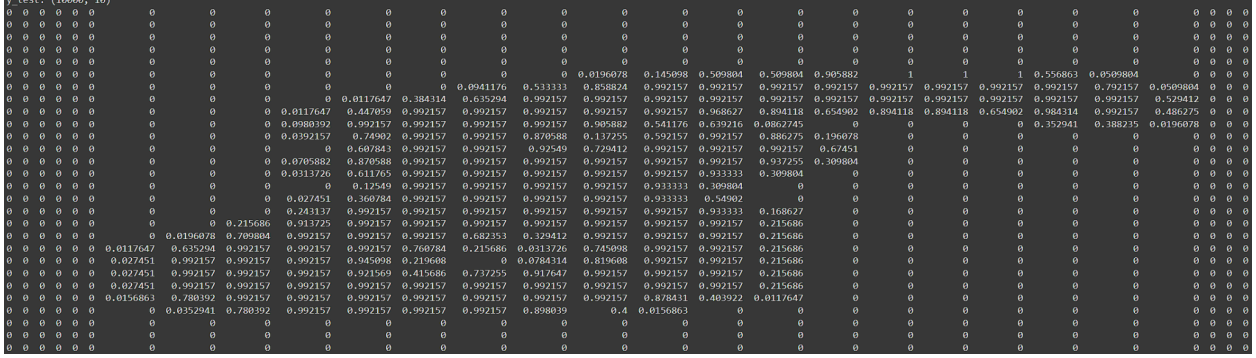
```
# %pip install tensorflow pandas numpy matplotlib

import numpy as np                    # advanced math library
import pandas as pd
import matplotlib.pyplot as plt      # MATLAB-like plotting routines
import random                         # for generating random numbers

from tensorflow.keras.models import Sequential  # Model type to be used
from tensorflow.keras.layers import Dense, Dropout, Activation  # Types of
layers
from tensorflow.keras.utils import to_categorical  # One-hot encoding


from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Завантаження датасету
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Нормалізація
X_train = X_train.reshape(-1, 784).astype("float32") / 255.0
X_test = X_test.reshape(-1, 784).astype("float32") / 255.0

# One-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

print('X_train:', X_train.shape)
print('y_train:', y_train.shape)
print('X_test:', X_test.shape)
print('y_test:', y_test.shape)

# Візуалізація зображень
x_train = X_train.reshape(-1, 28, 28)
x_test = X_test.reshape(-1, 28, 28)

plt.rcParams['figure.figsize'] = (9, 9)
```

```
for i in range(9):
    plt.subplot(3, 3, i + 1)
    num = random.randint(0, len(x_train) - 1)
    plt.imshow(x_train[num], cmap='gray', interpolation='none')
    plt.title("Class {}".format(np.argmax(y_train[num])))

plt.tight_layout()

# Функція для друку матриці
def matprint(mat, fmt="g"):
    col_maxes = [max([len(("{:" + fmt + "}").format(x)) for x in col]) for
col in mat.T]
    for x in mat:
        for i, y in enumerate(x):
            print(("{:" + str(col_maxes[i]) + fmt + "}").format(y), end="
")
        print("")

# Приклад виводу матриці
matprint(x_train[num])

# Побудова моделі
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))

model.summary()

# Компіляція та тренування моделі
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
EPOCHS = 10
```

```
model.fit(X_train, y_train, batch_size=128, epochs=EPOCHS, verbose=1)

# Оцінка моделі
score = model.evaluate(X_test, y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Передбачення класів
predict_x = model.predict(X_test)
predicted_classes = np.argmax(predict_x, axis=1)

# Візуалізація правильних передбачень
correct_indices = np.nonzero(predicted_classes == np.argmax(y_test,
axis=1))[0]
plt.figure()
for i, correct in enumerate(correct_indices[:9]):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X_test[correct].reshape(28, 28), cmap='gray',
interpolation='none')
    plt.title("Pred {}, Class {}".format(predicted_classes[correct],
np.argmax(y_test[correct])))
plt.tight_layout()

# Візуалізація неправильних передбачень
incorrect_indices = np.nonzero(predicted_classes != np.argmax(y_test,
axis=1))[0]
plt.figure()
for i, incorrect in enumerate(incorrect_indices[:9]):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X_test[incorrect].reshape(28, 28), cmap='gray',
interpolation='none')
    plt.title("Pred {}, Class {}".format(predicted_classes[incorrect],
np.argmax(y_test[incorrect])))
plt.tight_layout()
```

Приклад виводу:

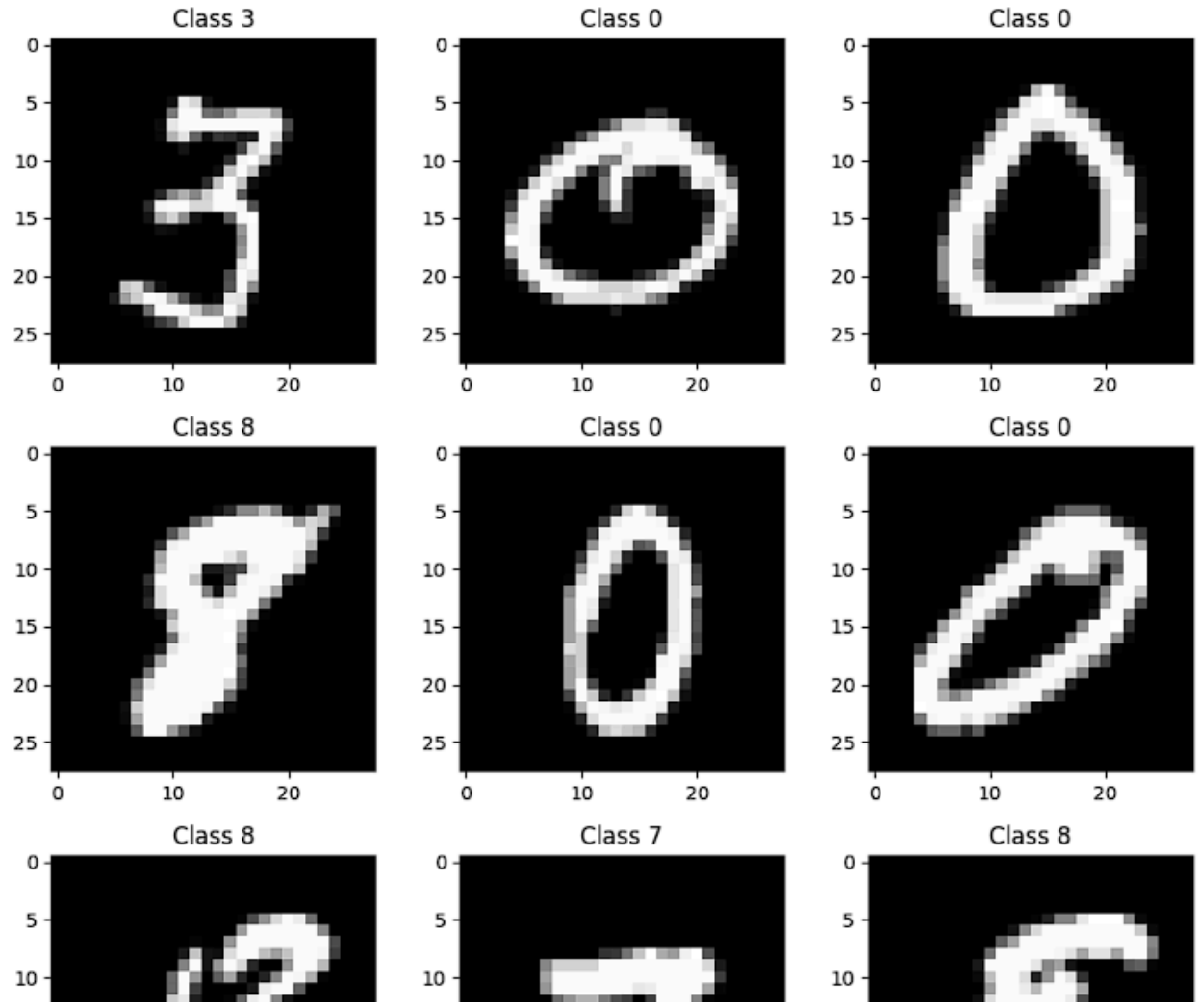
```
X_train: (60000, 784)
y_train: (60000, 10)
X_test: (10000, 784)
y_test: (10000, 10)
```



```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 512)	401,920
activation_6 (Activation)	(None, 512)	0
dropout_4 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 512)	262,656
activation_7 (Activation)	(None, 512)	0
dropout_5 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 10)	5,130
activation_8 (Activation)	(None, 10)	0

```
 Total params: 669,706 (2.55 MB)
 Trainable params: 669,706 (2.55 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.8613 - loss: 0.4574
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.9669 - loss: 0.1094
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9759 - loss: 0.0724
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9834 - loss: 0.0522
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.9864 - loss: 0.0407
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9867 - loss: 0.0390
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9880 - loss: 0.0346
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9906 - loss: 0.0266
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.9923 - loss: 0.0256
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9930 - loss: 0.0203
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9763 - loss: 0.0963
Test score: 0.08142556995153427
Test accuracy: 0.9785000085830688
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```


Class 3
Class 0
Class 0
Class 8
Class 0
Class 0
Class 8
Class 7
Class 8