Report 2 – Machine Learning

Galer Gabriel 442G

# Laboratory 2

## Exercise 7

import numpy as np

import keras

from keras import initializers

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import accuracy\_score

def define\_and\_model():

    model = Sequential()

    model.add(Dense(1, input\_dim=2, kernel\_initializer='zeros', bias\_initializer=initializers.Constant(-3), activation='sigmoid'))

    model.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer='adam')

    return model

def define\_xor\_model():

    model = Sequential()

    weights\_l1 = np.array([

        [2, -1],

        [2, -1]

    ])

    bias\_l1 = np.array([

        -1,

        1.5

    ])

    weights\_l2 = np.array([

        1,

        1

    ]).reshape(2, 1)

    bias\_l2 = np.array([

        -1.5

    ])

    model.add(Dense(2, input\_dim=2, activation='relu', weights=[weights\_l1, bias\_l1]))

    model.add(Dense(1, input\_dim=2, activation='sigmoid', weights=[weights\_l2, bias\_l2]))

    model.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer='adam')

    return model

def train\_and():

    model = define\_and\_model()

    x = np.array([[0,0],[0,1],[1,0],[1,1]], "uint8")

    y = np.array([0, 0, 0, 1], "uint8")

    model.fit(x, y, epochs=2000, verbose=0)

    predictions = model.predict(x)

    print(predictions)

def train\_xor():

    model = define\_xor\_model()

    x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], "uint8")

    y = np.array([0, 1, 1, 0], "uint8")

    model.fit(x, y, epochs=2000, verbose=0)

    predictions = model.predict(x)

    print(predictions)

if \_\_name\_\_ == '\_\_main\_\_':

    train\_xor()

## Exercise 8

import numpy as np

import keras

from keras.models import Sequential

from keras.layers import Dense

import pandas as pd

from sklearn.metrics import accuracy\_score

# TODO - Application 3 - Step 5 - Create the ANN model

def modelDefinition():

    # TODO - Application 3 - Step 5a - Define the model as a Sequential model

    model = Sequential()

    # TODO - Application 3 - Step 5b - Add a Dense layer with 13 neurons to the model

    model.add(Dense(8, input\_dim=13, kernel\_initializer='normal', activation='relu'))

    # TODO - Application 3 - Step 5c - Add a Dense layer (output layer) with 1 neuron

    model.add(Dense(1, kernel\_initializer='normal'))

    # TODO - Application 3 - Step 5d - Compile the model by choosing the optimizer(adam) ant the loss function (MSE)

    model.compile(loss='mean\_squared\_error', optimizer='adam')

    return model

def main():

    # TODO - Application 3 - Step 1 - Read data from "Houses.csv" file

    data = pd.read\_csv('./Houses.csv').values

    print(data)

    # TODO - Application 3 - Step 2 - Shuffle the data

    np.random.shuffle(data)

    # TODO - Application 3 - Step 3 - Separate the data from the labels (x\_data / y\_data)

    x\_data = data[:, :-1]

    y\_data = data[:, -1]

    print(x\_data.shape)

    print(y\_data.shape)

    # TODO - Application 3 - Step 4 - Separate the data into training/testing dataset

    eigthy\_percent = int(0.8 \* len(x\_data))

    x\_train = x\_data[:eigthy\_percent, :]

    y\_train = y\_data[:eigthy\_percent]

    print(x\_train.shape)

    print(y\_train.shape)

    x\_test = x\_data[eigthy\_percent:, :]

    y\_test = y\_data[eigthy\_percent:]

    # TODO - Application 3 - Step 5 - Call the function "modelDefinition"

    model = modelDefinition()

    # TODO - Application 3 - Step 6 - Train the model for 100 epochs and a batch of 16 samples

    model.fit(x\_train, y\_train, epochs=100, batch\_size=16, verbose=2)

    # Step 7

    predictions = model.predict(x\_test)

    # TODO - Exercise 8 - Compute the MSE for the test data

    mse = 1/len(x\_test) \* np.sum(np.square(predictions - y\_test))

    print(f"Mean Square Error = {mse}") # Mean Square Error = 16109.594866189918

    return

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Mean Square Error = 11270.85

## Exercise 9

Because the training data is shuffled.

## Exercise 10

import numpy as np

import keras

from keras.models import Sequential

from keras.layers import Dense

import pandas as pd

from sklearn.metrics import accuracy\_score

# TODO - Application 3 - Step 5 - Create the ANN model

def modelDefinition():

    # TODO - Application 3 - Step 5a - Define the model as a Sequential model

    model = Sequential()

    # TODO - Application 3 - Step 5b - Add a Dense layer with 13 neurons to the model

    model.add(Dense(8, input\_dim=13, kernel\_initializer='normal', activation='relu'))

    model.add(Dense(16, input\_dim=8, kernel\_initializer='normal', activation='relu'))

    # TODO - Application 3 - Step 5c - Add a Dense layer (output layer) with 1 neuron

    model.add(Dense(1, kernel\_initializer='normal'))

    # TODO - Application 3 - Step 5d - Compile the model by choosing the optimizer(adam) ant the loss function (MSE)

    model.compile(loss='mean\_squared\_error', optimizer='adam')

    return model

def main():

    # TODO - Application 3 - Step 1 - Read data from "Houses.csv" file

    data = pd.read\_csv('./Houses.csv').values

    print(data)

    # TODO - Application 3 - Step 2 - Shuffle the data

    np.random.shuffle(data)

    # TODO - Application 3 - Step 3 - Separate the data from the labels (x\_data / y\_data)

    x\_data = data[:, :-1]

    y\_data = data[:, -1]

    print(x\_data.shape)

    print(y\_data.shape)

    # TODO - Application 3 - Step 4 - Separate the data into training/testing dataset

    eigthy\_percent = int(0.8 \* len(x\_data))

    x\_train = x\_data[:eigthy\_percent, :]

    y\_train = y\_data[:eigthy\_percent]

    print(x\_train.shape)

    print(y\_train.shape)

    x\_test = x\_data[eigthy\_percent:, :]

    y\_test = y\_data[eigthy\_percent:]

    # TODO - Application 3 - Step 5 - Call the function "modelDefinition"

    model = modelDefinition()

    # TODO - Application 3 - Step 6 - Train the model for 100 epochs and a batch of 16 samples

    model.fit(x\_train, y\_train, epochs=100, batch\_size=16, verbose=2)

    # Step 7

    predictions = model.predict(x\_test)

    # TODO - Exercise 8 - Compute the MSE for the test data

    mse = 1/len(x\_test) \* np.sum(np.square(predictions - y\_test))

    print(f"Mean Square Error = {mse}") # Mean Square Error = 18099.76277140959

    return

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Mean Square Error = 14415.71

# Exercise 11

import numpy as np

import keras

from keras.models import Sequential

from keras.layers import Dense

import pandas as pd

from sklearn.metrics import accuracy\_score

# TODO - Application 3 - Step 5 - Create the ANN model

def modelDefinition():

    # TODO - Application 3 - Step 5a - Define the model as a Sequential model

    model = Sequential()

    # TODO - Application 3 - Step 5b - Add a Dense layer with 13 neurons to the model

    model.add(Dense(8, input\_dim=13, kernel\_initializer='normal', activation='relu'))

    model.add(Dense(16, input\_dim=8, kernel\_initializer='normal', activation='relu'))

    # TODO - Application 3 - Step 5c - Add a Dense layer (output layer) with 1 neuron

    model.add(Dense(1, kernel\_initializer='normal'))

    # TODO - Application 3 - Step 5d - Compile the model by choosing the optimizer(adam) ant the loss function (MSE)

    model.compile(loss='mean\_squared\_error', optimizer='adam')

    return model

def main():

    # TODO - Application 3 - Step 1 - Read data from "Houses.csv" file

    data = pd.read\_csv('./Houses.csv').values

    # TODO - Application 3 - Step 2 - Shuffle the data

    np.random.shuffle(data)

    # TODO - Application 3 - Step 3 - Separate the data from the labels (x\_data / y\_data)

    x\_data = data[:, [i for i in range(14) if i != 4]]

    y\_data = data[:, 4]

    print(x\_data.shape)

    print(y\_data.shape)

    # TODO - Application 3 - Step 4 - Separate the data into training/testing dataset

    eigthy\_percent = int(0.8 \* len(x\_data))

    x\_train = x\_data[:eigthy\_percent, :]

    y\_train = y\_data[:eigthy\_percent]

    print(x\_train.shape)

    print(y\_train.shape)

    x\_test = x\_data[eigthy\_percent:, :]

    y\_test = y\_data[eigthy\_percent:]

    # TODO - Application 3 - Step 5 - Call the function "modelDefinition"

    model = modelDefinition()

    # TODO - Application 3 - Step 6 - Train the model for 100 epochs and a batch of 16 samples

    model.fit(x\_train, y\_train, epochs=100, batch\_size=16, verbose=2)

    # Step 7

    predictions = model.predict(x\_test)

    # TODO - Exercise 8 - Compute the MSE for the test data

    mse = 1/len(x\_test) \* np.sum(np.square(predictions - y\_test))

    print(f"Mean Square Error = {mse}") # Mean Square Error = 18099.76277140959

    return

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Mean Square Error = 2.48

# Laboratory 3

## Exercise 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No of neurons | 8 | 16 | 32 | 64 | 128 |
| Accuracy | 0.10 | 0.10 | 0.14 | 0.19 | 0.10 |

## Exercise 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Batch size | 32 | 64 | 128 | 256 | 512 |
| Accuracy | 0.10 | 0.13 | 0.10 | 0.09 | 0.10 |

## Exercise 3

## Exercise 4

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense

from keras.utils import np\_utils

from keras.layers import Dropout

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from os import path

def baseline\_model(num\_pixels, num\_classes):

    model = Sequential()

    model.add(

        Dense(8, input\_dim=num\_pixels, kernel\_initializer="normal", activation="relu")

    )

    model.add(Dense(num\_classes, kernel\_initializer="normal"))

    model.compile(

        loss="categorical\_crossentropy", optimizer="adam", metrics=["mean\_squared\_error", "accuracy"]

    )

    return model

def trainAndPredictMLP(X\_train, Y\_train, X\_test, Y\_test):

    \_weights\_path = './weights\_ex4.h5'

    num\_pixels = X\_train.shape[1] \* X\_train.shape[2]

    X\_train = X\_train.reshape((X\_train.shape[0], num\_pixels)).astype("float32")

    X\_test = X\_test.reshape((X\_test.shape[0], num\_pixels)).astype("float32")

    X\_train = X\_train / 255

    X\_test = X\_test / 255

    Y\_train = np\_utils.to\_categorical(Y\_train)

    Y\_test = np\_utils.to\_categorical(Y\_test)

    num\_classes = Y\_test.shape[1]

    model = baseline\_model(num\_pixels, num\_classes)

    if path.exists(\_weights\_path):

        model.load\_weights(\_weights\_path)

    else:

        model.fit(

            X\_train,

            Y\_train,

            validation\_data=(X\_test, Y\_test),

            epochs=10,

            batch\_size=200,

            verbose=2,

        )

        model.save\_weights(\_weights\_path)

    scores = model.evaluate(X\_test, Y\_test, verbose=0)

    print(f"{scores[1]:.2f}")

    return

def main():

    (X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

    trainAndPredictMLP(X\_train, Y\_train, X\_test, Y\_test)

    return

if \_\_name\_\_ == "\_\_main\_\_":

    main()

## Exercise 5

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense

from keras.utils import np\_utils

from keras.layers import Dropout

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from os import path

import numpy as np

def baseline\_model(num\_pixels, num\_classes):

    model = Sequential()

    model.add(

        Dense(8, input\_dim=num\_pixels, kernel\_initializer="normal", activation="relu")

    )

    model.add(Dense(num\_classes, kernel\_initializer="normal"))

    model.compile(

        loss="categorical\_crossentropy", optimizer="adam", metrics=["mean\_squared\_error", "accuracy"]

    )

    return model

def trainAndPredictMLP(X\_train, Y\_train, X\_test, Y\_test):

    \_weights\_path = './weights\_ex4.h5'

    num\_pixels = X\_train.shape[1] \* X\_train.shape[2]

    X\_train = X\_train.reshape((X\_train.shape[0], num\_pixels)).astype("float32")

    X\_test = X\_test.reshape((X\_test.shape[0], num\_pixels)).astype("float32")

    X\_train = X\_train / 255

    X\_test = X\_test / 255

    Y\_train = np\_utils.to\_categorical(Y\_train)

    Y\_test = np\_utils.to\_categorical(Y\_test)

    num\_classes = Y\_test.shape[1]

    model = baseline\_model(num\_pixels, num\_classes)

    if path.exists(\_weights\_path):

        model.load\_weights(\_weights\_path)

    else:

        model.fit(

            X\_train,

            Y\_train,

            validation\_data=(X\_test, Y\_test),

            epochs=10,

            batch\_size=200,

            verbose=2,

        )

        model.save\_weights(\_weights\_path)

    scores = [np.argmax(score) for score in model.predict(X\_test[:5])]

    print(f"{scores}")

    return

def main():

    (X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

    trainAndPredictMLP(X\_train, Y\_train, X\_test, Y\_test)

    return

if \_\_name\_\_ == "\_\_main\_\_":

    main()

## Exercise 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kernel size | 1x1 | 3x3 | 5x5 | 7x7 | 9x9 |
| Accuracy | 0.97 | 0.99 | … | … | … |

## Exercise 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Neurons | 16 | 64 | 128 | 7x7 | 9x9 |
| Accuracy | 0.97 | 0.99 | … | … | … |

## Exercise 8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epochs | 1 | 2 | 5 | 10 | 20 |
| Accuracy | 0.96 | 0.97 | 0.98 | 0.98 | 0.99 |

## Exercise 9

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense

from keras.utils import np\_utils

from keras.layers import Dropout

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

def CNN\_model(input\_shape, num\_classes):

    # Application 2 - Step 6 - Initialize the sequential model

    model = Sequential()

    # TODO - Application 2 - Step 6 - Create the first hidden layer as a convolutional layer

    model.add(Conv2D(8, 3, activation="relu"))

    # TODO - Application 2 - Step 6 - Define the pooling layer

    model.add(MaxPooling2D())

    # TODO - Application 2 - Step 6 - Define the Dropout layer

    model.add(Dropout(0.2))

    # TODO - Application 2 - Step 6 - Define the flatten layer

    model.add(Flatten())

    # TODO - Application 2 - Step 6 - Define a dense layer of size 128

    model.add(Dense(128, activation="relu"))

    # TODO - Application 2 - Step 6 - Define the output layer

    model.add(Dense(num\_classes, activation="softmax"))

    # TODO - Application 2 - Step 7 - Compile the model

    model.compile(

        loss="categorical\_crossentropy", optimizer="adam", metrics=["accuracy"]

    )

    return model

def trainAndPredictCNN(X\_train, Y\_train, X\_test, Y\_test):

    # TODO - Application 2 - Step 3 - reshape the data to be of size [samples][width][height][channels]

    X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1).astype("float32")

    X\_test = X\_test.reshape(X\_test.shape[0], 28, 28, 1).astype("float32")

    # TODO - Application 2 - Step 4 - normalize the input values

    X\_train = X\_train / 255

    X\_test = X\_test / 255

    # TODO - Application 2 - Step 5 - Transform the classes labels into a binary matrix

    Y\_train = np\_utils.to\_categorical(Y\_train)

    Y\_test = np\_utils.to\_categorical(Y\_test)

    num\_classes = Y\_test.shape[1]

    # Application 2 - Step 6 - Call the cnn\_model function

    model = CNN\_model((28, 28, 1), num\_classes)

    # TODO - Application 2 - Step 8 - Train the model

    model.fit(

        X\_train,

        Y\_train,

        validation\_data=(X\_test, Y\_test),

        epochs=20,

        batch\_size=200,

        verbose=2,

    )

    # TODO - Application 2 - Step 8 - System evaluation - compute and display the prediction error

    scores = model.evaluate(X\_test, Y\_test, verbose=2)

    print(f"{scores[1]:.2f}")

    return

def main():

    (X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

    trainAndPredictCNN(X\_train, Y\_train, X\_test, Y\_test)

    return

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Accuracy: 0.99