НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:
«Лабораторная работа №15»

Выполнил: Осипов Лев,
студент группы БПИ121 (1).

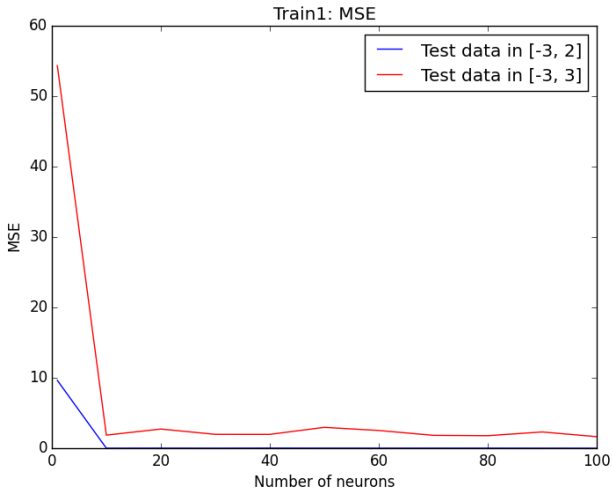
Москва, 2015 г.

СОДЕРЖАНИЕ

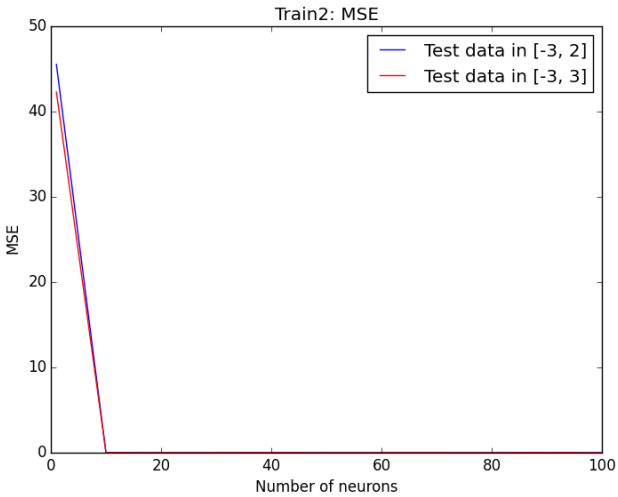
ПРАКТИЧЕСКАЯ ЧАСТЬ

ЗАДАНИЕ 1

Была построена и обучена нейронная сеть, аппроксимирующая значения функции $f(x) = x^3$. Задания заключались в измерении средней квадратичной ошибки в зависимости от числа нейронов в скрытом слое (измерения проводились на двух тестовых выборках – [-3, 2] и [-3, 3]) и в визуализации аппроксимации (тоже с различным количеством нейронов в скрытом слое, на одной тестовой выборке – [-3, 3]). Оба задания проводились на двух различных обучающих выборках – [-3, 2] и [-3, 1]∪[2,3]. Результаты измерений ошибок:



Зависимость ошибки от количества нейронов с обучающей выборкой [-3, 2]



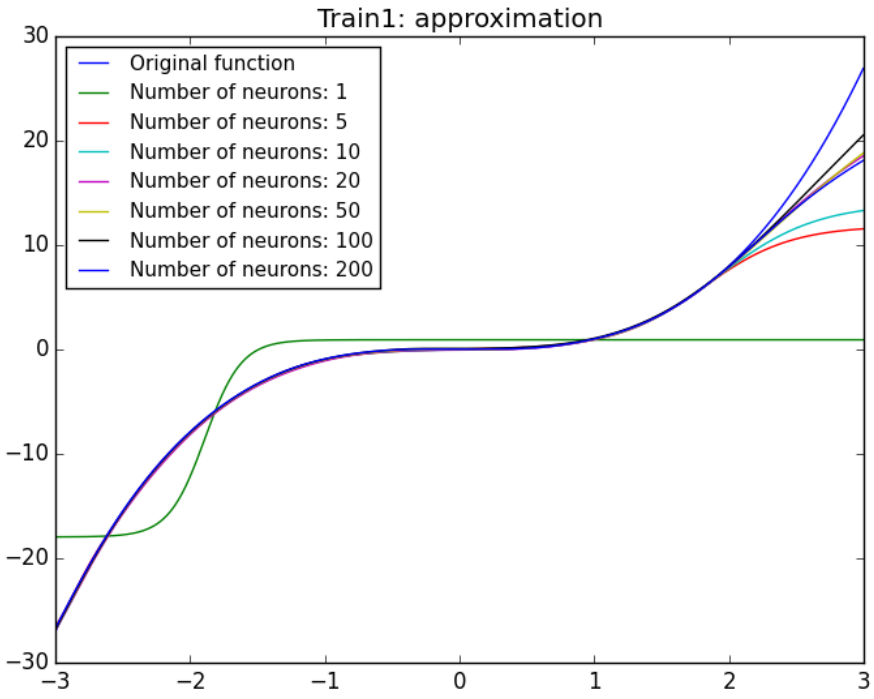
Зависимость ошибки от количества нейронов с обучающей выборкой [-3, 1]∪[2, 3]

По графикам видно, что для адекватной аппроксимации нужно приблизительно 10 нейронов в скрытом слое.

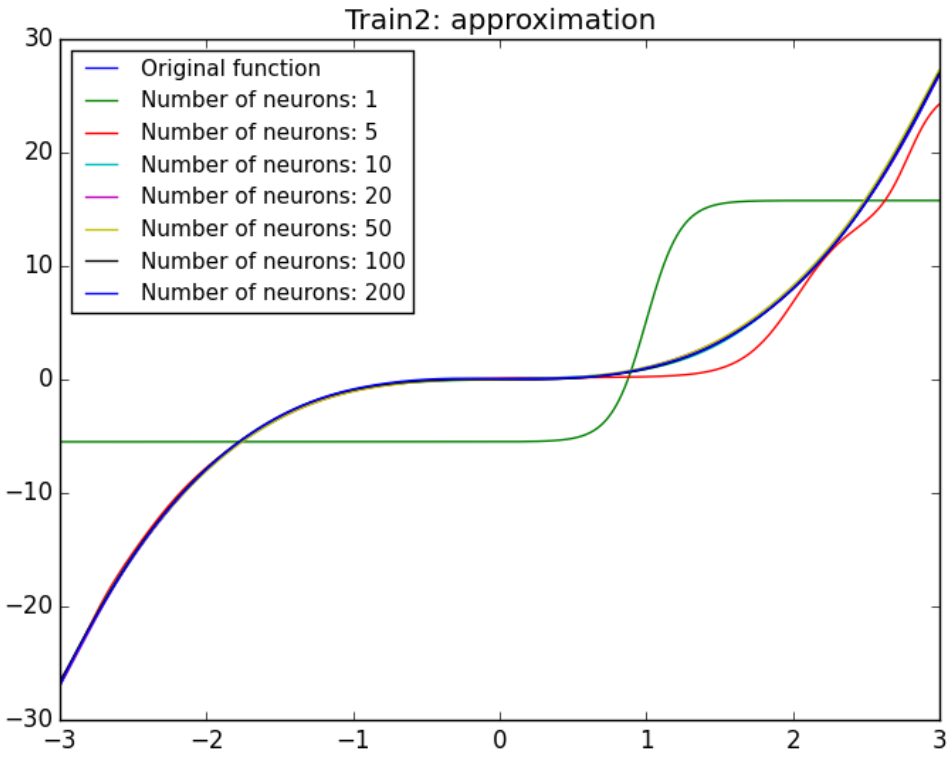
Что касается различий в ошибках, видно, что в начале первого графика, где тестовая выборка совпадала с обучающей, ошибок было мало даже при одном нейроне. В втором же случае, когда обучающая выборках лежала в диапазоне, отличном от обеих тестовых выборок, ошибок было примерно равное количество.

Что касается динамики ошибок после некой «стабилизации» графиков, видно, что с обучающей выборкой, охватывающей диапазон значений с обеих сторон (с пробелом посередине), ошибок было меньше, чем когда обучающая выборка брала значения только «с одной стороны»

Результаты аппроксимации:



Результаты аппроксимации с обучающей выборкой [-3, 2]



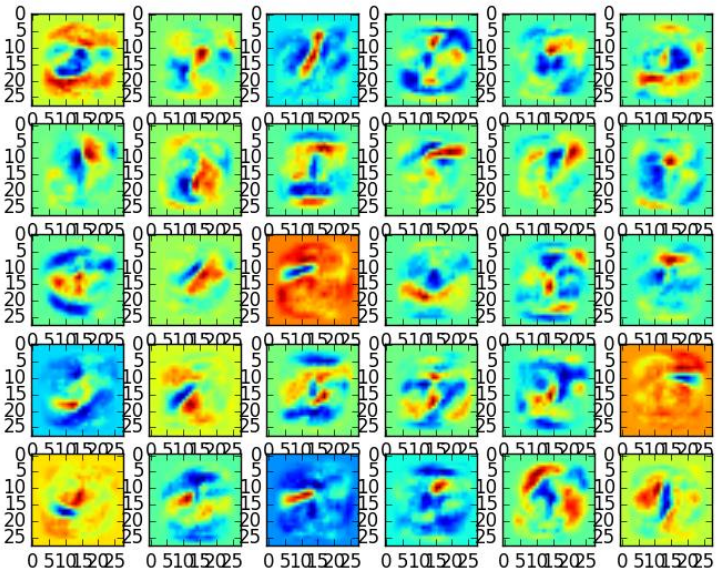
Результаты аппроксимации с обучающей выборкой [-3, 1]∪[2, 3]

Визуализация аппроксимации подтверждает, что для нормально аппроксимации с такими входными данными хватает 10 нейронов в скрытом слое нейросети.

ЗАДАНИЕ 2

Для классификации была построена и обучена нейронная сеть.

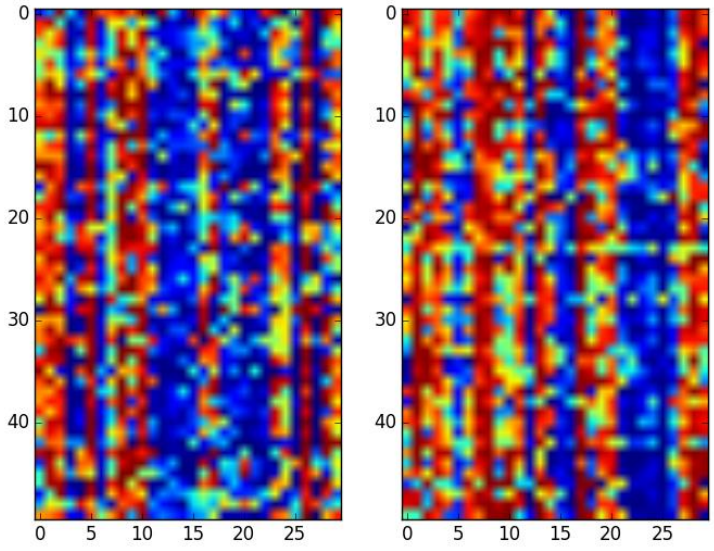
Визуализация весов связей сети входного слоя со скрытым слоем:



Визуализация весов связей

Можно предположить, что эти данные имеют какое-либо отношение к составным частям будущих цифр, а уже из них формируются выходные требования для цифр каждого класса.

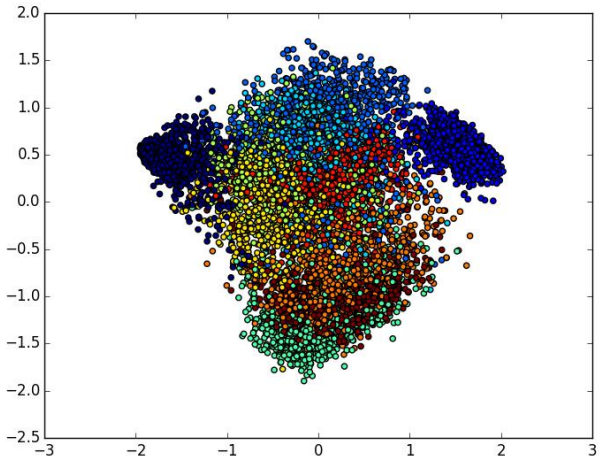
Далее, из данных, получившихся из выходных значений скрытого слоя, для 50 объектов 2-х классов были построены матрицы (строки – объекты, столбцы – новые признаки). Было решено взять матрицы для цифр «5» и «7», так как они имеют существенные различия в структуре:



Новые признаки объектов классов «5» (слева) и «7» (справа)

Объекты разных классов можно сравнить визуально, используя вновь получившиеся признаки. Объекты класса «5» отличаются значением несколько признаков, отчего в матрице объектов «5» (слева) выделяется еще одна синяя «полоса». В общем, визуальная разница заметна.

Далее, объекты были визуализированы с помощью PCA:



PCA

Видно, что данные не особо компактны. Это можно объяснить тем, что объекты достаточное неточные (насколько я понял, это те же данные, что были в предыдущих работах, а там по визуализации было видно, что они отображены достаточно нечетко и могут быть перепутаны друг с другом).

Далее, были проведены измерения точности классификации различных алгоритмов (нейронная сеть, kNN из старых признаков и kNN из признаков, полученных из выходных значений скрытого слоя):

```
Network score: 0.9454
KNN score: 0.90
KNN new score: 0.969
```

Результаты точности для трех алгоритмов

Видно, что самым точным оказался метод ближайших соседей, основанный на промежуточных результатах нейронной сети. Возможно, именно комбинация этих двух алгоритмов повлияла на самый высокий результат.

По поводу данной задачи следует сказать, что нужно брать в расчет ожидаемую точность и время вычислений. Если требуется большая точность – стоит подумать о различных комбинациях алгоритмов, иначе можно воспользоваться простым поиском ближайших соседей.

СПИСОК ЛИТЕРАТУРЫ

1) **Анализ данных (Программная инженерия)** – http://wiki.cs.hse.ru/Анализ_данных_(Программная_инженерия)

ТЕКСТ ПРОГРАММЫ

```
__author__ = 'Lev Osipov'


import numpy as np
from pybrain.datasets import SupervisedDataSet
from pybrain.datasets import ClassificationDataSet
from pybrain.tools.shortcuts import buildNetwork
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure import LinearLayer
from pybrain.structure import TanhLayer
from pybrain.structure import SigmoidLayer
from pybrain.structure import SoftmaxLayer
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from pybrain.utilities import percentError
from sklearn.neighbors import KNeighborsClassifier


from pybrain.tools.validation import ModuleValidator


def train_network_1(hidden_count, train_data):
    network = buildNetwork(1, hidden_count, 1,
hiddenclass=TanhLayer, outclass=LinearLayer)

    trainer = BackpropTrainer(network, train_data,
weightdecay=0.00001)
    trainer.trainEpochs(100)

    return network


def check_error(hidden_count, train_data, test_data_1,
test_data_2):
    network = train_network_1(hidden_count, train_data)

    error1 = ModuleValidator.MSE(network, test_data_1)
    error2 = ModuleValidator.MSE(network, test_data_2)

    return error1, error2


def approximate(hidden_count, train_data, test_data):
    network = train_network_1(hidden_count, train_data)

    results = np.zeros(shape=len(test_data))
    for i in xrange(len(test_data)):
        results[i] = network.activate(test_data['input'][i])

    return results
```

```
def do_checking_and_approximation(name, train_data, test_data_1,
test_data_2):
    errors1 = np.zeros(shape=11)
    errors2 = np.zeros(shape=11)
    numbs = [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
    for (i, hidden_count) in enumerate(numbs):
        error1, error2 = check_error(hidden_count, train_data,
test_data_1, test_data_2)
        errors1[i] = error1
        errors2[i] = error2
        print name + ': counted errors with ' +
str(hidden_count) + ' neurons'

    plt.title(name + ': MSE')
    plt.plot(numbs, errors1, 'b', label='Test data in [-3, 2]')
    plt.plot(numbs, errors2, 'r', label='Test data in [-3, 3]')
    plt.xlabel('Number of neurons')
    plt.ylabel('MSE')
    plt.legend()
    plt.savefig(name + '_MSE.png')
    plt.clf()

    numbs = [1, 5, 10, 20, 50, 100, 200]
    plt.plot(test_data_2['input'], test_data_2['target'],
label='Original function')
    for (i, hidden_count) in enumerate(numbs):
        results = approximate(hidden_count, train_data,
test_data_2)
        lbl = 'Number of neurons: ' + str(hidden_count)
        plt.plot(test_data_2['input'], results, label=lbl)
        print name + ': approximated with ' + str(hidden_count)
+ ' neurons'

    plt.title(name + ': approximation')
    plt.legend(loc='upper left', prop={'size': 11})
    plt.savefig(name + '_approx.png')
    plt.clf()


def task1():

    objects = np.arange(-3, 2, 5.0 / 2000)
    train_data = SupervisedDataSet(1, 1)
    for i in xrange(len(objects)):
        train_data.appendLinked(objects[i], objects[i] ** 3)

    objects = np.arange(-3, 2, 5.0 / 2000)
    test_data_1 = SupervisedDataSet(1, 1)
    for i in xrange(len(objects)):
        test_data_1.appendLinked(objects[i], objects[i] ** 3)

    objects = np.arange(-3, 3, 6.0 / 2000)
```

```
    test_data_2 = SupervisedDataSet(1, 1)
    for i in xrange(len(objects)):
        test_data_2.appendLinked(objects[i], objects[i] ** 3)

    do_checking_and_approximation('Train1', train_data,
test_data_1, test_data_2)

    temp1 = np.arange(-3, 1, 4.0 / 1600)
    temp2 = np.arange(2, 3, 1.0 / 400)
    objects = np.concatenate([temp1, temp2])
    train_data = SupervisedDataSet(1, 1)
    for i in xrange(len(objects)):
        train_data.appendLinked(objects[i], objects[i] ** 3)

    do_checking_and_approximation('Train2', train_data,
test_data_1, test_data_2)


def train_network_2(train_data):
    network = buildNetwork(784, 30, 10,
hiddenclass=SigmoidLayer, outclass=SoftmaxLayer)

    trainer = BackpropTrainer(network, train_data,
weightdecay=0.001)
    trainer.trainEpochs(25)

    return network, trainer


def show_connections(network):
    connections = []
    for cons in network.connections.values():
        for cs in cons:
            if cs.inmod.name == 'in' and cs.outmod.name ==
'hidden0':
                connections = cs
    connections = np.reshape(connections.params, (30, 28, 28))

    for i in xrange(len(connections)):
        plt.subplot(5, 6, i + 1)
        plt.imshow(connections[i])
    plt.savefig('connections.png')
    plt.clf()


def get_hidden_values(network, obj):
    network.activate(obj)
    values =
network['hidden0'].outputbuffer[network['hidden0'].offset]
    return values


def show_differences(network, test_data):
```

```
    objects_a = []
    objects_b = []
    for i in xrange(len(test_data)):
        if test_data['class'][i] == 5 and len(objects_a) < 50:
            objects_a.append(test_data['input'][i])
        if test_data['class'][i] == 7 and len(objects_b) < 50:
            objects_b.append(test_data['input'][i])
    new_objects_a = np.zeros(shape=(len(objects_a), 30))
    for i in xrange(len(objects_a)):
        new_objects_a[i] = get_hidden_values(network,
objects_a[i])

    new_objects_b = np.zeros(shape=(len(objects_b), 30))
    for i in xrange(len(objects_b)):
        new_objects_b[i] = get_hidden_values(network,
objects_b[i])

    plt.subplot(1, 2, 1)
    plt.imshow(new_objects_a)
    plt.subplot(1, 2, 2)
    plt.imshow(new_objects_b)
    plt.savefig("differences.png")
    plt.clf()


def create_new_data(network, data):
    new_train_data = np.zeros(shape=(len(data), 30))
    for i in xrange(len(data)):
        new_train_data[i] = get_hidden_values(network,
data['input'][i])
    return new_train_data


def show_pca(data, labels):
    pca = PCA(n_components=2)
    pca.fit(data)
    data = pca.transform(data)
    plt.scatter(data[:, 0], data[:, 1], c=labels)
    plt.savefig('pca.png')
    plt.clf()


def show_scores(trainer, train_data, test_data, new_train_data,
new_test_data):
    network_error =
percentError(trainer.testOnClassData(dataset=test_data),
test_data['class'])
    print 'Network score: ' + str((100 - network_error) / 100)

    knn1 = KNeighborsClassifier(n_neighbors=3)
    knn1.fit(train_data['input'][:50000],
np.reshape(train_data['class'][:50000], (50000,)))
    knn_score = knn1.score(test_data['input'][:5000],
```

```
np.reshape(test_data['class'][:5000], (5000,)))
    print 'KNN score: ' + str(knn_score)

    knn2 = KNeighborsClassifier(n_neighbors=3)
    knn2.fit(new_train_data[:50000],
np.reshape(train_data['class'][:50000], (50000,)))
    knn_new_score = knn2.score(new_test_data[:5000],
np.reshape(test_data['class'][:5000], (5000,)))
    print 'KNN new score: ' + str(knn_new_score)


def task2():
    train_data =
ClassificationDataSet.loadFromFile('mnist_train')
    test_data = ClassificationDataSet.loadFromFile('mnist_test')

    network, trainer = train_network_2(train_data)

    show_connections(network)

    show_differences(network, test_data)

    new_train_data = create_new_data(network, train_data)
    new_test_data = create_new_data(network, test_data)

    show_pca(new_test_data, test_data['class'])

    show_scores(trainer, train_data, test_data, new_train_data,
new_test_data)


task1()
task2()
```