

Neural Network Computing

Homework 5

Liang Niu
N19486574
ln932@nyu.edu

December 6, 2016

1 Program

1.1 Source Code

SOURCE CODE (Python 2.7, numpy needed, anaconda recommended):

```
import numpy as np
from itertools import combinations
import random

MAXLOOP = 10000 # parameter for AM.predict_converge(): maximum loop times
TESTLOOP = 100 # parameter for Q2: for a err number, try how many times
ERRRANGE = 35 # parameter for Q2: the domain of err number
VERBOSE = False # True to output more details

def trans(x): # turn {1,0,-1,?} into {# , ?, -}
    if x == 1: return '#'
    if x == -1: return '-'
    if x == 0: return '?'
    return '_'

def load_data(filename): # load data from txt file
    with open(filename) as f:
        lines = [x.ljust(35, '_').replace('_', '0').replace('#', '1') for x in f.r
        data = [[1 if x=='1' else -1 for x in list(s) ] for s in lines]
        return np.array(data)

def display(data): # visualize a pattern(a 35x1 array)
    assert(data.size and data.size%35==0)
    data.reshape(data.size/35, 35)
    if len(data.shape)==1:
```

```

        data = np.array([data])
    for d in data:
        assert(d.shape[0]==35)
        d = [trans(e) for e in d]
        for i in range(7):
            print ''.join(d[5*i:5*i+5])
        print '-'*80

def disturb(raw_data, n, mode = "missing"):
    """
    take (data, n, mode) as input,
    data is a (35,) np array,
    generate some noise to data, mode is 'missing' or 'mistake',
    n is the number of noisy point.
    raw data is not changed
    """
    assert(mode == "missing" or mode == "mistake")
    assert(raw_data.size == 35)
    assert(n<=35 and n>=0)
    factor = 0 if mode == "missing" else -1
    data = np.copy(raw_data)
    data = data.reshape((35,))
    chose = random.sample(range(35), n)
    for ind in chose:
        data[ind] *= factor
    return data

class AM:
    def __init__(self, X):
        self.q = np.size(X,0)
        self.n = np.size(X,1)
        self.w = np.zeros((self.n, self.n))
        self.threshold = 0
        self.w = X.T.dot(X)
        self.w0 = np.copy(self.w)
        np.fill_diagonal(self.w0, 0)
        # print "w is:\n", self.w
        # print "w0 is:\n", self.w0
        def f_notvec(x):
            if x > self.threshold:
                return 1
            elif x < self.threshold:
                return -1
            else:
                return 0
        self.f = np.vectorize(f_notvec, cache=True)

```

```

def predict(self, x):
    """
    simply predict
    """
    return (self.f(x.dot(self.w)), self.f(x.dot(self.w0)))

def predict_converge(self, x):
    """
    predict iteratively until converge
    """
    p0 = self.f(x.dot(self.w0))
    for i in range(MAXLOOP): # at most iterate MAXLOOP times
        p0_temp = self.f(p0.dot(self.w0))
        if np.array_equal(p0, p0_temp):
            break
        else:
            p0 = p0_temp
    p = self.f(x.dot(self.w))
    for i in range(MAXLOOP): # at most iterate MAXLOOP times
        p_temp = self.f(p.dot(self.w))
        if np.array_equal(p, p_temp):
            break
        else:
            p = p_temp
    return (p, p0)

if __name__ == "__main__":
    # load data
    train_file = "TenDigitPatterns.txt"
    train_data = load_data(train_file)
    if VERBOSE:
        display(train_data)

    #####
    # first question #
    #####
    good_c = []
    if True:
        # if False:
        print "*" * 80
        print "#_first_question_#"
        for num in range(1,8):
            for c in combinations(range(10), num):
                # print c
                part_train_data = train_data[np.array(c)]

```

```

am = AM(part_train_data)

for p in [(0,am.predict_converge(part_train_data)), (1,am.predict_converge(part_train_data))]:
    mode = "converge" if p[0]==0 else "direct"
    pre = p[1]
    p0, p1= False, False
    if np.array_equal(pre[0], part_train_data):
        p0 = True
    if np.array_equal(pre[1], part_train_data):
        p1 = True
    if p0 or p1:
        print mode, c,
        good_c.append(c)
        if p0:
            print "p",
        if p1:
            print "p0",
        print ""

print good_c

#####
# second question #
#####
if True:
    # if False:
        print "*" * 80
        print "#_second_question_#"
        for num in range(1,8):
            for c in combinations(range(10), num):
                if c in good_c:
                    print c
                    part_train_data = train_data[np.array(c)]
                    am = AM(part_train_data)

DN = part_train_data.shape[0]
for err_n in range(ERR_RANGE): # error number
    err_missing = 0.0
    err_mistake = 0.0
    for i in range(TEST_LOOP):
        for d in part_train_data:
            missing_d= disturbe(d, err_n, mode="missing")
            mistake_d= disturbe(d, err_n, mode="mistake")
            missing_pre = am.predict(missing_d)[1] # using u
            mistake_pre = am.predict(mistake_d)[1] # using u
            if VERBOSE:
                display(d)

```

```

display(missing_d)
display(mistake_d)
display(missing_pre)
display(mistake_pre)
if not np.array_equal(missing_pre, d):
    err_missing += 1
if not np.array_equal(mistake_pre, d):
    err_mistake += 1
err_rate_missing = err_missing / (TEST_LOOP * D_N)
err_rate_mistake = err_mistake / (TEST_LOOP * D_N)
print "When_err_is", err_n, ",_missing_ERR_rate:", err_ra

#####
# Third question #
#####
# if True:
if False:
    print "*" * 80
    print "#_second_question_#"
    def spurious(pre, train_data): # given a predict, judge if it's in the s
        for d in train_data:
            if np.array_equal(d, pre):
                return False
        return True
    # c= (1, 4, 6, 7, 9)
    # part_train_data = train_data[np.array(c)]
    # am = AM(part_train_data)
    set_p = set()
    total_n = 0

    # find out the spurious patterns
    for num in range(1, 8):
        for c in combinations(range(10), num):
            if c in good_c:
                # print c
                part_train_data = train_data[np.array(c)]
                am = AM(part_train_data)
                # pre = am.predict_converge(part_train_data)[0]
                pre = am.predict(part_train_data)[1] # using w0 to predict
                total_n += pre.shape[0]
                for p in pre:
                    p.reshape(35,)
                    if spurious(p, part_train_data):
                        t = tuple(p)
                        set_p.add(t)

```

```

        # show these spurious patterns
        for sp in set_p:
            display(np.array(sp))
        print len(set_p)
        print total_n

print "*" * 80

```

1.2 Some Explanation

Run Environment I also included a source file of my program in the zipped file. If you want to run it, please make sure that you have numpy installed. I am using Python 2.7 in this project. Also Anaconda is highly recommended.

Parameters There are some parameters in this program that one can adjust to see the result. I put them on the beginning of the source code and make sure they are well commented so that you can know the meaning for every parameter.

Toggle There are three main questions so there are three corresponding parts in the “main” part of the program. And before every part, there is a “if” statement to let you decide whether you want to run this part. Every part is indepent with each other. By default, they will all run.

Run Time On my machine(Macbook Pro 13, early 2015), the prgram need about half a minute to finish. This may be different from 10s to 5m I guess.

Others Because the output will be a lot, I suggest to use the following command to redirect the output:

```
python am.py | tee output.txt
```

2 Analysis and Answers

2.1 Program

I wrote a class called AM to do the job. It can predict in two policies: direct predict and predict until we get to a converge or run out of loop times. Also you can choose to use w or w0, w is the weight matrix that diagonal elements all untouched, w0 is the weight matrix that diagonal elements are all set to 0. Which means I implemented 4 kinds of predict methods:(direct, w), (direct, w0), (converge, w), (converge, w0). You can see in the Q1 part code, I tried all of 4 kinds of prediction.

display() can help to display the patterns for you to recognize.
disturbe() is to help make noise into the data.

2.2 Q1

The output for Q1 is in a manner that “method,numbers,weight”.

“method” maybe direct or converge. Direct means predict just use $w * input$, “converge” means predict iteratively until converge or reach the maximum loop times.

“numbers” is the working combination of the patterns.

“weight” may be “p” or “p0”. “p” means we get this result by using the weight matrix with digonal elements stay untouched. “p0” means we get this result by using the weght matrix with all diagonal elemets set 0.

As we can see, there are 456 kinds of combinations that can be stored and retrived smoothly. Thus the maximum capacity of AM in this problem is 5.

The full results of combinations that can be stored is attached.

2.3 Q2

For this question, I just do the experiments under those patterns that can be stored. And I implemented two kinds of noise: the first one is that missing, which turn some points in the pattern into 0, the second one is mistake, which is to flip some points in the pattern.

Observe the result, we can conclude that:

1. When number of errors increased, error rate increased.
2. When number of patterns stored increased, the number of errors that it can tolerate decreased.
3. When error number is big enough, the network will only give out wrong result.

2.4 Q3

For this question, I just do the experiments under those patterns that can be stored. And I implemented using direct predict with w_0 . Typically, the spurious patterns looks like some of the train patterns very much, just with some points flipped or missing. For example, there are some spurious patterns(1 is #, -1 is space, 0 is ?) in the following figure.

In the number 9, some points are missing. In the number 2, some points are flipped(from 1 to -1), in the number 4, some points which should be -1 are missing.

###		###		#
#	#		#	#
#	#		#	?#
###?		#		#
#		#		#####
#				? ?#
####		#####		#

####		#####		###
#	?		#	#
#			#	#
####		#	##	##
#	#		#	#
#	#		#	#
###		#####		###