визначена дальність прогнозу, розумно буде спеціально навчити мережу саме на таку дальність.

У пакеті Statistica Neural Networks для рішення задач прогнозу часових рядів можна застосовувати мережі всіх типів (тип мережі повинний підходити, у залежності від задачі, для регресії чи класифікації). Мережа конфігурується для прогнозу часового ряду установкою параметрів Часове вікно – Steps та Обрій – Lookahead.

Параметр Часове вікно задає число попередніх значень, які варто подавати на вхід, а параметр Обрій указує, як далеко потрібно будувати прогноз. Кількість вхідних і вихідних змінних може бути довільною. Однак, найчастіше в якості вхідної й одночасно (з урахуванням обрію) вихідної виступає єдина змінна.

При конфігуруванні мережі для аналізу часових рядів змінюється метод передобробки даних (витягаються не окремі спостереження, а їхні блоки), але навчання і робота мережі відбуваються точно так само, як і в задачах інших типів.

6.3 Моделювання нейронних мереж засобами бібліотек мови Python

Мова *Python* є однією з найбільш популярних мов програмування в останнє десятиріччя. Завдяки безкоштовним засобам розробки та потужним бібліотекам Python знаходить широке використання при розробці прикладного програмного забезпечення та вирішенні наукових задач. Зокрема велика кількість потужних бібліотек Python для моделювання нейронних мереж та розпізнавання образів дозволяє вирішувати задачі штучного інтелекту та інтелектуального аналізу даних.

Бібліотека NeuroLab (https://pythonhosted.org/neurolab/) – це бібліотека для Python, що містить моделі та методи навчання нейромереж з інтерфейсом, подібним до Neural Network Toolbox пакету MATLAB.

Інсталяція бібліотеки здійснюється одним зі способів:

– використовуючи setuptools:

```
easy_install neurolab
```

– використовуючи pip:

```
pip install neurolab
```

– використовуючи python:

```
python setup.py install
```

Модуль net – модуль, що містить базові архітектури нейромереж.

Тип мережі	Функція створення	Кількість шарів	Підтримка функцій навчання	Функція помилки
Одношаровий персептрон	newp	1	train_delta	SSE
Багатошаровий персептрон (БНМ)	newff	≥ 1	train_gd, train_gdm, train_gda, train_gdx, train_rprop, train_bfgs, train_cg	SSE
Конкуруючий шар (SOM)	newc	1	train_wta, train_cwta*	SAE
LVQ	newlvq	2	train_lvq	MSE
Мережа Елмана	newelm	≥ 1	train_gdx	MSE
Мережа Хопфілда	newhop	1	–	–
Мережа Хеммінга	newhem	2	–	–

`neurolab.net.newp(minmax, cn, transf)`– створення одношарового персептрона. Параметри: minmax – список списків, які містять мінімальне та максимальне значення відповідної вхідної ознаки, cn – кількість вихідних нейронів, transf – функція активації. Повертає мережу net.

```
>>> # створення мережі з двома входами та 10 нейронами
>>> net = newp([[-1, 1], [-1, 1]], 10)
```

`neurolab.net.newff(minmax, size, transf=None)` – створення багатошарового персептрона (БНМ). Параметри: minmax – список списків, які містять мінімальне та максимальне значення відповідної вхідної ознаки, size – перелік, довжина якого відповідає кількості шарів, не враховуючи вхідний шар, елементи якого відповідають кількості нейронів відповідного шару, transf – функція активації. Повертає мережу net.

```
>>> # створення нейромережі з двома входами
>>> # діапазон для кожного входу: [-0.5, 0.5]
>>> # 3 нейрони у прихованому шарі, 1 вихідний нейрон
>>> # 2 шари, включаючи прихований та вихідний шари
>>> net = newff([[-0.5, 0.5], [-0.5, 0.5]], [3, 1])
```

```
>>> net.ci
2
>>> net.co
1
>>> len(net.layers)
2
```

`neurolab.net.newhop(target, transf=None, max_init=10, delta=0)` – створює рекурентну мережу Хопфілда. Параметри: target – масив цільових шаблонів, transf – функція активації, max_init – максимальна кількість рекурентних ітерацій, delta – мінімальна різниця між двома виходами для зупинення рекурентного циклу. Повертає нейромережу net.

```
>>> net = newhop([[-1, -1, -1], [1, -1, 1]])
>>> output = net.sim([[-1, 1, -1], [1, -1, 1]])
```

`neurolab.net.newhem(target, transf=None, max_iter=10, delta=0)` – створення рекурентної мережі Хеммінга з двома шарами. Параметри: target – масив з навчальними шаблонами, transf – функція активації вхідного шару, max_init – максимальна кількість рекурентних ітерацій, delta – мінімальна різниця між двома виходами для зупинення рекурентного циклу. Повертає мережу net.

```
>>> net = newhem([[-1, -1, -1], [1, -1, 1]])
>>> output = net.sim([[-1, 1, -1], [1, -1, 1]])
```

`neurolab.net.newelm(minmax, size, transf=None)` – створення рекурентної мережі Елмана. Параметри: minmax – перелік мінімальних та максимальних значень для входів, size – список довжиною, що дорівнює кількостів шарів, без урахування вхідного шару, елементами якого є кількості нейронів відповідного шару. Повертає мережу net.

```
>>> # один вхід, діапазон входу [-1, 1],
>>> # один вихідний нейрон,
>>> # один шар, включаючи вихідний шар
>>> net = newelm([[-1, 1]], [1], [trans.PureLin()])
>>> # встановлення ваг для усіх вхідних нейронів в 1
>>> net.layers[0].np['w'][:] = 1
>>> # встановлення порогів усіх вхідних нейронів у 0
>>> net.layers[0].np['b'][:] = 0
>>> net.sim([[1], [1] ,[1], [3]])
array([[ 1.],
       [ 2.],
       [ 3.],
       [ 6.]])
```

`neurolab.net.newc(minmax, cn)` – створення конкурентного шару (мережі Кохонена). Параметри: minmax – перелік мінімальних та

максимальних значень для входів, cn – кількість вихідних нейронів. Повертає мережу net.

```
>>> # створення мережі з двома входами та 10 нейронами
>>> net = newc([[-1, 1], [-1, 1]], 10)
```

`neurolab.net.newlvq(minmax, cn0, pc)` – створює мережу LVQ. Параметри: minmax – список списків, які містять мінімальне та максимальне значення відповідної вхідної ознаки, cn0 – кількість нейронів у вхідному шарі, pc – список процентів, сума яких має дорівнювати одиниці. Повертає мережу net.

```
>>> # створення мережі з двома входами,
>>> # двома шарами та 10 нейронами у кожному шарі
>>> net = newlvq([[-1, 1], [-1, 1]], 10, [0.6, 0.4])
```

`trans` – функції активації нейронів.

`deriv(x, y)` – похідна функції активації.

```
>>> import numpy as np
>>> f = TanSig()
>>> x = np.linspace(-5,5,100)
>>> y = f(x)
>>> df_on_dy = f.deriv(x, y) # обчислення похідної
>>> f.out_minmax  # діапазон виходу [min, max]
[-1, 1]
>>> f.inp_active  # діапазон входу [min, max]
[-2, 2]
```

`class neurolab.trans.Competitive` – функція активації змагального шару. Параметри: x – вхідний масив. Повертає y – масив бінарних елементів: 1, якщо це – найменший елемент в x, 0 – інакше.

```
>>> f = Competitive()
>>> f([-5, -0.1, 0, 0.1, 100])
array([ 1.,  0.,  0.,  0.,  0.])
>>> f([-5, -0.1, 0, -6, 100])
array([ 0.,  0.,  0.,  1.,  0.])
```

`class neurolab.trans.HardLim` – порогова функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = HardLim()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([ 0.,  0.,  0.,  1.,  1.])
```

`class neurolab.trans.HardLims` – симетрична порогова функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = HardLims()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([-1., -1., -1.,  1.,  1.])
```

`class neurolab.trans.LogSig` – функція активації логарифмічний сигмоїд. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = LogSig()
>>> x = np.array([-np.Inf, 0.0, np.Inf])
>>> f(x).tolist()
[0.0, 0.5, 1.0]
```

`class neurolab.trans.PureLin` – лінійна функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> import numpy as np
>>> f = PureLin()
>>> x = np.array([-100., 50., 10., 40.])
>>> f(x).tolist()
[-100.0, 50.0, 10.0, 40.0]
```

`class neurolab.trans.SatLin` – насичена лінійна функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = SatLin()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([ 0. ,  0. ,  0. ,  0.1,  1. ])
```

`class neurolab.trans.SatLinPrm(k=1, out_min=0, out_max=1)` – лінійна функція активації з параметричним виходом. Параметри: k – дійсне число, out_min, out_max – мінімальне та максимальне значення виходу, x – вхідний масив. Повертає y – вихід.

```
>>> f = SatLinPrm()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([ 0. ,  0. ,  0. ,  0.1,  1. ])
>>> f = SatLinPrm(1, -1, 1)
>>> f(x)
array([-1. , -0.1,  0. ,  0.1,  1. ])
```

`class neurolab.trans.SatLins` – функція активації симетрична насичена лінійна.

```
>>> f = SatLins()
>>> x = np.array([-5, -1, 0, 0.1, 100])
>>> f(x)
array([-1. , -1. ,  0. ,  0.1,  1. ])
```

`neurolab.trans.SoftMax` – функція активації Soft max.

```
>>> from numpy import floor
>>> f = SoftMax()
>>> floor(f([0, 1, 0.5, -0.5]) * 10)
array([ 1.,  4.,  2.,  1.])
```

`neurolab.trans.TanSig` – сигмоїдна функція активації гіперболічний тангенс.

```
>>> f = TanSig()
>>> f([-np.Inf, 0.0, np.Inf])
array([-1.,  0.,  1.])
```

`init` – функції ініціалізації шарів мережі.

`neurolab.init.InitRand(minmax, init_prop)` – ініціалізація заданих властивостей шару випадковими числами у заданих обмеженнях.

`neurolab.init.init_rand(layer, min=-0.5, max=0.5, init_prop='w')` –ініціалізація заданої властивості шару випадковими числами у заданих обмеженнях.

`neurolab.init.init_zeros(layer)` – встановлення усіх властивостей шару layer у нуль.

`neurolab.init.initnw(layer)` – ініціалізація ваг шару layer методом Нгуена-Уідроу.

`neurolab.init.initwb_lin(layer)` – ініціалізація лінійного простору ваг та порогів layer невипадковими значеннями.

`neurolab.init.initwb_reg(layer)` – ініціалізація ваг та порогів шару layer у діапазоні, заданому активаційною функцією.

`neurolab.init.midpoint(layer)` – встановлення вагових коефіцієнтів шару layer у центрі діапазонів входів.

`train` – методи навчання нейромереж. Типові параметри: input – значення вхідних ознак розпізнаваних екземплярів, target – цільові значення вихідних ознак для розпізнаваних екземплярів, epochs – максимально припустима кількість циклів навчання, show – період відображення поточних результатів навчання, goal – значення цільової функції, при досягненні якого навчання зупиняється, lr – крок навчання.

Навчання одношарового персептрона.

`neurolab.train.train_delta`() – дельта-правило.

Градієнтні методи навчання БНМ. Типові параметри градієнтних методів (у доповнення до розглянутих вище параметрів train): adapt – тип навчання, rr – коефіцієнт регуляризації навчання, mc – константа моменту навчання, lr_inc – коефіцієнт збільшення швидкості навчання, lr_dec – коефіцієнт зменшення швидкості навчання, max_perf_inc – максимально припустиме збільшення цільової функції, rate_dec – зменшення зміни ваг, rate_inc – приріст зміни ваг, rate_min – мінімальне значення градієнта цільової функції, rate_max – максимальне значення зміни вагових коефіцієнтів.

`neurolab.train.train_gd()` – градієнтний спуск зі зворотним поширенням помилки.

`neurolab.train.train_gdm()` – градієнтний спуск з моментом зі зворотним поширенням помилки.

`neurolab.train.train_gda()` – градієнтний спуск з адаптивним навчальним кроком.

`neurolab.train.train_gdx()`– градієнтний спуск з моментом зі зворотним поширенням помилки та адаптивним навчальним кроком.

`neurolab.train.train_rprop()` – еластичне зворотне поширення помилки.

`neurolab.train.train_bfgs()` – метод Бройдена-Флетчера–Гольдфарба-Шанно.

`neurolab.train.train_cg()` – метод спряжених градієнтів.

`neurolab.train.train_ncg()` – метод спряжених градієнтів Ньютона.

Методи навчання на основі правила "переможець отримує усе" (Winner Take All).

`neurolab.train.train_wta()` – метод "переможець отримує усе" для мережі SOM. Параметри: input – значення вхідних ознак екземплярів вибірки, epochs – максимально припустима кількість епох, show – період друку у процесі навчання, goal –значення цільової функції, при якому навчання зупиняється.

`neurolab.train.train_cwta()` – совісливий метод "переможець отримує усе" для мережі SOM. Параметри: input – значення вхідних ознак екземплярів вибірки, epochs – максимально припустима кількість епох, show – період друку у процесі навчання, goal –значення цільової функції, при якому навчання зупиняється.

Методи навчання мереж LVQ.

`neurolab.train.train_lvq()` – метод навчання LVQ1.

Параметри: input – значення вхідних ознак екземплярів вибірки, epochs – максимально припустима кількість епох, show – період друку у процесі навчання, goal –значення цільової функції, при якому навчання зупиняється, lr – крок навчання, adapt – тип навчання.

`error` – функції помилки нейромережі.

`deriv(target, output)` – похідна функції помилки нейромережі.

```
>>> msef = MSE()
>>> x = np.array([[1.0, 0.0], [2.0, 0.0]])
>>> msef(x, 0)
1.25
>>> # calc derivative:
>>> msef.deriv(x[0], 0)
array([ 1.,  0.])
```

`class neurolab.error.CEE` – функція помилки кросентропія. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

`class neurolab.error.MAE` – функція помилки середнє модулів помилки. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

`class neurolab.error.MSE` – середньоквадратична функція помилки. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

```
>>> f = MSE()
>>> x = np.array([[1.0, 0.0], [2.0, 0.0]])
>>> f(x, 0)
1.25
>>> f = MSE()
>>> x = np.array([1.0, 0.0])
>>> # calc derivative:
>>> f.deriv(x, 0)
array([ 1.,  0.])
```

`class neurolab.error.SAE` – функція помилки суми модулів. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

`class neurolab.error.SSE` – функція сумарної квадратичної помилки. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

Бібліотека Keras (https://keras.io/) є потужним засобом для моделювання нейронних мереж мовою Python.

За замовчуванням Keras використовує бібліотеку Tensorflow для маніпуляції тензорами. Тому перед інсталяцією Keras потрібно встановити бібліотеку Tensorflow.

Інсталяція бібліотеки Keras здійснюється одним із двох способів:

– з PyPI (рекомендується):

```
sudo pip install keras
pip install keras
```

– з джерела GitHub:

```
git clone https://github.com/keras-team/keras.git
cd keras
sudo python setup.py install
```

Розглянемо найважливіші методи бібліотеки Keras.

Метод `compile(optimizer, loss=None, metrics=None, loss_weights=None, sample_weight_mode=None, weighted_metrics =None, target_tensors=None)` конфігурує модель для навчання. Як аргументи використовує: optimizer – рядок з назвою оптимізатора, loss – рядок з ім'ям функції втрат, metrics – список метрик, що обраховуються під час навчання та тестування моделі, loss_weights – необов'язковий перелік скалярних коефіцієнтів втрат, sample_weight_mode –режим визначення вагових коефіцієнтів, weighted_metrics – перелік метрик, що обраховуються та зважуються вагами вибірки або вагами класу під час навчання та тестування, target_tensors – визначення керованих параметрів навчання.

Метод `fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch= None, validation_steps=None, validation_freq=1, max_queue_size=10, workers=1, use_multiprocessing=False)` навчає модель для заданої кількості циклів навчання (епох – проходів вибірки). Як аргументи використовує: x – вхідні дані, y – цільові дані, batch_size – розмір пакету (кількість екземплярів на оновлення градієнта), epochs – кількість циклів навчання моделі на усій вибірці даних, verbose – режим відображення ходу навчання, callbacks – список викликів, validation_split – доля навчальних даних, що використовуються для перевірки моделі, validation_data – дані для обрахунку функції втрат та метрик наприкінці кожної епохи навчання, shuffle – регулювання перемішування навчальних даних перед кожною епохою навчання, class_weight – ваги класів при розрахунку функції втрат, sample_weight – ваги екземплярів, враховувані при визначенні функції втрат, initial_epoch – епоха, з якої починається навчання, steps_per_epoch – кількість пакетів (підвибірок) екземплярів, що обробляються на одній епосі, validation_steps – кількість пакетів (підвибірок) екземплярів для перевірки моделі перед зупиненням, validation_freq – кількість епох навчання, що виконуються до запуску перевірки, max_queue_size – максимальний розмір черги генератора, workers – максимальна кількість процесів, що можуть прискорюватися при використання потоків на основі процесів, use_multiprocessing – прапорець

використання потоків. Повертає об’єкт History, властивість якого History.history – це запис значень втрат навчання та метрик навчання на успішних епохах, а також значень втрат та метрик тестування моделі.

Метод `predict(x, batch_size=None, verbose=0, steps= None, callbacks=None, max_queue_size=10, workers=1, use_multiprocessing=False)` генерує передбачення (оцінку) виходу для вхідних екземплярів. Як аргументи використовує: x – вхідні дані, що розпізнаються, batch_size – розмір пакету на оновлення градієнта, verbose – прапорець пояснень, steps – кількість кроків (пакетів екземплярів) перед завершенням циклу оцінювання, callbacks – список зворотних викликів, max_queue_size – максимальний розмір черги генератора, workers – максимальна кількість процесів, use_multiprocessing – прапорець використання багатопроцесорних потоків. Повертає масив(и) Numpy, що містять передбачення (оцінки) виходу.

Розглянемо використання засобів бібліотеки на прикладах фрагментів тексту програм, пояснення до яких наведено у коментарях (джерело: https://www.datacamp.com/).

Базовий приклад.

```
# підключення бібліотек та імпорт їхніх об'єктів
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
# генерація випадкового масиву даних
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
# створення структури моделі нейромережі
>>> model = Sequential()
>>> model.add(Dense(32, activation='relu', input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
# компіляція
>>> model.compile(optimizer='rmsprop',
loss='binary_crossentropy', metrics=['accuracy'])
# навчання моделі
>>> model.fit(data,labels,epochs=10,batch_size=32)
# запуск емуляції моделі для набору даних
>>> predictions = model.predict(data)
```

Дані мають зберігатися як масиви NumPy або як списки масивів NumPy. Бажано первинну вибірку даних розбити на навчальну та тестові вибірки.

Завантаження даних з наборів Keras.

```
>>> from keras.datasets import boston_housing, mnist,
cifar10, imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) =
boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) =
cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) =
imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Завантаження даних з мережі Інтернет.

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Попередня обробка даних.

```
# Заповнення послідовності
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
# Кодування даних
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
# Формування навчальної та тестової вибірок даних
>>> from sklearn.model_selection import train_test_split
>>> X_train5, X_test5, y_train5, y_test5 =
train_test_split(X, y, test_size=0.33, random_state=42)
# Нормалізація даних
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Визначення архітектури моделі нейромережі:

```
# Послідовна модель
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

– багатошаровий персептрон:

```
# бінарна класифікація:
>>> from keras.layers import Dense
>>> model.add(Dense(12, input_dim=8, kernel_initializer=
'uniform', activation='relu'))
>>> model.add(Dense(8, kernel_initializer='uniform',
```

```
activation='relu'))
>>> model.add(Dense(1, kernel_initializer='uniform',
activation='sigmoid'))
# багатокласова класифікація:
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,
)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
# оцінювання:
>>> model.add(Dense(64, activation='relu',
input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

– згорткова нейромережа:

```
>>> from keras.layers import Activation, Conv2D,
MaxPooling2D, Flatten
>>> model2.add(Conv2D(32, (3,3), padding='same',
input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32, (3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64, (3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64, (3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

– рекурентна нейромережа:

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,
recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Огляд моделі.

```
# форма виходу моделі
>>> model.output_shape
# зведене подання моделі
```

```
>>> model.summary()
# конфігурація моделі
>>> model.get_config()
# перелік усіх тензорів ваг моделі
>>> model.get_weights()
```
Компіляція моделей:

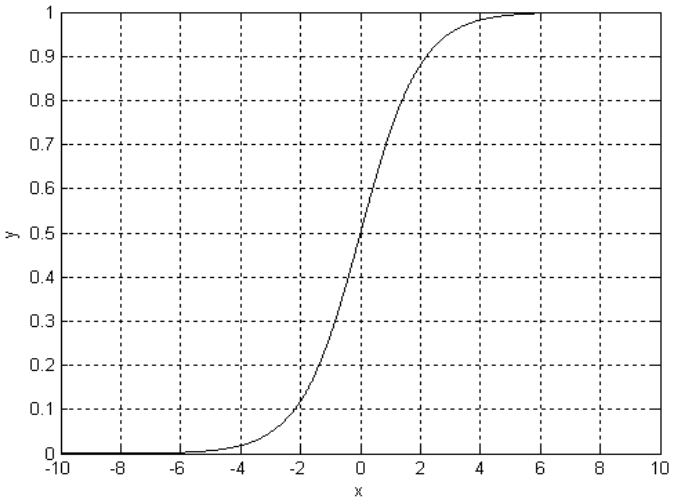
– багатошаровий персептрон:
```
# Бінарна класифікація
>>> model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])
# Багатокласова класифікація
>>> model.compile(optimizer='rmsprop',
loss='categorical_crossentropy', metrics=['accuracy'])
# Оцінювання
>>> model.compile(optimizer='rmsprop', loss='mse',
metrics=['mae'])
```
– рекурентна нейромережа:
```
# Визначення параметрів моделі
>>> model3.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
```
Навчання моделі.
```
>>> model3.fit(x_train4, y_train4, batch_size=32,
epochs=15, verbose=1, validation_data=(x_test4,
y_test4))
```
Обчислення якості моделі.
```
>>> score = model3.evaluate(x_test, y_test,
batch_size=32)
```
Емуляція (запуск) моделі.
```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```
Збереження та завантаження моделі.
```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```
Оптимізація параметрів моделі.
```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
optimizer=opt, metrics=['accuracy'])
```
Раннє зупинення навчання моделі.
```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4, y_train4, batch_size=32,
epochs=15, validation_data=(x_test4, y_test4),
callbacks=[early_stopping_monitor])
```

6.4 Приклади виконання завдань

Приклад 1. Побудувати за допомогою пакету Matlab графік сигмоїдної функції активації.

```
% створюємо набір значень аргументу функції
% із певним кроком
x=-10:0.01:10;
% обчислюємо значення функції активації
% логістичний сигмоїд
y=logsig(x);
% будуємо графік залежності y(x)
% суцільною лінією чорного кольору
plot(x, y,'k-');
% підписуємо осі графіку
xlabel('x');
ylabel('y');
% додаємо на графік координатну сітку
grid(on);
```

У результаті виконання розробленої програми в окремій формі (графічному вікні) на екрані отримаємо такий графік.



Приклад 2. Моделювання і навчання радіально-базисних НМ.

```
%Задаємо значення ознак екземплярів навчальної
вибірки: %3 екземпляри (стовпці), 1 ознака (рядки).
x = [1 2 3];
%Задаємо значення параметра, що прогнозується,
%для 3 екземплярів навчальної вибірки.
y = [2.0 4.1 5.9];
%Створюємо і навчаємо радіально-базисну НМ
net = newrb(x, y);
%Обчислюємо по навченій мережі net значення параметра,
%що прогнозується, для екземплярів, що характеризуються
%набором значень ознак x та видаємо його на екран.
a = sim(net, x)
```

Приклад 3. Моделювання і навчання НМ Хопфілда.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%2 екземпляри (стовпці), 3 ознаки (рядки).
x = [-1    1;
     -1   -1;
      1    1];
net = newhop(x); % Створюємо і навчаємо НМ Хопфілда.
```

Приклад 4. Моделювання і навчання НМ LVQ.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%7 екземплярів (стовпці), 1 ознака (рядки).
x=[1 -2 2 0 4 -5 3];
%Задаємо номери класів для 7 екземплярів
%навчальної вибірки.
y=[1  2 1 2 1  2 1];
%Перетворюємо номери класів у внутрішній формат.
yc=ind2vec(y);
%Створюємо нейронну мережу net і визначаємо її
%топологію: діапазон зміни значень ознак визначається
%функцією minmax, кількість схованих нейронів
%(кластерів) – 4, апріорна імовірність віднесення
%екземплярів до одного класу – 0.6, до іншого – 0.4, у
%якості методу навчання НМ використовуємо метод LVQ1.
net=newlvq(minmax(x),4, [0.6 0.4], 'learnlv1');
%Задаємо максимально припустиму кількість циклів
%навчання (епох).
net. trainParam. epochs= 1000;
%Задаємо період відображення інформації про процес
%навчання на екрані в циклах (епохах) навчання.
net. trainParam. show= 100;
net. trainParam. lr=0.05; %Задаємо крок навчання.
%Навчаємо нейронну мережу net на основі навчальної
%вибірки, представленої набором значень ознак
```

```
%екземплярів x і набором значень відповідних їм номерів
%класів y.
net=train(net, x, yc);
%Обчислюємо по навченій мережі net номери класів для
%екземплярів, що характеризуються набором значень
%ознак x.
a=sim(net, x);
%Перетворюємо номери кластерів у зручний для сприйняття
%формат і видаємо на екран.
ac=vec2ind(a)
```

Приклад 5. Моделювання і навчання багатошарового персептрона.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%6 екземплярів (стовпці), 3 ознаки (рядки).
x = [0.1 0.5 0.2 0.4 0.3 0.9;
     0.9 0.5 0.8 0.6 0.7 0.1;
     0.3 0.0 0.6 0.1 0.2 0.9];
%Задаємо номери класів для 6 екземплярів навчальної
%вибірки.
y = [1   0   1   0   1   0];
%Створюємо нейронну мережу net і визначаємо її
%топологію: діапазон зміни значень ознак [0 1],
%кількість ознак – 3, кількість вихідних змінних – 1, на
%першому шарі – 2 нейрони, на другому шарі 1 – нейрон,
%нейрони 1 і 2 шарів мають сигмоїдні функції активації
%(logsig), для навчання мережі використовується метод
% Левенберга-Марквардта (trainlm).
net=newff(repmat([0 1], 3, 1),[2,1],{'logsig',
'logsig'}, 'trainlm');
%Задаємо період відображення інформації про процес
%навчання на екрані в циклах (епохах) навчання.
net. trainparam. show=25;
net. trainparam. lr= 0.01; %Задаємо крок навчання.
%Задаємо максимально припустиму кількість циклів
%навчання (епох).
net. trainparam. epochs=500;
%Задаємо максимально припустиме значення критерію
%навчання (помилки навчання).
net. trainparam. goal=0.01;
%Визначаємо і запам'ятовуємо поточне значення лічильника
%часу в змінній ct.
ct=cputime;
%Навчаємо нейронну мережу net на основі навчальної
%вибірки, представленої набором значень ознак
%екземплярів x і набором значень відповідних їм
%номерів класів y.
net=train(net, x, y);
%Визначаємо поточне значення лічильника часу, віднімаємо
```

```
%від нього значення змінної ct – визначаємо час навчання
%НМ, що заносимо в змінну ct і видаємо на екран (ознака
%друку на екран – відсутність символу «;» наприкінці
%оператора).
ct=cputime-ct
%Обчислюємо по навченій мережі net номери класів для
%екземплярів, що характеризуються набором значень
%ознак x.
a=round(sim(net, x));
```

Приклад 6. Моделювання і навчання НМ SOM.

```
%Задаємо набір значень для однієї ознаки 400 екземплярів
%за допомогою генератора випадкових чисел.
x = rand(1,400);
%Створюємо нейронну мережу net і визначаємо її
%топологію: діапазон зміни значень ознак визначається
%функцією minmax, у першому шарі масив нейронів – 2x5.
net=newsom(minmax(x), [2 5]);
%Навчаємо нейронну мережу net (формуємо кластери) на
%основі навчальної вибірки, представленої набором
%значень ознак екземплярів x.
net=train(net, x);
%Обчислюємо по навченій мережі net номера кластерів,
%зіставлених нейронам вихідного шару НМ для екземплярів,
%що характеризуються набором значень ознак x.
y=sim(net, x);
%Перетворюємо номера кластерів у зручний для сприйняття
% формат і видаємо на екран.
yc=vec2ind(y)
```

Приклад 7. Моделювання і навчання мереж Елмана.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%8 екземплярів, 1 ознака.
x = [1 0 1 1 1 0 1 1]
%Задаємо значення цільової ознаки екземплярів.
y = [0 0 0 1 1 0 0 1]
%Створюємо мережу Елмана, що складається з 5 нейронів
%на першому шарі та 1 нейрона на другому шарі, які мають
%функції активації 'tansig' та 'logsig', відповідно.
net = newelm([0 1], [5 1], {'tansig', 'logsig'});
%Навчаємо мережу Елмана.
net = train(net, x, y);
%Обчислюємо по навченій мережі net номери класів для
%екземплярів, що характеризуються набором значень
%ознак x.
Y = sim(net, x);
```

Приклад 8. Використовуючи засоби пакету Matlab побудувати мережу Елмана для набору даних simpleseries_dataset.

```
% завантажуємо набір даних у вибірку <X, T>,
% де X – вхідні, а T – вихідні значення
[X, T] = simpleseries_dataset;
% створюємо мережу Елмана
% з затримками для шарів 1 і 2 та
% розміром схованого шару 10 нейронів
% задаємо як функцію навчання
% метод Левенберга-Марквардта
net = elmannet(1:2,10, 'trainlm');
% готуємо вхідні та цільові дані для моделювання
% та тренування мережі
[Xs, Xi, Ai, Ts] = preparets(net, X, T);
% запускаємо навчання мережі
net = train(net, Xs, Ts, Xi, Ai);
% емулюємо мережу
Y = net(Xs, Xi, Ai);
% визначаємо якість навчання мережі
perf = perform(net, Ts, Y)
```

Приклад 9. Написати функцію, що налагоджує ваги двошарової нейронної мережі, що містить $k1$ нейронів на першому шарі та $k2$ нейронів – на другому й навчається за вибіркою X (матриця незалежних змінних) і Y (вектор значень цільової функції). Точність апроксимації 0,01. Підрахувати кількість викликів функції помилки. У випадку, якщо не задані параметри $k1$, $k2$, X и Y, настроїти нейронну мережу, що містить 4 нейрони на першому шарі й 1 нейрон на другому шарі.

Розв'язок оформимо у вигляді двох файлів NNweights. m і ErrorPodbor. m.

Файл NNweights. m

```
function mean1 = NNweights(k1_, k2_, x1, y1)
%встановлюємо глобальні змінні
global Q k1 k2 net x y
%net - двошарова нейронна мережа
%k1 - кількість нейронів на 1-му шарі
%k2 - кількість нейронів на 2-му шарі
if (nargin~=4) % параметри не задані
k1=4; k2=1;
x=[елементимасивуX]'; y=[елементимасивуY];
%Тут замінити елементимасивуX та елементимасивуY
%конкретними числами
```

```
    end;
    if (nargin==4)        % усі параметри задані
        k1=k1_; k2=k2_; x=x1; y=y1;
    end;
    %Нормалізація X та Y
    x_min=min(x',[],1); x_max=max(x',[],1);
    for i=1:1: size(x,1)
       for j=1:1: size(x,2)
         x(i,j)=(x(i,j)-x_min(i))./abs(x_max(i)-x_min(i));
       end;
    end;
    y_max=max(y); y_min=min(y);
    y=(y-y_min)/abs(y_max-y_min);
    %Створення нейронної мережі прямого поширення, нейрони
    %якої використовують функції активації логістичний
    %сигмоїд, а для навчання застосовується метод
    %Левенберга-Марквардта (тут вказано просто як заглушка).
    net=newff(repmat([0   1],   size(x,1),1),   [k1,   k2],
{'logsig','logsig'},'trainlm');
    Q=0;
    %Запуск навчання нейромережі за допомогою
    %генетичного пошуку
    [W error] = ga(@ErrorPodbor, k1*(size(x,1)+1)+k2*(k1+1),
gaoptimset('FitnessLimit', 0.01));
    %Видаємо розраховані значення змінних на екран
    Q
    error
    net. IW{1,1}
    net. b{1,1}
    net. b{2,1}
    net. LW{2,1}
    mean1 = W'; % Присвоюємо функції значення
```

Файл ErrorPodbor. m

```
function mean1 = ErrorPodbor(w)
%встановлюємо глобальні змінні
global k1 k2 net x y Q
%net - двошарова нейронна мережа
%k1 - кількість нейронів на 1-му шарі
%k2 - кількість нейронів на 2-му шарі
%Q - номер поточної ітерації
%Встановлюємо значення ваг
net_IW=[];
for i=1:1: k1
    for j=1:1: size(x,1)
        net_IW(i, j)=w(j+size(x,1)*(i-1));
    end;
end;
```

```
net. IW{1,1}=net_IW; net. IW{1,1}; net_b11=[];
for i=1:1: k1
    net_b11(i)=w(i+size(x,1)*k1);
end;
net. b{1,1} = net_b11'; net_b21=[];
for i=1:1: k2
    net_b21(i)=w(i+size(x,1)*k1+k1);
end;
net. b{2,1} = net_b21; net_LW=[];
for i=1:1: k2
   for j=1:1: k1
    net_LW(i, j)=w(j+size(x,1)*(i-1)+size(x,1)*k1+k1+k2);
   end;
end;
net. LW{2,1} = net_LW;
%Емулюємо роботу мережі при встановлених вагах
a=sim(net, x);
%Обчислюємо помилку мережі
for i=1:1: size(y,1)
    E(i)=abs(y(i)-a(i))/(abs(y(i)));
end;
Q=Q+1;
A=[Q mean(E)];
mean1=mean(E); % результат функції - середня помилка
```

? 6.5 Контрольні питання

1. Які ви знаєте програмні засоби для моделювання НМ?
2. У якому модулі пакету Matlab містяться засоби для моделювання нейромереж?
3. Для чого призначений засіб nntool пакету Matlab?
4. Опишіть подання структури нейромережі у пакеті Matlab.
5. Функції для моделювання нейромереж у програмному та ручному режимах у пакеті Matlab.
6. Функції вирішення задач оптимізації на основі еволюційного пошуку бібліотеки Genetic Algorithm and Direct Search Toolbox пакету Matlab.
7. Засоби моделювання нейронних мереж у пакеті Statistica Neural Networks.

8. Процес побудови нейромережі пакеті Statistica Neural Networks.

9. Моделювання і навчання радіально-базисних НМ у пакеті Matlab.

10. Моделювання і навчання НМ Хопфілда у пакеті Matlab.

11. Моделювання і навчання НМ LVQ у пакеті Matlab.

12. Моделювання і навчання багатошарового персептрона у пакеті Matlab.

13. Моделювання і навчання НМ SOM у пакеті Matlab.

14. Моделювання і навчання мереж Елмана у пакеті Matlab.

15. Як впливає репрезентативність навчальної вибірки на результат побудови мережі?

16. Що таке нормалізація даних?

17. З якою метою проводиться масштабування даних?

18. Назвіть основні принципи створення навчальної вибірки.

19. Чи впливає репрезентативність навчальної вибірки на точність класифікації екземплярів тестової вибірки?

20. Чи впливає репрезентативність тестової вибірки на точність класифікації екземплярів тестової вибірки?

21. Чи впливає репрезентативність тестової вибірки на точність навчання мережі по навчальній вибірці?

22. Чи залежить якість навчання нейромереж від якості та обсягу навчальної вибірки?

23. Що таке генеральна сукупність, вибірка, екземпляр, ознака?

24. Вимоги до навчальних вибірок даних.

25. Що таке репрезентативна вибірка даних?

26. Чи повинна навчальна вибірка бути репрезентативною?

27. Чи повинна тестова вибірка бути репрезентативною?

28. Чи впливає обсяг навчальної вибірки на швидкість навчання нейромереж?

29. Модуль Neural Network Toolbox.

30. Пакет Statistica Neural Networks.

31. Моделі нейроелементів у пакеті Matlab.

32. Проаналізуйте внутрішню структуру функції ga пакету Matlab: основні змінні, параметри, методи та допоміжні функції, їх призначення та використання.

33. Опишіть основні засоби бібліотеки NeuroLab.

34. Як інсталювати бібліотеку NeuroLab?

35. Як здійснюється моделювання багатошарових нейромереж у бібліотеці NeuroLab?

36. Як здійснюється моделювання нейромереж Хопфілда у бібліотеці NeuroLab?

37. Як здійснюється моделювання нейромереж Кохонена у бібліотеці NeuroLab?

38. Як здійснюється моделювання нейромереж Хеммінга у бібліотеці NeuroLab?

39. Як здійснюється моделювання нейромереж Елмана у бібліотеці NeuroLab?

40. Опишіть основні засоби бібліотеки Keras.

41. Як інсталювати бібліотеку Keras?

42. Розглянемо найважливіші методи бібліотеки Keras.

43. Як конфігурувати модель для навчання у бібліотеці Keras?

44. Як здійснити навчання нейромережі у бібліотеці Keras?

45. Як отримати передбачення (оцінку) для розпізнаваного екземпляра нейромережею у бібліотеці Keras?

6.6 Практичні завдання

Завдання 1. Лабораторна робота

«Нейромережі прямого поширення»

*Мета роботи***:** Вивчити архітектури формального нейрона, багатошарового персептрона і радіально-базисної мережі, а також методи їхнього навчання; ознайомитися з програмними продуктами, що моделюють персептрони та радіально-базисні мережі.

Завдання до роботи.

1. Ознайомитися з рекомендованою літературою.

2. Вивчити архітектури формального нейрона, багатошарового персептрона та радіально-базисної мережі, а також методи їхнього навчання.

3. Використовуючи документацію програмних засобів (Python з бібліотекою Keras або Matlab, або Statistica Neural Networks), вивчити їх архітектуру і компоненти, призначені для моделювання і навчання нейромереж прямого поширення, введення і виведення даних, підготовки звітів (виведення і відображення результатів роботи).

4. Сформувати навчальну і тестову вибірки даних. Визначити за табл. 6.4 кількості шарів та нейронів у шарах для побудови багатошарових персептронів, а також функцію активації для завдання п. 6.

Таблиця 6.4 – Параметри нейромереж для варіантів

Номер варіанта	Багатошаровий персептрон		Назва функції активації
	Кількість шарів	Кількості нейронів у шарах	
1	2	3–1	логістична сигмоїдна
2	3	3–3–1	тангенційна сигмоїдна
3	4	3–3–3–1	логістична сигмоїдна
4	2	2–1	тангенційна сигмоїдна
5	3	2–2–1	логістична сигмоїдна
6	4	2–2–2–1	тангенційна сигмоїдна
7	2	5–1	логістична сигмоїдна
8	3	5–5–1	тангенційна сигмоїдна
9	4	5–5–5–1	логістична сигмоїдна
10	2	4–1	тангенційна сигмоїдна
11	3	4–4–1	логістична сигмоїдна
12	4	4–4–4–1	тангенційна сигмоїдна
13	2	6–1	логістична сигмоїдна
14	3	6–3–1	тангенційна сигмоїдна
15	4	6–4–3–1	логістична сигмоїдна
16	2	7–1	тангенційна сигмоїдна
17	3	7–7–1	логістична сигмоїдна
18	4	7–3–4–1	тангенційна сигмоїдна
19	2	8–1	логістична сигмоїдна
20	3	8–8–1	тангенційна сигмоїдна
21	4	8–7–6–1	логістична сигмоїдна
22	2	2–1	тангенційна сигмоїдна
23	3	2–3–1	логістична сигмоїдна
24	4	2–4–4–1	тангенційна сигмоїдна
25	2	3–1	логістична сигмоїдна
26	3	6–2–1	тангенційна сигмоїдна
27	4	7–6–3–1	логістична сигмоїдна
28	2	4–1	тангенційна сигмоїдна
29	3	9–2–1	логістична сигмоїдна
30	4	9–3–2–1	тангенційна сигмоїдна

5. Використовуючи програмний засіб для відповідних варіанту студента вхідних даних вирішити прикладну задачу на основі одношарового та багатошарового персептронів,

використовуючи різні методи навчання та різні функції активації. Вирішити ту саму задачу на основі радіально-базисної мережі.

6. Змінюючи значення кроку навчання, для одношарового та багатошарового персептронів з заданою функцією активації, що відповідає номеру варіанту студента (див. табл. 6.4) дослідити, як впливає величина кроку навчання на час навчання. Побудувати графіки залежності часу навчання персептронів від величини кроку навчання.

7. Зберегти у файлі на диску початкові параметри нейромоделей, результати їхнього навчання (матрицю ваг та структуру з зазначенням функцій активації і кількості нейроелементів у шарах) та роботи для навчальної та контрольної вибірок (помилку класифікації (оцінювання), значення на входах і виході, час навчання, час класифікації).

8. Результати виконання пп. 5–7 занести у таблицю, стовпці якої повинні мати назви: назва архітектури нейромережі, кількість шарів, кількість нейронів у шарах, функції активації нейронів у шарах, метод навчання, час навчання, час класифікації (оцінювання) навченої мережі для навчальної вибірки, помилка класифікації (оцінювання) для навчальної вибірки, час класифікації (оцінювання) навченої мережі для тестової вибірки, помилка класифікації (оцінювання) для тестової вибірки.

9. Проаналізувати отримані результати і зробити висновки про те, як впливає вид функції активації формального нейрона на час навчання і час класифікації (оцінювання), а також величину помилки навчання і класифікації (оцінювання) одношарового персептрона; як впливають вид функції активації і вид коригувального правила ваг (метод навчання) на час навчання і класифікації (оцінювання), а також величину помилки навчання та класифікації багатошарового персептрона; яка модель нейромереж прямого поширення краще підходить для вирішуваної задачі.

10. Порівняти одношаровий та багатошаровий персептрони, радіально-базисну мережу та багатошаровий персептрон за швидкістю навчання і точністю класифікації; принципами побудови архітектури.

11. Оформити звіт з роботи.

Зміст звіту.

1. Тема та мета роботи.

2. Завдання до роботи, що містить стислий опис вирішуваної практичної задачі (не більше 0,5 сторінки).

3. Короткі теоретичні відомості (не більше 2 сторінок), що містять стислий опис архітектур формального нейрона, багатошарового персептрона і радіально-базисної мережі та методів їхнього навчання.

4. Опис процесу виконання роботи (не більше 3 сторінок).

5. Тексти програм, розроблених (модифікованих) студентом.

6. Вхідні дані та результати роботи програм, узагальнюючі таблиці, графіки.

7. Висновки, що відображують результати виконання роботи та їх критичний аналіз.

 *Завдання 2.* Лабораторна робота

«Нейромережі зі зворотними зв'язками»

Мета роботи – вивчити моделі і методи навчання нейромереж зі зворотними зв'язками, розглянути приклади їхнього практичного використання; порівняти їхні можливості з можливостями нейромереж прямого поширення; ознайомитися зі стандартними програмними засобами для моделювання мереж зі зворотними зв'язками.

Завдання до роботи.

1. Ознайомитися з рекомендованою літературою.

2. Вивчити архітектури нейромереж Хопфілда та Елмана, а також методи їхнього навчання.

3. Використовуючи документацію Python або Matlab, вивчити його архітектуру і компоненти, призначені для моделювання і навчання нейромереж зі зворотними зв'язками, введення і виведення даних, підготовки звітів (виведення і відображення результатів роботи).

4. Сформувати навчальну і тестову вибірки даних.

5. Використовуючи Python або Matlab для сформованих вибірок даних вирішити задачу побудови нейромереж Хопфілда та Елмана.

6. Зберегти у файлі на диску початкові параметри нейромоделей, результати їхнього навчання (матрицю ваг та структуру з зазначенням функцій активації і кількості

нейроелементів у шарах) та роботи для навчальної та контрольної вибірок (помилку класифікації (оцінювання), значення на входах і виході, час навчання, час класифікації).

7. Результати виконання пп. 5–6 занести у таблицю, стовпці якої повинні мати назви: назва архітектури нейромережі, кількість шарів, кількість нейронів у шарах, функції активації нейронів у шарах, метод навчання, час навчання, час класифікації навченої мережі для навчальної вибірки, помилка класифікації для навчальної вибірки, час класифікації навченої мережі для тестової вибірки, помилка класифікації для тестової вибірки.

8. Проаналізувати отримані результати і дати порівняльну характеристику мереж Хопфілда та Елмана. Дати порівняльну характеристику мереж зі зворотними зв'язками та мереж прямого поширення за швидкістю навчання і точністю класифікації; принципами побудови архітектури.

9. Оформити звіт з роботи.

Зміст звіту.

1. Тема та мета роботи.

2. Завдання до роботи, що містить стислий опис вирішуваної практичної задачі (не більше 0,5 сторінки).

3. Короткі теоретичні відомості (не більше 2 сторінок), що містять стислий опис архітектур та методів навчання нейромереж зі зворотними зв'язками.

4. Опис процесу виконання роботи (не більше 3 сторінок).

5. Тексти програм, розроблених (модифікованих) студентом.

6. Вхідні дані та результати роботи програм, узагальнюючі таблиці, графіки.

7. Висновки, що відображують результати виконання роботи та їх критичний аналіз.

 *Завдання 3.* Лабораторна робота

«Нейромережі з латеральними зв'язками»

Мета роботи – вивчити моделі нейронних мереж з латеральними зв'язками та методи їхнього навчання; порівняти можливості нейромереж з латеральними зв'язками з можливостями нейромереж прямого поширення та можливостями мереж зі зворотними зв'язками; ознайомитися з

програмними засобами, що моделюють нейромережі з латеральними зв'язками.

Завдання до роботи.

1. Ознайомитися з рекомендованою літературою.

2. Вивчити архітектури нейромереж з латеральними зв'язками, а також методи їхнього навчання.

3. Використовуючи документацію Python або Matlab, або Statistica Neural Networks, вивчити їх архітектуру і компоненти, призначені для моделювання і навчання нейромереж з латеральними зв'язками, введення і виведення даних, підготовки звітів (виведення і відображення результатів роботи).

4. Сформувати навчальну і тестову вибірки даних.

5. Використовуючи Python або Matlab, або Statistica Neural Networks для сформованих вибірок навчити нейромережу Кохонена LVQ. При цьому окремо дослідити властивості різних методів навчання. Для тієї ж задачі сформувати кластери на основі SOM.

6. Зберегти у файлі на диску початкові параметри нейромоделей, результати їхнього навчання (матрицю ваг та структуру з зазначенням функцій активації і кількості нейроелементів у шарах) та роботи для навчальної та контрольної вибірок (помилку класифікації (оцінювання), значення на входах і виході, час навчання, час класифікації).

7. Результати виконання пп. 5–6 занести у таблицю, стовпці якої повинні мати назви: назва архітектури нейромережі, кількість шарів, кількість нейронів у шарах, функції активації нейронів у шарах, метод навчання, час навчання, час класифікації навченої мережі для навчальної вибірки, помилка класифікації для навчальної вибірки, час класифікації навченої мережі для тестової вибірки, помилка класифікації для тестової вибірки.

8. Проаналізувати отримані результати і дати порівняльну характеристику мереж з латеральними зв'язками SOFM та LVQ. Дати порівняльну характеристику мереж з латеральними зв'язками, мереж зі зворотними зв'язками та мереж прямого поширення за швидкістю навчання і точністю класифікації; принципами побудови архітектури.

9. Оформити звіт з роботи.

Зміст звіту.

1. Тема та мета роботи.

2. Завдання до роботи, що містить стислий опис вирішуваної практичної задачі (не більше 0,5 сторінки).

3. Короткі теоретичні відомості (не більше 2 сторінок), що містять стислий опис архітектур та методів навчання нейромереж з латеральними зв’язками.

4. Опис процесу виконання роботи (не більше 3 сторінок).

5. Тексти програм, розроблених (модифікованих) студентом.

6. Вхідні дані та результати роботи програм, узагальнюючі таблиці, графіки.

7. Висновки, що відображують результати виконання роботи та їх критичний аналіз.