

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра штучного інтелекту

Дисципліна: “Штучні нейронні мережі: архітектура”

ЛАБОРАТОРНА РОБОТА 1

“ОЗНАЙОМЛЕННЯ З ВІЗУАЛЬНИМ СЕРЕДОВИЩЕМ ІМІТАЦІЙНОГО
МОДЕЛЮВАННЯ MATLAB. СТВОРЕННЯ НЕЙРОННОЇ МЕРЕЖІ З ПРЯМОЇ
ПЕРЕДАЧІ ІНФОРМАЦІЇ. АЛГОРИТМИ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ.”

Виконали ст. гр. ІТШІ-18-1:
Апраксін Антон Романович
Михно Євген Віталійович
Соколенко Дмитро Олександрович

Прийняла:
Чала О. С.
з оцінкою “_____”
“___” _____ 20__р

Харків 2021

1 ОЗНАЙОМЛЕННЯ З ВІЗУАЛЬНИМ СЕРЕДОВИЩЕМ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ MATLAB. СТВОРЕННЯ НЕЙРОННОЇ МЕРЕЖІ З ПРЯМОЇ ПЕРЕДАЧІ ІНФОРМАЦІЇ. АЛГОРИТМИ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

1.1 Мета роботи:

Ознайомлення із візуальним середовищем імітаційного моделювання MATLAB (Matrix laboratory). Освоєння методики створення нейронної мережі із прямою передачею даних. Освоєння різноманітних алгоритмів навчання нейронних мереж та моделювання їх у середовищі MATLAB.

1.2 Лістинг коду:

1.2.1 Оптимізатор CGF

```
class OptimizerCGF( OptimizerAbstract ):
    def __init__( self , learning_rate=0.001, decay=0., epsilon=1e-7,
max_update=10 ):
        self.learning_rate = learning_rate
        self.current_learning_rate = learning_rate
        self.decay = decay
        self.iterations = 0
        self.epsilon = epsilon
        self.max_update = max_update

    def pre_update_params( self ):
        if self.decay:
            self.current_learning_rate = self.learning_rate * (1. / (1.
+ self.decay * self.iterations))

    # Update parameters
    def update_params( self , layer ):
        def calc_beta( dweights , dweights_prev ):
            assert dweights.shape[0] == dweights.size
            assert dweights_prev.shape[0] == dweights_prev.size
            return dweights.dot(dweights) / (dweights_prev.dot(
dweights_prev) + 1)

        if not hasattr( layer , 'prev_dweights ' ):
            layer.prev_dweights = layer.dweights.copy()
            layer.prev_dbias = layer.dbias.copy()
            layer.weight_p = np.zeros_like(-layer.dweights)
```

```

        layer.bias_p = np.zeros_like(-layer.dbiases)

        weight_update = self.current_learning_rate * layer.weight_p
        biases_update = self.current_learning_rate * layer.bias_p

        beta_weights = np.array([calc_beta(dweight, dweight_prev) for (
dweight, dweight_prev) in
                                zip(layer.dweights.T, layer.
prev_dweights.T)])
        beta_biases = np.array([calc_beta(dweight, dweight_prev) for (
dweight, dweight_prev) in
                                zip(layer.dbiases.T, layer.prev_dbiances
.T)])

        layer.weight_p = -layer.dweights + beta_weights * layer.
weight_p
        layer.bias_p = -layer.dbiases + beta_biases * layer.bias_p

        layer.prev_dweights = layer.dweights.copy()
        layer.prev_dbiances = layer.dbiases.copy()

        layer.weights += weight_update # np.clip(weight_update, -self.
max_update, self.max_update)
        layer.biases += biases_update # np.clip(biases_update, -self.
max_update, self.max_update)

```

```

def post_update_params(self):
    self.iterations += 1

```

1.2.2 Оптимізатор GDM

```

class OptimizerGDM(OptimizerAbstract):
def __init__(self, learning_rate=0.001, decay=0., momentum=0.):
    self.learning_rate = learning_rate
    self.current_learning_rate = learning_rate
    self.decay = decay
    self.iterations = 0
    self.momentum = momentum

def pre_update_params(self):
    if self.decay:
        self.current_learning_rate = self.learning_rate * \
(1. / (1. + self.decay * self.

```

```
iterations))
```

```
def update_params(self, layer):
    if not hasattr(layer, 'weight_momentums'):
        layer.weight_momentums = np.zeros_like(layer.weights)
        layer.bias_momentums = np.zeros_like(layer.biases)

    weight_updates = \
        self.momentum * layer.weight_momentums + \
        (1.0 - self.momentum) * self.current_learning_rate * layer.
dweights
    layer.weight_momentums = weight_updates

    bias_updates = \
        self.momentum * layer.bias_momentums + \
        (1.0 - self.momentum) * self.current_learning_rate * layer.
dbiases
    layer.bias_momentums = bias_updates

    layer.weights -= weight_updates
    layer.biases -= bias_updates

def post_update_params(self):
    self.iterations += 1
```

1.2.3 Оптимізатор BFGS

```
class OptimizerBFGS(OptimizerAbstract):
    def __init__(self, learning_rate=0.001, decay=0., epsilon=1e-7)
:
        self.learning_rate = learning_rate
        self.current_learning_rate = learning_rate
        self.decay = decay
        self.iterations = 0
        self.epsilon = epsilon

    def pre_update_params(self):
        if self.decay:
            self.current_learning_rate = self.learning_rate * (1. /
(1. + self.decay * self.iterations))

        # Update parameters
    def update_params(self, layer):
```

```

I = np.eye(layer.weights.shape[1])
if not hasattr(layer, f'prev_weights'):
    layer.H = I
    layer.prev_weights = np.zeros_like(layer.weights)
    layer.prev_dweights = np.zeros_like(layer.dweights)

sk = layer.weights - layer.prev_weights
yk = layer.dweights - layer.prev_dweights

rho_inv = yk.T @ sk
rho = 1 / (rho_inv + 0.1)

A1 = (I - rho * (yk.T @ sk))
A2 = (I - rho * (sk.T @ yk))
left = A1 @ (layer.H @ A2)
layer.H = left + rho * (yk.T @ yk)

weight_update = self.current_learning_rate * -(layer.H @
layer.dweights.T)

layer.weights += weight_update.T

layer.prev_weights = layer.weights.copy()
layer.prev_dweights = layer.dweights.copy()

def post_update_params(self):
    self.iterations += 1

```

1.3 Результати виконання:

```
[theredrover@TheRedRoverLTP_NW_lab1]$ python lab_01.py
epoch: 0, loss: 1.268 lr: 0.001
epoch: 100, loss: 1.034 lr: 0.0009990109791306607
epoch: 200, loss: 0.958 lr: 0.0009980139522350524
epoch: 300, loss: 0.926 lr: 0.000997018913448788
epoch: 400, loss: 0.910 lr: 0.0009960258568312435
epoch: 500, loss: 0.901 lr: 0.0009950347764654374
epoch: 600, loss: 0.897 lr: 0.000994045666457917
epoch: 700, loss: 0.895 lr: 0.0009930585209386388
epoch: 800, loss: 0.894 lr: 0.0009920733340608538
epoch: 900, loss: 0.893 lr: 0.000991090100000991
epoch: 1000, loss: 0.892 lr: 0.0009901088129585442
epoch: 1100, loss: 0.892 lr: 0.000989129467155956
epoch: 1200, loss: 0.892 lr: 0.0009881520568385063
epoch: 1300, loss: 0.891 lr: 0.000987176576274198
epoch: 1400, loss: 0.891 lr: 0.0009862030197536466
epoch: 1500, loss: 0.891 lr: 0.0009852313815899665
epoch: 1600, loss: 0.890 lr: 0.0009842616561186626
epoch: 1700, loss: 0.890 lr: 0.000983293837697519
epoch: 1800, loss: 0.890 lr: 0.0009823279207064903
epoch: 1900, loss: 0.889 lr: 0.0009813638995475912
epoch: 2000, loss: 0.888 lr: 0.0009804017686447907
epoch: 2100, loss: 0.888 lr: 0.0009794415224439025
```

```
epoch: 14100, loss: 0.190 lr: 0.0008764318705685414
epoch: 14200, loss: 0.184 lr: 0.0008756644103713692
epoch: 14300, loss: 0.178 lr: 0.0008748982930734303
epoch: 14400, loss: 0.173 lr: 0.0008741335151531044
epoch: 14500, loss: 0.169 lr: 0.0008733700731010752
epoch: 14600, loss: 0.165 lr: 0.0008726079634202742
epoch: 14700, loss: 0.162 lr: 0.0008718471826258294
epoch: 14800, loss: 0.159 lr: 0.0008710877272450108
epoch: 14900, loss: 0.156 lr: 0.0008703295938171786
epoch: 15000, loss: 0.154 lr: 0.0008695727788937295
epoch: 15100, loss: 0.152 lr: 0.0008688172790380455
epoch: 15200, loss: 0.150 lr: 0.0008680630908254411
epoch: 15300, loss: 0.148 lr: 0.0008673102108431123
epoch: 15400, loss: 0.147 lr: 0.0008665586356900839
epoch: 15500, loss: 0.146 lr: 0.0008658083619771601
epoch: 15600, loss: 0.144 lr: 0.0008650593863268713
epoch: 15700, loss: 0.143 lr: 0.000864311705373426
epoch: 15800, loss: 0.142 lr: 0.0008635653157626576
epoch: 15900, loss: 0.141 lr: 0.0008628202141519772
epoch: 16000, loss: 0.140 lr: 0.0008620763972103208
epoch: 16100, loss: 0.139 lr: 0.0008613338616181018
epoch: 16200, loss: 0.138 lr: 0.0008605926040671605
epoch: 16300, loss: 0.137 lr: 0.000859852621260716
epoch: 16400, loss: 0.137 lr: 0.0008591139099133154
epoch: 16500, loss: 0.136 lr: 0.0008583764667507876
epoch: 16600, loss: 0.136 lr: 0.0008576402885101931
epoch: 16700, loss: 0.135 lr: 0.0008569053719397766
epoch: 16800, loss: 0.135 lr: 0.0008561717137989195
epoch: 16900, loss: 0.134 lr: 0.0008554393108580913
epoch: 17000, loss: 0.134 lr: 0.0008547081598988025
epoch: 17100, loss: 0.134 lr: 0.0008539782577135586
epoch: 17200, loss: 0.133 lr: 0.0008532496011058114
epoch: 17300, loss: 0.133 lr: 0.0008525221868899139
epoch: 17400, loss: 0.133 lr: 0.0008517960118910723
epoch: 17500, loss: 0.133 lr: 0.0008510710729453017
epoch: 17600, loss: 0.132 lr: 0.0008503473668993783
epoch: 17700, loss: 0.132 lr: 0.0008496248906107954
epoch: 17800, loss: 0.132 lr: 0.000848903640947716
epoch: 17900, loss: 0.132 lr: 0.0008481836147889296
epoch: 18000, loss: 0.132 lr: 0.0008474648090238052
epoch: 18100, loss: 0.131 lr: 0.0008467472205522486
epoch: 18200, loss: 0.131 lr: 0.0008460308462846555
epoch: 18300, loss: 0.131 lr: 0.0008453156831418694
epoch: 18400, loss: 0.131 lr: 0.0008446017280551355
epoch: 18500, loss: 0.131 lr: 0.0008438889779660588
epoch: 18600, loss: 0.131 lr: 0.0008431774298265583
epoch: 18700, loss: 0.130 lr: 0.0008424670805988256
epoch: 18800, loss: 0.130 lr: 0.0008417579272552799
epoch: 18900, loss: 0.130 lr: 0.0008410499667785263
epoch: 19000, loss: 0.130 lr: 0.0008403431961613123
epoch: 19100, loss: 0.130 lr: 0.0008396376124064854
epoch: 19200, loss: 0.129 lr: 0.0008389332125269506
epoch: 19300, loss: 0.129 lr: 0.0008382299935456291
epoch: 19400, loss: 0.129 lr: 0.0008375279524954145
epoch: 19500, loss: 0.129 lr: 0.0008368270864191332
epoch: 19600, loss: 0.129 lr: 0.0008361273923695014
epoch: 19700, loss: 0.129 lr: 0.0008354288674090845
epoch: 19800, loss: 0.129 lr: 0.0008347315086102555
epoch: 19900, loss: 0.128 lr: 0.0008340353130551548
--- 403.47551679611206 seconds ---
```

Рис. 1: Показники початкових та кінцевих епох з використанням CGF

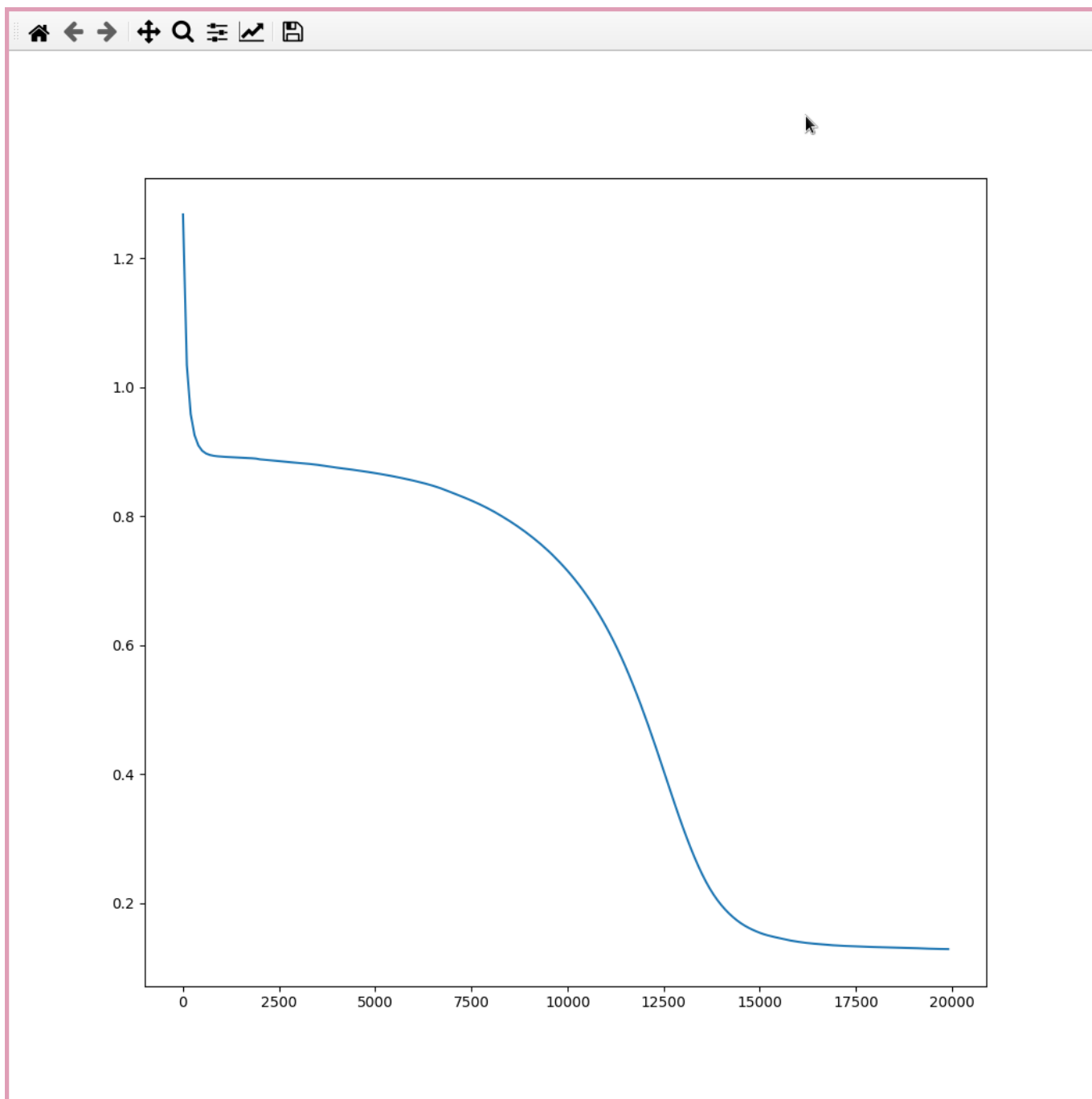


Рис. 2: Зниження помилки відносно епох з використанням CGF

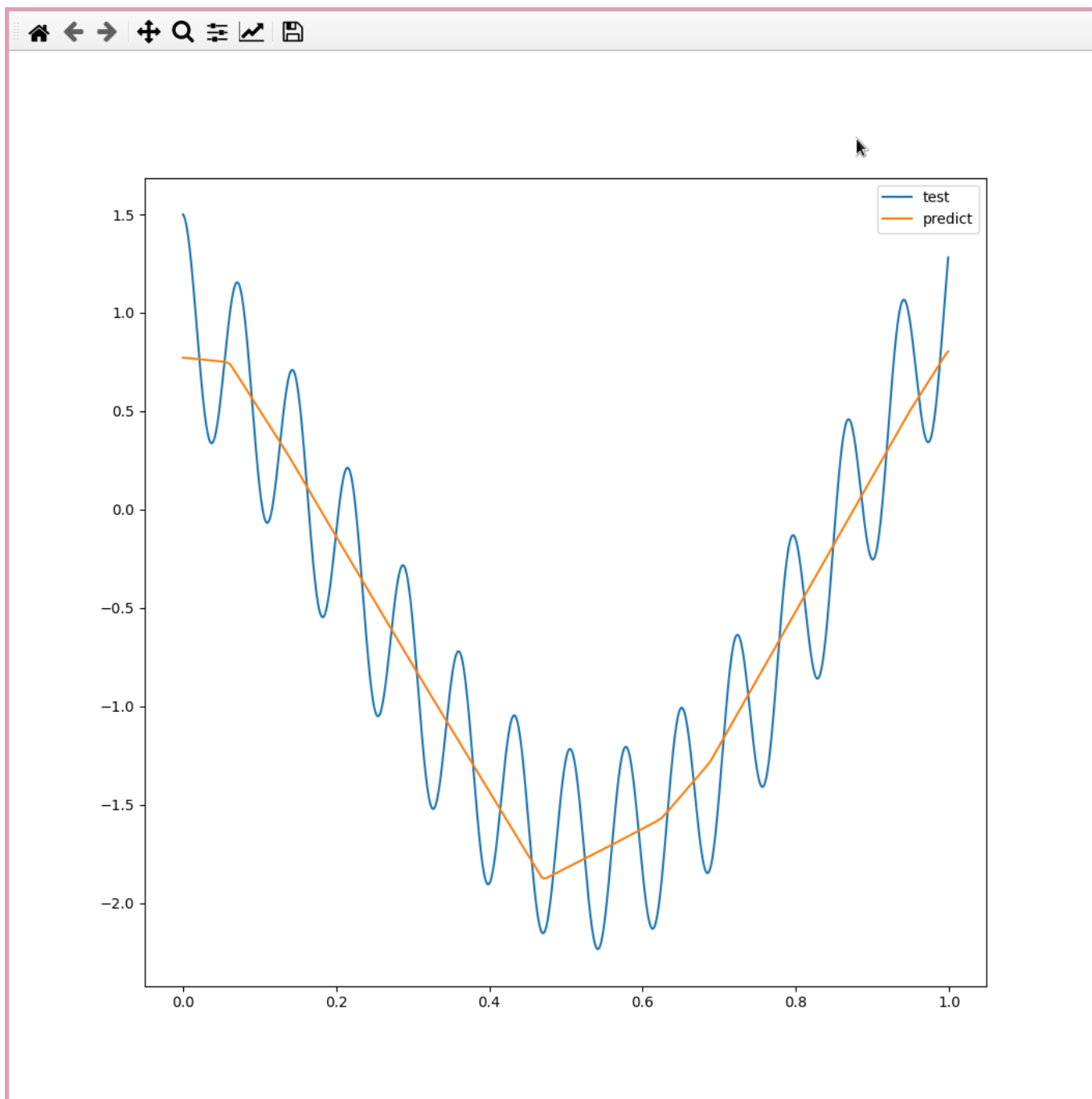


Рис. 3: Класифікація з використанням CGF


```
[theredrover@TheRedRoverLTP NN_lab1]$ python lab_01.py
epoch: 0, loss: 1.276 lr: 0.1
epoch: 100, loss: 0.881 lr: 0.09990109791306606
epoch: 200, loss: 0.550 lr: 0.09980139522350523
epoch: 300, loss: 0.122 lr: 0.09970189134487882
epoch: 400, loss: 0.122 lr: 0.09960258568312436
epoch: 500, loss: 0.122 lr: 0.09950347764654374
epoch: 600, loss: 0.122 lr: 0.09940456664579173
epoch: 700, loss: 0.122 lr: 0.09930585209386389
epoch: 800, loss: 0.122 lr: 0.0992073334060854
epoch: 900, loss: 0.121 lr: 0.09910901000009911
epoch: 1000, loss: 0.121 lr: 0.09901088129585443
epoch: 1100, loss: 0.121 lr: 0.0989129467155956
epoch: 1200, loss: 0.121 lr: 0.09881520568385065
epoch: 1300, loss: 0.121 lr: 0.09871765762741981
epoch: 1400, loss: 0.121 lr: 0.09862030197536466
epoch: 1500, loss: 0.121 lr: 0.09852313815899665
epoch: 1600, loss: 0.121 lr: 0.09842616561186628
epoch: 1700, loss: 0.121 lr: 0.09832938376975192
epoch: 1800, loss: 0.121 lr: 0.09823279207064904
epoch: 1900, loss: 0.121 lr: 0.09813638995475912
epoch: 2000, loss: 0.121 lr: 0.09804017686447908
epoch: 2100, loss: 0.121 lr: 0.09794415224439025
```

```
epoch: 14100, loss: 0.111 lr: 0.08764318705685414
epoch: 14200, loss: 0.115 lr: 0.08756644103713693
epoch: 14300, loss: 0.114 lr: 0.08748982930734303
epoch: 14400, loss: 0.111 lr: 0.08741335151531045
epoch: 14500, loss: 0.116 lr: 0.08733700731010752
epoch: 14600, loss: 0.111 lr: 0.08726079634202742
epoch: 14700, loss: 0.109 lr: 0.08718471826258295
epoch: 14800, loss: 0.116 lr: 0.08710877272450109
epoch: 14900, loss: 0.111 lr: 0.08703295938171786
epoch: 15000, loss: 0.109 lr: 0.08695727788937295
epoch: 15100, loss: 0.107 lr: 0.08688172790380455
epoch: 15200, loss: 0.117 lr: 0.08680630908254411
epoch: 15300, loss: 0.108 lr: 0.08673102108431124
epoch: 15400, loss: 0.109 lr: 0.0866558635690084
epoch: 15500, loss: 0.126 lr: 0.086580836197716
epoch: 15600, loss: 0.107 lr: 0.08650593863268713
epoch: 15700, loss: 0.106 lr: 0.0864311705373426
epoch: 15800, loss: 0.104 lr: 0.08635653157626577
epoch: 15900, loss: 0.111 lr: 0.08628202141519772
epoch: 16000, loss: 0.104 lr: 0.08620763972103207
epoch: 16100, loss: 0.103 lr: 0.08613338616181018
epoch: 16200, loss: 0.107 lr: 0.08605926040671606
epoch: 16300, loss: 0.105 lr: 0.0859852621260716
epoch: 16400, loss: 0.102 lr: 0.08591139099133155
epoch: 16500, loss: 0.101 lr: 0.08583764667507876
epoch: 16600, loss: 0.100 lr: 0.0857640288510193
epoch: 16700, loss: 0.102 lr: 0.08569053719397768
epoch: 16800, loss: 0.103 lr: 0.08561717137989194
epoch: 16900, loss: 0.130 lr: 0.08554393108580913
epoch: 17000, loss: 0.100 lr: 0.08547081598988025
epoch: 17100, loss: 0.124 lr: 0.08539782577135586
epoch: 17200, loss: 0.096 lr: 0.08532496011058115
epoch: 17300, loss: 0.104 lr: 0.0852522186889914
epoch: 17400, loss: 0.108 lr: 0.08517960118910722
epoch: 17500, loss: 0.094 lr: 0.08510710729453018
epoch: 17600, loss: 0.093 lr: 0.08503473668993783
epoch: 17700, loss: 0.092 lr: 0.08496248906107955
epoch: 17800, loss: 0.090 lr: 0.0848903640947716
epoch: 17900, loss: 0.091 lr: 0.08481836147889296
epoch: 18000, loss: 0.105 lr: 0.08474648090238052
epoch: 18100, loss: 0.135 lr: 0.08467472205522486
epoch: 18200, loss: 0.095 lr: 0.08460308462846555
epoch: 18300, loss: 0.086 lr: 0.08453156831418694
epoch: 18400, loss: 0.092 lr: 0.08446017280551356
epoch: 18500, loss: 0.084 lr: 0.08438889779660588
epoch: 18600, loss: 0.084 lr: 0.08431774298265583
epoch: 18700, loss: 0.100 lr: 0.08424670805988256
epoch: 18800, loss: 0.086 lr: 0.084175792725528
epoch: 18900, loss: 0.110 lr: 0.08410499667785264
epoch: 19000, loss: 0.091 lr: 0.08403431961613123
epoch: 19100, loss: 0.176 lr: 0.08396376124064854
epoch: 19200, loss: 0.099 lr: 0.08389332125269507
epoch: 19300, loss: 0.079 lr: 0.08382299935456292
epoch: 19400, loss: 0.075 lr: 0.08375279524954145
epoch: 19500, loss: 0.075 lr: 0.08368270864191332
epoch: 19600, loss: 0.075 lr: 0.08361273923695013
epoch: 19700, loss: 0.125 lr: 0.08354288674090846
epoch: 19800, loss: 0.072 lr: 0.08347315086102555
epoch: 19900, loss: 0.069 lr: 0.08340353130551548
--- 185.7106065750122 seconds ---
```

Рис. 4: Показники початкових та кінцевих епох з використанням GDM

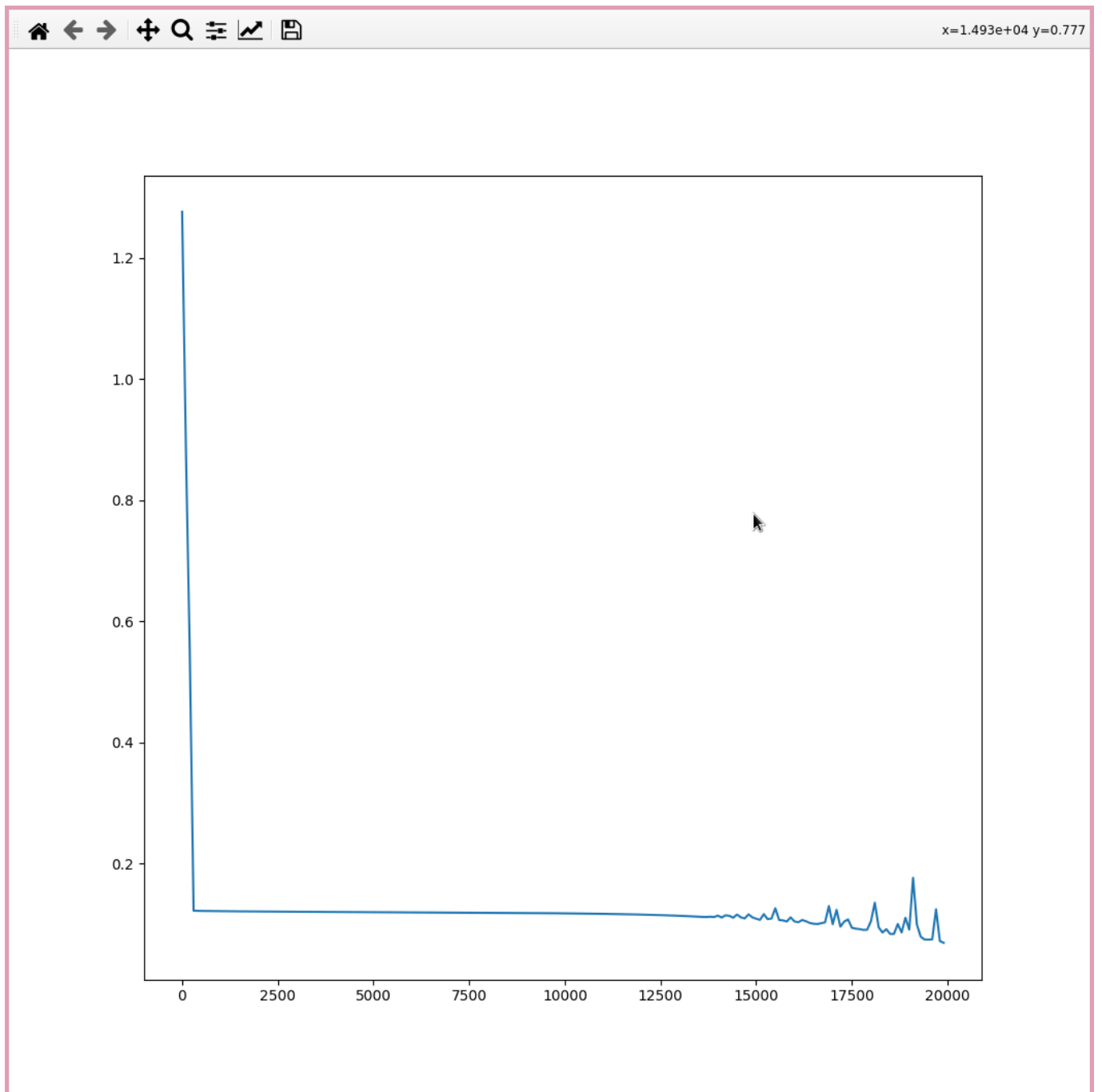


Рис. 5: Класифікація з використанням GDM

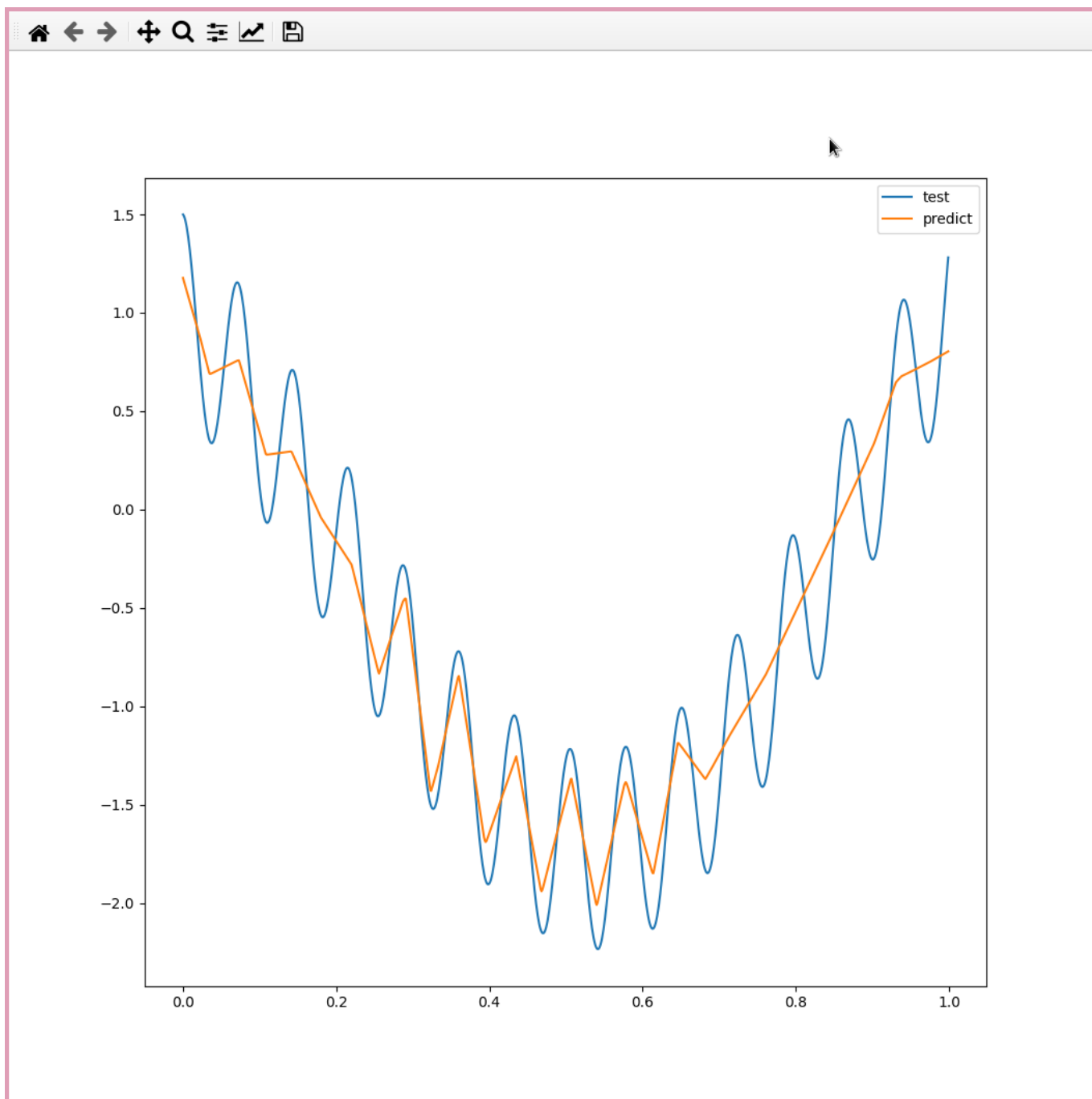


Рис. 6: Класифікація з використанням GDM

```

[theredrover@TheRedRoverLTP NN_lab1]$ python lab_01.py
epoch: 0, loss: 1.272 lr: 0.2
epoch: 100, loss: 0.923 lr: 0.19803940984255866
epoch: 200, loss: 0.923 lr: 0.19609765663300324
epoch: 300, loss: 0.923 lr: 0.1941936110301971
epoch: 400, loss: 0.923 lr: 0.19232618521011635
epoch: 500, loss: 0.923 lr: 0.1904943327935994
epoch: 600, loss: 0.923 lr: 0.18869704689121614
epoch: 700, loss: 0.923 lr: 0.1869333582577811
epoch: 800, loss: 0.923 lr: 0.18520233354940271
epoch: 900, loss: 0.923 lr: 0.18350307367648408
epoch: 1000, loss: 0.923 lr: 0.18183471224656786
epoch: 1100, loss: 0.923 lr: 0.18019641409135956
epoch: 1200, loss: 0.923 lr: 0.1785873738726672
epoch: 1300, loss: 0.923 lr: 0.17700681476236835
epoch: 1400, loss: 0.923 lr: 0.17545398719185895
epoch: 1500, loss: 0.923 lr: 0.17392816766675367
epoch: 1600, loss: 0.923 lr: 0.17242865764290027
epoch: 1700, loss: 0.923 lr: 0.17095478246003934
epoch: 1800, loss: 0.923 lr: 0.169505890329689
epoch: 1900, loss: 0.923 lr: 0.16808135137406507
epoch: 2000, loss: 0.923 lr: 0.16668055671305945
epoch: 2100, loss: 0.923 lr: 0.1653029175964956
epoch: 2200, loss: 0.923 lr: 0.16394786457906385
epoch: 2300, loss: 0.923 lr: 0.16261484673550697
epoch: 2400, loss: 0.923 lr: 0.16130333091378338
epoch: 2500, loss: 0.923 lr: 0.16001280102408194
epoch: 2600, loss: 0.923 lr: 0.15874275736169538
^CTraceback (most recent call last):
  File "/home/theredrover/projects/python/NN_lab1/lab_01.py", line 49, in <module>
    neural_network.fit(X, y, epochs=20000)
  File "/home/theredrover/projects/python/NN_lab1/ai/neural_network.py", line 24, in fit
    predictions = self.forward(X)
  File "/home/theredrover/projects/python/NN_lab1/ai/neural_network.py", line 48, in forward
    layer.forward(x)
  File "/home/theredrover/projects/python/NN_lab1/ai/layers/layer_dense.py", line 26, in forward
    self.output = np.dot(inputs, self.weights) + self.biases
  File "<__array_function__ internals>", line 5, in dot
KeyboardInterrupt

[theredrover@TheRedRoverLTP NN_lab1]$ █

```

Рис. 7: Невдача при створенні алгоритму BFGS

Висновки:

В ході виконання лабораторної роботи ми ознайомилися із принципами створення нейронної мережі з прямою передачею інформації. За основу був взят код із книжки [The Neural Networks from Scratch](#). Нами були дописані необхідні алгоритми оптимізації, що використовуються всередині мереж.

Алгоритм GDM показав найкращі результати щодо швидкості та точності виконання, але, можливо, це є результатом того, що інші алгоритми вийшли неоптимізованими, та в нас є сумніви щодо правильності їх реалізації.