**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4**

по курсу «Нейроинформатика».

Тема: «Сети с радиальными базисными элементами».

Студент: Якимович А.И.

Группа: 80-408Б

Вариант: 20

Оценка:

Москва, 2018

Постановка задачи.

Целью работы является исследование свойств некоторых видов сетей с радиальными базисными элементами, алгоритмов обучения, а также применение сетей в задачах классификации и аппроксимации функции.

Задание 1: использовать вероятностную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.

*Вариант 20:*

*Эллипс: a = 0.4, b = 0.4, α = 0, x0 = −0.1, y0 = 0.15*

*Эллипс: a = 0.7, b = 0.7, α = 0, x0 = 0, y0 = 0*

*Парабола: p = −1, α = 0, x0 = 0.8, y0 = 0*

Задание 2: использовать сеть с радиальными базисными элементами (RBF) для классификации точек в случае, когда классы не являются линейно разделимыми.

Задание 3: использовать обобщенно-регрессионную нейронную сеть для аппроксимации функции. Проверить работу сети с рыхлыми данными.

*Вариант 20: x = cos(cos(t)\*t^2 + 5t) , t ∈ [0, 3.5], h = 0.01*

Описание алгоритма.

Cеть с радиальными базисными элементами — локально-рецептивные многослойные сети, в которых, в противоположность MLP, добавление/исключение нового элемента или корректировка его параметров оказывает локальное воздействие на свойства сети в целом. Эта локальность обусловлена тем, что для любого входного вектора вносить вклад в выход локально-рецептивной сети будет лишь ограниченное число нейронов за счет того, что в таких сетях «ненулевое» срабатывание происходит только для части нейронов сети. Следствие «локальности» сети — любое изменение обучающего набора (добавление, изъятие или замена части обучающих примеров) требует только частичного переобучения сети.

Сеть с радиальными базисными элементами использует понятие потенциальной функции. Потенциальная функция *K(x; y)* — функция двух переменных*, x; y* — точки из пространства X. Функция K(x; y) всюду положительна, она убывает при удалении точки *(x; y)* от точки *y =* , при *x =* она достигает максимума. *K* — функция расстояния между точками *x* и *y*.

Вероятностная нейронная сеть – сеть с радиальными базисными элементами, использующаяся для классификации, *i-ый* выход которой интерпретируется как вероятность принадлежности объекта к *i-ому* классу.

**Задача классификации.**

**Обучение.**

Пусть учителем показана точка *x* = и сообщено, что она принадлежит к классу *A*. Примем точку x = за «источник потенциала», положив x = , т.е. построим «холм» с вершиной в этой точке и запомним, что этот холм относится к точке из класса *A.* Для следующих точек из *A* или из *B* — аналогичные холмы с указанием на класс, к которому принадлежит каждый холм. Потенциалы, построенные над точками, принадлежащими классам *A* и *B:*, . Холмы, построенные над точками x из A, образуют в совокупности «гору» над областью, где расположены эти точки. Аналогичная «гора» — для точек x из B.

**Предсказание.** Точка x относится к классу *A*, если:и к классу *B* — в противном случае.

**Задача аппроксимации.**

Область определения функции дробится на отрезки. На границы отрезков помещаются центры потенциальных функций. В ходе обучения сеть настраивает максимальные значения этих функций и степень их «крутости» (отклонение) таким образом, чтобы среднеквадратическая ошибка стала минимальной.

Общая информация.

Программа написана на языке Python с применением библиотек numpy, neupy, sklearn, pandas и matplotlib (для графиков).

Данная программа может решать задачи связанные с аппроксимацией функций многих переменных, классификацией объектов, в том числе и линейно-неразделимых.

Запуск программы.

Чтобы воспользоваться программой, необходимо запустить программу на любом интерпретаторе Python версии 3.x.x.

Входные данные и результаты.

Вариант 20.

|  |  |
| --- | --- |
| **Вход** | **Выход** |
| Вероятностная сеть STD = 0.1  C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\inputrgb12.png | C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task1_train0.1.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task1_test0.1.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task1_valid0.1.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task1_final0.1.png.png  Точность предсказания train = 1, test = 1, valid = 1 |
| Вероятностная сеть STD = 0.3 | C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task1_valid0.3.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task1_final0.3.png |
| RBF STD = 0.1  C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\inputrgb12.png | C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_test0.9.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_train0.9.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_valid0.9.png  Точность предсказания train = 0.805, test = 0.741, valid = 0.770 |
| RBF STD = 0.3  C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\inputrgb12.png | C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_valid0.7.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_train0.7.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_test0.7.pngC:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task2_final0.7.png  Точность предсказания train = 0.894, test = 0.833, valid = 0.877 |
| Обобщенно-регрессионная сеть 90% и 10%.  *x = cos(cos(t)\*t^2 + 5t) , t ∈ [0, 3.5], h = 0.01* | **Квадратичная ошибка на train = 0.012**  **Квадратичная ошибка на test = 0.015** |
| Обобщенно-регрессионная сеть 80% и 20%.  *x = cos(cos(t)\*t^2 + 5t) , t ∈ [0, 3.5], h = 0.01*  *C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task3_input.png* | C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\task3_trpr20.png  **Квадратичная ошибка на train = 0.014**  **Квадратичная ошибка на test = 0.016** |

Выводы:

RBF – это локально-рецептивные многослойные сети, в которых, в противоположность MLP, добавление/исключение нового элемента или корректировка его параметров оказывает локальное воздействие на свойства сети в целом. Эта локальность обусловлена тем, что для любого входного вектора вносить вклад в выход локально-рецептивной сети будет лишь ограниченное число нейронов за счет того, что в таких сетях «ненулевое» срабатывание происходит только для части нейронов сети. Следствие «локальности» сети — любое изменение обучающего набора (добавление, изъятие или замена части обучающих примеров) требует только частичного переобучения сети. Это позволяет эффективно строить предсказания, когда обучающие данные поступают динамически.

Также, как оказалось, на нашем примере в задаче аппроксимации сети RBF показали лучший результат на тестовой выборке чем MLP, однако на обучающей выборке проиграли.

Исходный код.

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.metrics import accuracy\_score, mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from neupy.algorithms import PNN, GRNN

from neupy import estimators

import sys

from sys import stdout

np.set\_printoptions(threshold=np.nan)

a1 = 0.4

b1 = 0.4

x0\_1 = 0.1

y0\_1 = -0.15

a2 = 0.7

b2 = 0.7

x0\_2 = 0

y0\_2 = 0

p3 = -1

x0\_3 = 0.8

y0\_3 = -0.8

step = 0.025

t = np.linspace(0, 2\*np.pi, int(2\*np.pi/step), endpoint=True)

x = np.linspace(-1, 1, int(2/step), endpoint=True)

def f(x0, a, t):

return x0 + a\*np.cos(t)

def g(y0, b, t):

return y0 + b\*np.sin(t)

def build\_model(model, train, test, valid):

model.fit(train.iloc[:, :-1], train.target)

p = []

p.append((model.predict\_proba(train.iloc[:, :-1])).reshape(-1, 3))

#print('Train score:', accuracy\_score(train['target'], p[-1]))

p.append((model.predict\_proba(test.iloc[:, :-1])).reshape(-1, 3))

#print('Test score', accuracy\_score(test['target'], p[-1]))

p.append((model.predict\_proba(valid.iloc[:, :-1])).reshape(-1, 3))

#print('Valid score', accuracy\_score(valid['target'], p[-1]))

titles = ['train', 'test', 'valid']

for idx, df in enumerate((train, test, valid)):

plt.scatter(df.x, df.y, c=p[idx])

plt.grid(True)

plt.title(titles[idx])

plt.show()

h = 0.025

grid\_pred = np.array([model.predict\_proba(np.array([[i, j]])) for i in np.arange(-1.2, 1.2+h, h)

for j in np.arange(-1.2, 1.2+h, h)])

grid\_pred = grid\_pred.reshape(-1, 3)

x\_vals = np.arange(-1.2, 1.2+h, h)

y\_vals = np.arange(-1.2, 1.2+h, h)

xx, yy = np.meshgrid(x\_vals, y\_vals)

#lb = LabelBinarizer()

#data = lb.fit\_transform(grid\_pred.flatten())

#print(data)

plt.scatter(yy, xx, c=grid\_pred)

plt.show()

x1 = f(x0\_1, a1, t)

y1 = g(y0\_1, b1, t)

x2= f(x0\_2, a2, t)

y2= g(y0\_2, b2, t)

#x3= f(a3, t)

#y3= g(b3, t)

x3 = x + x0\_3

y3 = p3 \* x \* x + y0\_3

plt.plot(x1, y1, 'r')

plt.plot(x2, y2, 'g')

plt.plot(x3, y3, 'b')

plt.grid(True)

plt.show()

df1 = pd.DataFrame({'x' : x1, 'y' : y1, 'target' : 0})

df2 = pd.DataFrame({'x' : x2, 'y' : y2, 'target' : 1})

df3 = pd.DataFrame({'x' : x3, 'y' : y3, 'target' : 2})

def split\_df(df):

x\_train, x\_test = train\_test\_split(df, test\_size=0.3, shuffle=True, random\_state=21)

x\_valid, x\_test = train\_test\_split(x\_test, test\_size=0.3, shuffle=True, random\_state=14)

return x\_train, x\_valid, x\_test

train = []

valid = []

test = []

for df in (df1, df2 ,df3):

tr, v, te = split\_df(df)

train.append(tr)

valid.append(v)

test.append(te)

train = pd.concat(train)

valid = pd.concat(valid)

test = pd.concat(test)

build\_model(PNN(std=0.1), train, test, valid)

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.metrics import accuracy\_score, mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from neupy.algorithms import PNN, GRNN

from neupy import estimators

a1 = 0.4

b1 = 0.4

x0\_1 = 0.1

y0\_1 = -0.15

a2 = 0.7

b2 = 0.7

x0\_2 = 0

y0\_2 = 0

p3 = -1

x0\_3 = 0.8

y0\_3 = -0.8

step = 0.025

t = np.linspace(0, 2\*np.pi, int(2\*np.pi/step), endpoint=True)

x = np.linspace(-1, 1, int(2/step), endpoint=True)

def f(x0, a, t):

return x0 + a\*np.cos(t)

def g(y0, b, t):

return y0 + b\*np.sin(t)

def build\_model(model, train, test, valid):

model.fit(train.iloc[:, :-1], train.target)

p = []

p.append(model.predict(train.iloc[:, :-1]))

print('Train score:', accuracy\_score(train['target'], p[-1]))

p.append(model.predict(test.iloc[:, :-1]))

print('Test score', accuracy\_score(test['target'], p[-1]))

p.append(model.predict(valid.iloc[:, :-1]))

print('Valid score', accuracy\_score(valid['target'], p[-1]))

titles = ['train', 'test', 'valid']

for idx, df in enumerate((train, test, valid)):

plt.scatter(df.x, df.y, c=p[idx])

plt.grid(True)

plt.title(titles[idx])

plt.show()

h = 0.025

grid\_pred = np.array([model.predict(np.array([[i, j]])).round() for i in np.arange(-1.2, 1.2+h, h)

for j in np.arange(-1.2, 1.2+h, h)])

x\_vals = np.arange(-1.2, 1.2+h, h)

y\_vals = np.arange(-1.2, 1.2+h, h)

xx, yy = np.meshgrid(x\_vals, y\_vals)

lb = LabelBinarizer()

data = lb.fit\_transform(grid\_pred.flatten())

plt.scatter(yy, xx, c=data)

plt.show()

x1 = f(x0\_1, a1, t)

y1 = g(y0\_1, b1, t)

x2= f(x0\_2, a2, t)

y2= g(y0\_2, b2, t)

#x3= f(a3, t)

#y3= g(b3, t)

x3 = x + x0\_3

y3 = p3 \* x \* x + y0\_3

plt.plot(x1, y1, 'navy')

plt.plot(x2, y2, 'purple')

plt.plot(x3, y3, 'yellow')

plt.grid(True)

plt.show()

df1 = pd.DataFrame({'x' : x1, 'y' : y1, 'target' : 0})

df2 = pd.DataFrame({'x' : x2, 'y' : y2, 'target' : 1})

df3 = pd.DataFrame({'x' : x3, 'y' : y3, 'target' : 2})

def split\_df(df):

x\_train, x\_test = train\_test\_split(df, test\_size=0.3, shuffle=True, random\_state=21)

x\_valid, x\_test = train\_test\_split(x\_test, test\_size=0.3, shuffle=True, random\_state=14)

return x\_train, x\_valid, x\_test

train = []

valid = []

test = []

for df in (df1, df2 ,df3):

tr, v, te = split\_df(df)

train.append(tr)

valid.append(v)

test.append(te)

train = pd.concat(train)

valid = pd.concat(valid)

test = pd.concat(test)

build\_model(SVC(gamma=0.9), train, test, valid)

……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

import numpy as np

import sklearn.metrics

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from neupy.algorithms import PNN, GRNN

from neupy import estimators

h = 0.01

t = np.linspace(0, 3.5, int(3.5/0.02), endpoint=True)

x = np.cos(np.cos(t) \* t\*\*2 + 5\*t)

plt.plot(t, x, c = 'r');

plt.show()

def build\_model2(std, train\_size, t, x):

train\_size = int(t.shape[0] \* train\_size)

# X\_train = t[:train\_size]

# y\_train = x[:train\_size]

# X\_test = t[train\_size:]

# y\_test = x[train\_size:]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(t, x, train\_size=train\_size, shuffle=True, random\_state=14)

scaler\_x = StandardScaler()

scaler\_y = StandardScaler()

tmp\_train\_scaled\_x = scaler\_x.fit\_transform(X\_train[:, np.newaxis])

tmp\_test\_scaled\_x = scaler\_x.transform(X\_test[:, np.newaxis])

tmp\_train\_scaled\_y = scaler\_y.fit\_transform(y\_train[:, np.newaxis])

grnn = GRNN(std=std)

grnn.fit(tmp\_train\_scaled\_x, tmp\_train\_scaled\_y)

pred\_x = grnn.predict(tmp\_train\_scaled\_x)

pred\_x = scaler\_y.inverse\_transform(pred\_x)

mse = mean\_squared\_error(y\_train, pred\_x.flatten())

print(f'RMSE = {np.sqrt(mse)}')

plt.plot(t, x, c = 'r');

plt.scatter(X\_train, y\_train, label='train')

plt.scatter(X\_train, pred\_x, label='predict')

plt.legend()

plt.show()

pred\_x = grnn.predict(tmp\_test\_scaled\_x)

pred\_x = scaler\_y.inverse\_transform(pred\_x)

mse = mean\_squared\_error(y\_test, pred\_x.flatten())

print(f'RMSE = {np.sqrt(mse)}')

plt.plot(t, x, c = 'r');

plt.scatter(X\_test, y\_test, label='test')

plt.scatter(X\_test, pred\_x, label='predict')

plt.legend()

plt.show()

build\_model2(0.05, 0.8, t, x)