Министерство высшего образования и науки РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

**Отчет по лабораторной работе**

по дисциплине "Интеллектуальные информационные системы"

на тему: "Нейросети"

Выполнил

студенты гр. 3530902/70201

\_\_\_\_\_\_\_\_\_\_\_ Ю. Ю. Мансуров

Проверил

Доцент, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_ С. В. Хлопин

дата, подпись

Санкт-Петербург

2020

Содержание

Введение 4

1 Нейросеть 5

1.1 Описание 5

1.2 Математическая модель 6

1.3 Обучение нейросети 7

2 Реализация 9

2.1 Архитектура 9

2.2 Интерфейс 10

2.3 Пример работы программы 10

3 Исследование 12

3.1 Задание 12

3.2 Подготовка данных 12

3.3 Обучение нейросетей 13

3.4 Анализ зависимостей 18

3.4.1 Зависимость от количества слоев 18

3.4.2 Зависимость от количества нейронов на слое 20

3.4.3 Зависимость от размера тренировочного набора 21

3.4.4 Зависимость от вида распределения 22

3.4.5 Зависимость от количества эпох 25

3.5 Результаты обучения 28

Вывод 31

Приложение 1. Код класса, реализующего нейросеть 33

Приложение 2. Список команд пользователя 37

Приложение 3. Ошибки сетей для всех попыток 39

Приложение 4. Код скрипта для построения графиков 41

# Введение

В ходе данной работы требуется создать многослойную нейронную сеть и обучить ее для интерполяции периодической функции двух переменных. Требуется исследовать зависимость качества работы нейросети от количества слоев в сети и количества нейронов в слое. Кроме этого, требуется исследовать зависимость качества аппроксимации от количества тренировочных случаев и их распределения в области.

Для выполнения задания была написана нейронная сеть на языке C# на платформе .NET.

## Нейросеть

### Описание

Нейросеть является объектом, который для любых допустимых входных данных возвращает некоторые выходные данные. Многослойная нейросеть состоит из входного, выходного и любого количества внутренних слоев.

Каждый слой состоит из нейронов, хранящих некоторое значение. Значения входных нейронов являются входными данными нейросети и устанавливаются извне. Каждый нейрон, кроме входных, представляет собой объект, вычисляющий взвешенную сумму значений нейронов предыдущего слоя, к которой затем применяется функция активации. Также на каждом слое кроме выходного, используется нейрон сдвига, всегда имеющий значение 1 и участвующий при расчете нейронов следующего уровня.

Функция активации – некоторая математическая функция от одной переменной, определенная на всем множестве действительных чисел. В данной работе в качестве функции активации была взята логистическая функция .

При прямом проходе нейросети задаются значения входных нейронов, а затем послойно вычисляются значения нейронов на остальных слоях. Значения нейронов последнего слоя являются выходными данными.

При создании нейросети начальные веса для каждого нейрона задаются случайно (обычно от -1 до 1). Для того, чтобы сеть работала корректно ее необходимо обучить. В данной работе рассматривается только обучение с учителем. При этом используются подготовленные тренировочные данные, состоящие из входных данных и правильных выходных данных. Во время обучения веса нейронов изменяются так, чтобы уменьшить ошибку между выходными данными нейросети и правильными выходными данными. Для этого обычно используется метод обратного распространения ошибки, основанный на методе градиентного спуска.

При применении метода обратного распространения ошибки используется понятие пакета обучения. Пакет – это часть тренировочного набора, состоящая из определенного количества элементов. Веса нейросети корректируются после прохождения всего пакета. Использование пакетов позволяет сделать стремление весов к оптимальным более плавным, однако использование пакетов слишком большого размера может привести к тому, что веса нейросети примут локально оптимальные значения, которые не будут являться глобально оптимальными (веса окажутся в локальном минимуме функции зависимости ошибки нейросети от весов).

Обычно одного прохода по тренировочному набору недостаточно, чтобы хорошо обучить нейросеть, поэтому нейросеть обучается на этом наборе несколько раз. Один проход по тренировочному набору называется эпохой.

Тестирование нейросети производится на некотором наборе данных, для которого вычисляется ошибка между выходными данными нейросети и правильными выходными данными. Ошибка может вычисляться различными способами.

### Математическая модель

С математической точки зрения многослойная нейронная сеть представляет собой функционал, переводящий вектор входных данных в вектор выходных данных . Данный функционал можно описать следующей формулой

где

Здесь – оператор применения функции активации к каждому элементу вектора, – матрица весов, – вектор сдвигов. При этом , а также .

При известных и можно рассчитать выходное значение нейросети для любых входных данных. Из-за использования функции активации область значений нейросети оказывается ограничена. Областью определения нейросети (область допустимых входных значений) является все множество векторов .

### Обучение нейросети

Обучение нейросети производится на некотором наборе обучающих случаев вида , где – правильные выходные данные, соответствующие *x*.

Рассмотрим алгоритм обратного распространения ошибки:

1. Для каждого нейрона последнего (выходного) слоя вычисляется производная
2. Для всех нейронов внутренних слоев начиная с последнего вычисляется производная

где сумма проводится по всем нейронам следующего уровня. Соответственно – производная нейрона, – вес связи между нейроном текущего уровня и нейроном следующего уровня.

1. Для всех слоев, кроме входного, вычисляется изменение весов и сдвигов  
   где – скорость обучения, – значение нейрона текущего слоя, – значение производной нейрона следующего слоя.
2. Пункты 1-3 выполняются для всех элементов пакета.
3. Корректируются веса нейросети

здесь – значение весов для текущего пакета, – сумма изменений весов для текущего пакета, – значение весов для следующего пакета

1. Пункты 4-5 выполняются для всех пакетов данной эпохи
2. Пункт 6 выполняется для всех эпох

Пункты 1-3 алгоритма можно записать в векторно-матричной форме для логистической функции активации. При этом . Используем следующие обозначения: – матрица весов, – вектор сдвигов, – вектор значений нейронов, , – вектор входных данных, – вектор правильных выходных данных.

Здесь знак обозначает операцию поэлементного умножения векторов. 1 – вектор необходимого размера, состоящий из единиц.

## Реализация

В ходе выполнения работы была написана программа на языке C# на платформе .NET.

### Архитектура

Программа состоит из пяти частей (модулей), каждая из которых находится в своем пространстве имен. Информация о частях программы приведена в таблице 1.

Таблица 1. Модули программы

|  |  |
| --- | --- |
| **Пространство имен** | **Описание модуля** |
| MatrixNeuralNetwork | Содержит класс многослойной нейросети MatrixNN и класс набора данных CasesSet |
| MatrixNeuralNetwork.FileStorage | Содержит функции для экспорта нейросети или набора данных в текстовый файл; импорта нейросети или набора данных из файла; записи информации о изменении ошибки нейросети на каждой эпохе |
| NeuralNetwork\_Console.Function | Содержит функции для создания случайного или равномерного (сеточного) тренировочного (тестового) набора данных для заданной функции; линейного преобразования пространства значений функции в пространство значений нейросети и обратно |
| NeuralNetwork\_Console.Interface | Содержит интерфейс командной строки с обработчиком команд пользователя |
| NeuralNetwork\_Console.Models | Предоставляет функциональный интерфейс взаимодействия с не интерфейсными модулями |

Приложение использует бесконечный цикл для обработки команд пользователя. Для получения команд пользователя используется интерфейс командной строки.

Исходный код приложения доступен в Github репозитории по ссылке:

* https://github.com/kiarro/NeuralNetwork\_Console

Код класса, реализующего нейросеть находится в приложении 1.

### Интерфейс

Приложение использует интерфейс командной строки, поэтому пользователь управляет работой программы с помощью текстовых команд. Весь список команд приведен в приложении 2.

### Пример работы программы

Примеры работы программы показаны на следующих рисунках. На Рисунок 1 показан пример создания и удаления сетей и наборов данных. Также показаны команды вывода списка сетей и списка наборов.

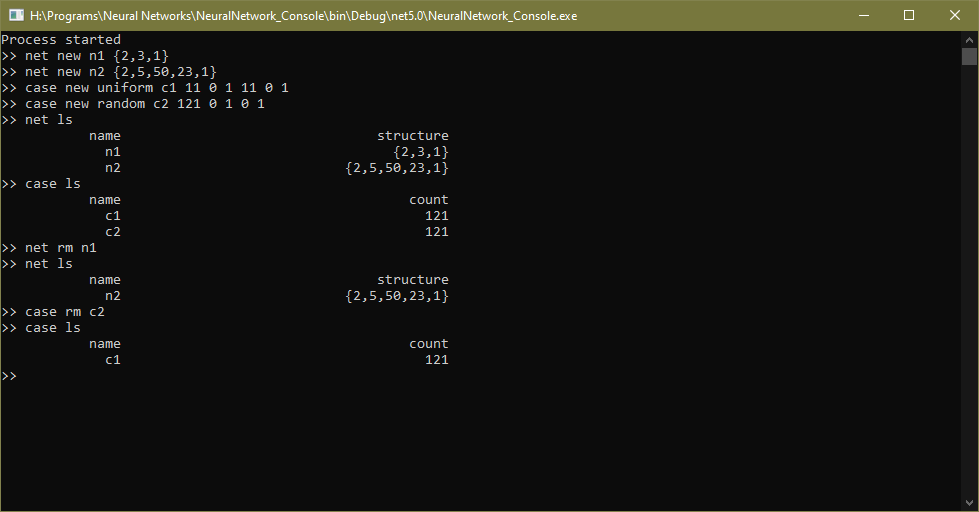


Рисунок 1 – Пример работы команд создания, удаления и вывода списка

На рисунке 2 показан пример работы команд для обучения и тестирования нейросетей. На рисунке 3 показан пример использования команды для обучения (train) с различными параметрами.



Рисунок 2 - Пример работы команд тестирования и обучения

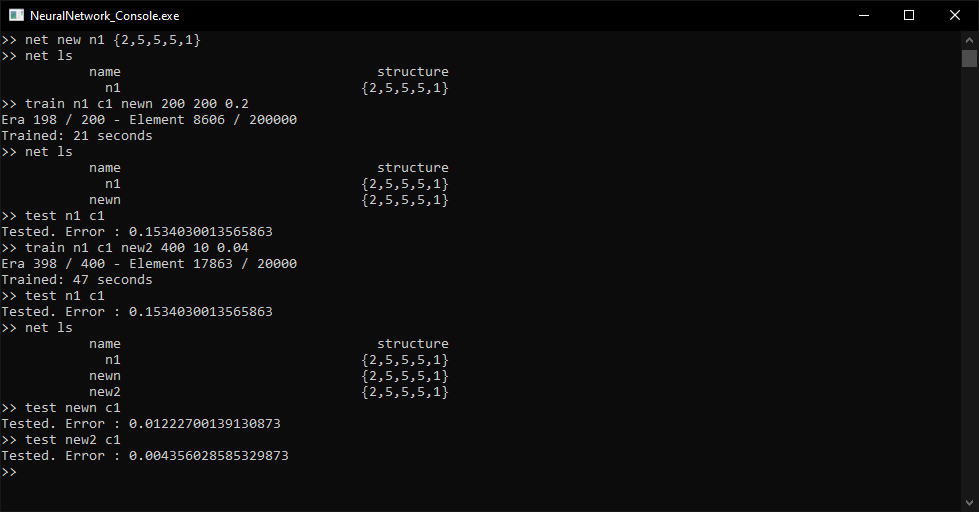


Рисунок 3 – Пример работы команды обучения с параметрами

## Исследование

### Задание

В ходе работы необходимо с помощью нейросети интерполировать функцию

.

В качестве области, на которой будет обучаться и тестироваться сеть, была выбрана область .

Требуется определить зависимость качества работы нейросети от следующих параметров:

* количества внутренних слоев
* количества нейронов во внутренних слоях
* объема обучающей выборки
* распределения точек, на которых обучается сеть, в области
* количества эпох

Качество работы нейросети определяется по ошибке на тестовом наборе данных

где – результат работы нейросети для *i*-й точки тестового набора, – правильный результат для *i*-й точки тестового набора.

### Подготовка данных

Для исследования были созданы наборы данных с сеточным распределением с различным шагом сетки. Для каждого набора с сеточным распределением был создан набор со случайным распределением, содержащий такое же количество точек. Это требуется, чтобы определить качество обучения сети от вида распределения тренировочных данных. Оба вида распределения являются равномерными. Таблица 2 содержит информацию о созданных наборах данных.

Таблица 2. Наборы данных

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Вид распределения | Шаг распределения | Количество элементов |
| c10 | сеточное | 0.6 | 121 |
| c20 | сеточное | 0.3 | 441 |
| c40 | сеточное | 0.15 | 1681 |
| c80 | сеточное | 0.075 | 6561 |
| c160 | сеточное | 0.0375 | 25921 |
| r10 | случайное | - | 121 |
| r20 | случайное | - | 441 |
| r40 | случайное | - | 1681 |
| r80 | случайное | - | 6561 |
| r160 | случайное | - | 25921 |

В таблице для сеточных наборов данных указан шаг сетки, одинаковый по обеим осям. Для случайно распределенных наборов данных шаг не может быть указан ввиду случайности выбора точек из области .

Наборы распределенных по сетке данных содержат точки, упорядоченные сначала по оси X, а затем по оси Y.

### Обучение нейросетей

В ходе эксперимента использовались нейросети с двумя входными нейронами и одним выходным нейроном. Это необходимо, чтобы количество входных и выходных данных нейросети соответствовало их количеству для моделируемой функции. Были использованы нейросети с различным количеством внутренних слоев (1, 2, 3 и 4), а также различным количеством нейронов на внутренних слоях (5, 10, 20, 30 и 40). На всех внутренних слоях было использовано одинаковое количество нейронов. Всего было использовано 20 нейросетей с различной структурой.

При инициализации нейросети есть вероятность, что веса окажутся неудачными, т.е. нейросеть будет невозможно обучить на некотором наборе. Чтобы уменьшить вероятность получения такой неудачной нейросети, для каждой структуры нейросеть инициализировалась и обучалась 10 раз. После из 10 полученных нейросетей с одинаковой структурой, обученных на одном и том же наборе данных, была выбрана сеть с наилучшей точностью (наименьшей ошибкой) при тестировании на равномерном наборе данных c160.

Обучение нейросетей проводилось с использованием следующих параметров:

* скорость обучения
* размер пакета – 100
* количество эпох – 100

При обучении данные из тренировочных наборов не перемешивались.

В результате были отобраны лучшие сети, ошибка которых указана в таблице 3.

Таблица 3. Ошибка нейросетей

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Сеть \ Набор** | **c10** | **c20** | **c40** | **c80** | **c160** | **r10** | **r20** | **r40** | **r80** | **r160** |
| n5 | 0,053063 | 0,063911 | 0,061238 | 0,061428 | 0,091257 | 0,048976 | 0,049089 | 0,034833 | 0,03133 | 0,04073 |
| n5\_5 | 0,058196 | 0,052079 | 0,046875 | 0,048386 | 0,124859 | 0,049194 | 0,047907 | 0,039011 | 0,033121 | 0,032652 |
| n5\_5\_5 | 0,0603 | 0,051046 | 0,046222 | 0,051897 | 0,145518 | 0,060363 | 0,05945 | 0,053344 | 0,03966 | 0,030104 |
| n5\_5\_5\_5 | 0,060902 | 0,064058 | 0,060141 | 0,049943 | 0,161613 | 0,060474 | 0,060158 | 0,058333 | 0,051195 | 0,029565 |
| n10 | 0,056021 | 0,063957 | 0,061772 | 0,061049 | 0,097052 | 0,048334 | 0,036296 | 0,050905 | 0,030248 | 0,027272 |
| n10\_10 | 0,060884 | 0,056272 | 0,04923 | 0,037574 | 0,112233 | 0,055776 | 0,06135 | 0,033193 | 0,011659 | 0,006351 |
| n10\_10\_10 | 0,060842 | 0,063936 | 0,052368 | 0,039561 | 0,131443 | 0,057383 | 0,058744 | 0,04375 | 0,018794 | 0,006689 |
| n10\_10\_10\_10 | 0,060897 | 0,062856 | 0,052665 | 0,047673 | 0,152725 | 0,060278 | 0,061326 | 0,049279 | 0,023792 | 0,006641 |
| n20 | 0,059078 | 0,05679 | 0,06098 | 0,091725 | 0,130641 | 0,067359 | 0,037994 | 0,039189 | 0,03014 | 0,034822 |
| n20\_20 | 0,057343 | 0,059362 | 0,039943 | 0,055692 | 0,105252 | 0,051607 | 0,043716 | 0,024265 | 0,006954 | 0,00092 |
| n20\_20\_20 | 0,067665 | 0,062769 | 0,047566 | 0,056958 | 0,161564 | 0,056977 | 0,057528 | 0,028947 | 0,004616 | 0,000469 |
| n20\_20\_20\_20 | 0,065709 | 0,065379 | 0,054498 | 0,03977 | 0,143087 | 0,059796 | 0,059487 | 0,041904 | 0,007507 | 0,001082 |
| n30 | 0,07804 | 0,065456 | 0,062537 | 0,059593 | 0,112417 | 0,091956 | 0,041294 | 0,030542 | 0,030966 | 0,033564 |
| n30\_30 | 0,056461 | 0,06398 | 0,049255 | 0,036015 | 0,089166 | 0,287025 | 0,037622 | 0,025482 | 0,005415 | 0,000572 |
| n30\_30\_30 | 0,054664 | 0,058243 | 0,058221 | 0,054178 | 0,142887 | 0,309387 | 0,044059 | 0,08681 | 0,005543 | 0,000413 |
| n30\_30\_30\_30 | 0,30987 | 0,309856 | 0,057461 | 0,069886 | 0,16319 | 0,187271 | 0,064953 | 0,038834 | 0,002735 | 0,002085 |
| n40 | 0,186248 | 0,135298 | 0,06942 | 0,072756 | 0,125705 | 0,125729 | 0,090887 | 0,035218 | 0,031233 | 0,038393 |
| n40\_40 | 0,308586 | 0,082205 | 0,094958 | 0,078824 | 0,177787 | 0,285394 | 0,116245 | 0,038855 | 0,009087 | 0,001583 |
| n40\_40\_40 | 0,083492 | 0,146188 | 0,174284 | 0,309978 | 0,309974 | 0,309499 | 0,302884 | 0,031643 | 0,002992 | 0,000322 |
| n40\_40\_40\_40 | 0,309362 | 0,309978 | 0,30998 | 0,309982 | 0,309981 | 0,281342 | 0,072814 | 0,038769 | 0,00104 | 0,000651 |

Можно видеть, что некоторые сети имеют ошибку, сильно отличающуюся от ошибок нейросетей близких по структуре и обучающему набору. Это может быть вызвано неудачной инициализацией весов сети или видом обучающего набора. Будем считать удачными сети с ошибкой менее 0,28.

Можно видеть (Таблица 4), что такие неудачные значения характерны для сетей с большим количеством нейронов на слое. Кроме того, они чаще встречаются при обучении на сеточном распределении, из чего следует, что при обучении на последовательных сеточных данных выше вероятность попадания в локальный минимум.

Таблица 4. Неудачные нейросети

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Сеть \ Набор** | **c10** | **c20** | **c40** | **c80** | **c160** | **r10** | **r20** | **r40** | **r80** | **r160** |
| n5 | 0,053063 | 0,063911 | 0,061238 | 0,061428 | 0,091257 | 0,048976 | 0,049089 | 0,034833 | 0,03133 | 0,04073 |
| n5\_5 | 0,058196 | 0,052079 | 0,046875 | 0,048386 | 0,124859 | 0,049194 | 0,047907 | 0,039011 | 0,033121 | 0,032652 |
| n5\_5\_5 | 0,0603 | 0,051046 | 0,046222 | 0,051897 | 0,145518 | 0,060363 | 0,05945 | 0,053344 | 0,03966 | 0,030104 |
| n5\_5\_5\_5 | 0,060902 | 0,064058 | 0,060141 | 0,049943 | 0,161613 | 0,060474 | 0,060158 | 0,058333 | 0,051195 | 0,029565 |
| n10 | 0,056021 | 0,063957 | 0,061772 | 0,061049 | 0,097052 | 0,048334 | 0,036296 | 0,050905 | 0,030248 | 0,027272 |
| n10\_10 | 0,060884 | 0,056272 | 0,04923 | 0,037574 | 0,112233 | 0,055776 | 0,06135 | 0,033193 | 0,011659 | 0,006351 |
| n10\_10\_10 | 0,060842 | 0,063936 | 0,052368 | 0,039561 | 0,131443 | 0,057383 | 0,058744 | 0,04375 | 0,018794 | 0,006689 |
| n10\_10\_10\_10 | 0,060897 | 0,062856 | 0,052665 | 0,047673 | 0,152725 | 0,060278 | 0,061326 | 0,049279 | 0,023792 | 0,006641 |
| n20 | 0,059078 | 0,05679 | 0,06098 | 0,091725 | 0,130641 | 0,067359 | 0,037994 | 0,039189 | 0,03014 | 0,034822 |
| n20\_20 | 0,057343 | 0,059362 | 0,039943 | 0,055692 | 0,105252 | 0,051607 | 0,043716 | 0,024265 | 0,006954 | 0,00092 |
| n20\_20\_20 | 0,067665 | 0,062769 | 0,047566 | 0,056958 | 0,161564 | 0,056977 | 0,057528 | 0,028947 | 0,004616 | 0,000469 |
| n20\_20\_20\_20 | 0,065709 | 0,065379 | 0,054498 | 0,03977 | 0,143087 | 0,059796 | 0,059487 | 0,041904 | 0,007507 | 0,001082 |
| n30 | 0,07804 | 0,065456 | 0,062537 | 0,059593 | 0,112417 | 0,091956 | 0,041294 | 0,030542 | 0,030966 | 0,033564 |
| n30\_30 | 0,056461 | 0,06398 | 0,049255 | 0,036015 | 0,089166 | 0,287025 | 0,037622 | 0,025482 | 0,005415 | 0,000572 |
| n30\_30\_30 | 0,054664 | 0,058243 | 0,058221 | 0,054178 | 0,142887 | 0,309387 | 0,044059 | 0,08681 | 0,005543 | 0,000413 |
| n30\_30\_30\_30 | 0,30987 | 0,309856 | 0,057461 | 0,069886 | 0,16319 | 0,187271 | 0,064953 | 0,038834 | 0,002735 | 0,002085 |
| n40 | 0,186248 | 0,135298 | 0,06942 | 0,072756 | 0,125705 | 0,125729 | 0,090887 | 0,035218 | 0,031233 | 0,038393 |
| n40\_40 | 0,308586 | 0,082205 | 0,094958 | 0,078824 | 0,177787 | 0,285394 | 0,116245 | 0,038855 | 0,009087 | 0,001583 |
| n40\_40\_40 | 0,083492 | 0,146188 | 0,174284 | 0,309978 | 0,309974 | 0,309499 | 0,302884 | 0,031643 | 0,002992 | 0,000322 |
| n40\_40\_40\_40 | 0,309362 | 0,309978 | 0,30998 | 0,309982 | 0,309981 | 0,281342 | 0,072814 | 0,038769 | 0,00104 | 0,000651 |

Приложение 3 содержит в табличной форме информацию об ошибках некоторых нейросетей для всех 10 попыток. Также в этой таблице указано минимальное и максимальное значение ошибки для данной сети и количество удачных сетей. На следующих рисунках показано количество удачных сетей для различных сетей при обучении на различных наборах. По оси Ox отложено количество нейронов на слое нейросети, по оси Oy отложено количество внутренних слоев.



Рисунок 4 – Удачные сети при обучении на c80

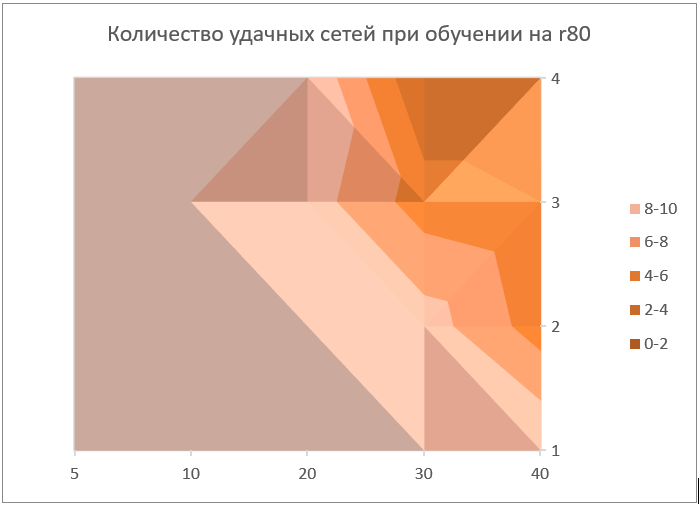


Рисунок 5 – Удачные сети при обучении на r80

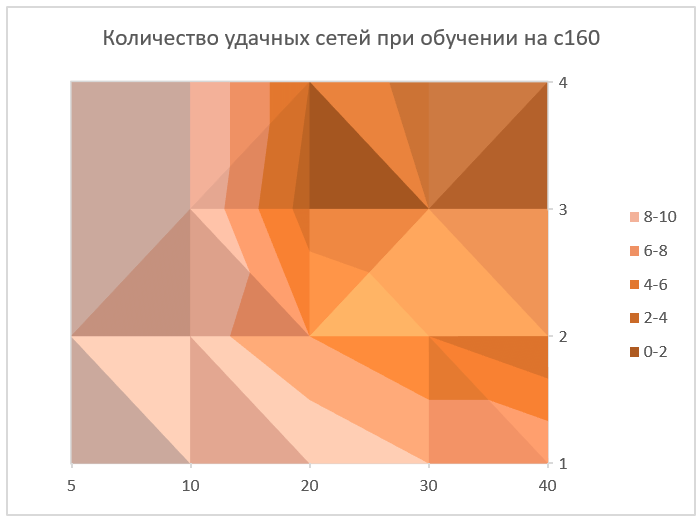


Рисунок – Удачные сети при обучении на c160

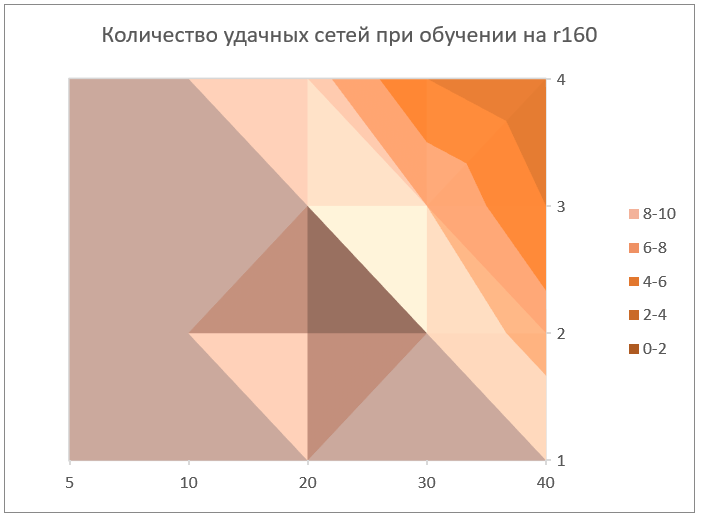


Рисунок – Удачные сети при обучении на r160

Можно видеть, что чем больше слоев и нейронов содержит нейросеть, тем меньше шансов получить удачные начальные веса.

### Анализ зависимостей

Для анализа зависимостей качества работы нейросети от различных параметров, указанных в задании, используем данные из Таблица 3.

#### Зависимость от количества слоев

Построим графики зависимости ошибки нейросети от количества слоев для различного количества нейронов на слое и различных наборов данных для обучения.

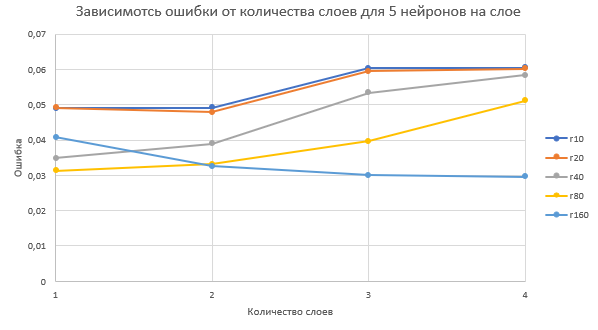
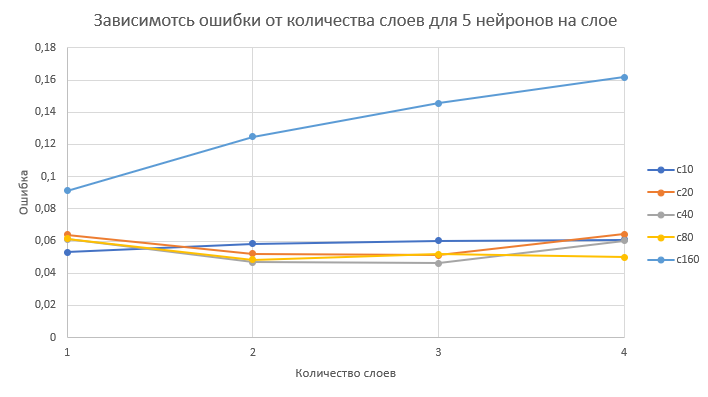


Рисунок – Зависимость ошибки от количества слоев для 5 нейронов на слое

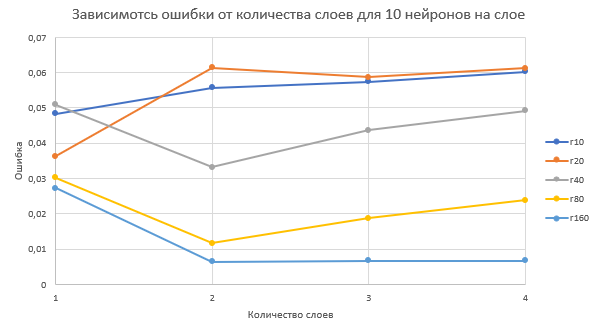
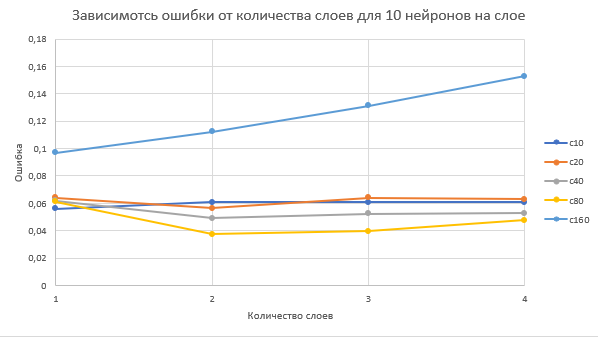


Рисунок – Зависимость ошибки от количества слоев для 10 нейронов на слое

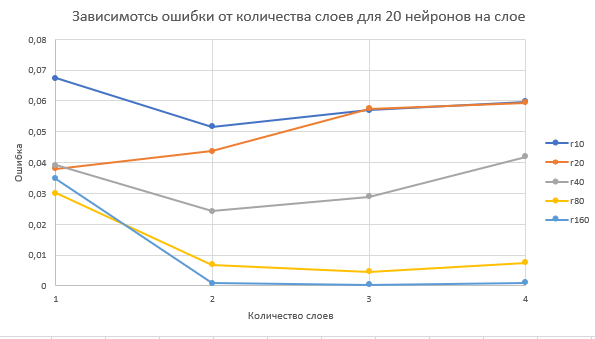
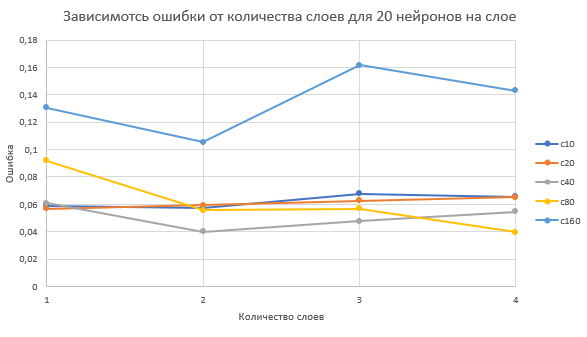


Рисунок – Зависимость ошибки от количества слоев для 20 нейронов на слое

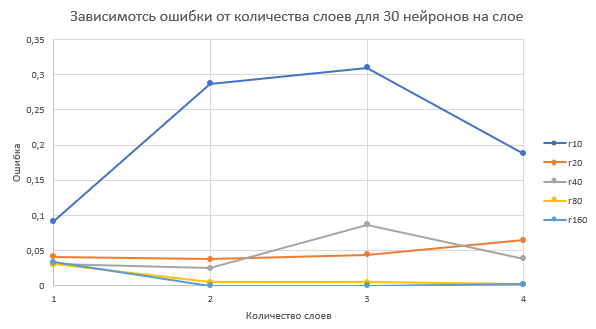
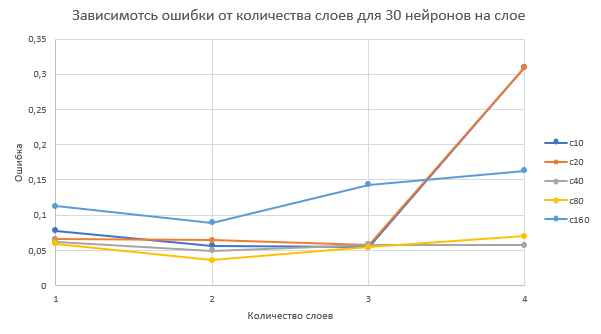


Рисунок – Зависимость ошибки от количества слоев для 30 нейронов на слое

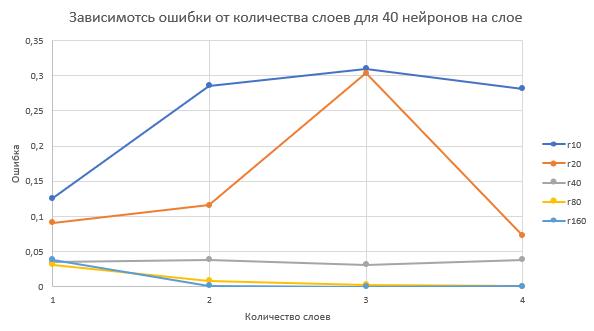
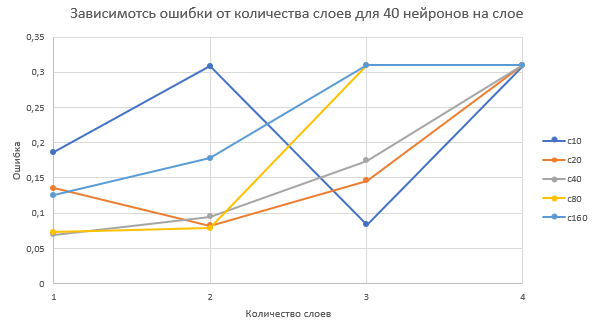


Рисунок – Зависимость ошибки от количества слоев для 40 нейронов на слое

По данным графикам можно сделать вывод, что при увеличении количества слоев точность нейросети может как улучшаться, так и ухудшаться.

При обучении на сеточном распределении сети с малым количеством нейронов на слое изменение количества слоев немного улучшает точность нейросети. Ухудшение точности сети с большим количеством нейронов на слое (30 и 40) при увеличении числа слоев может быть вызвано переобучением сети для последних элементов набора за счет ухудшения точности для первых элементов набора. Это предположение подтверждается тем, что при обучении на случайно распределенном наборе, ухудшение точности происходит значительно реже.

При обучении на случайном распределении точность сети в большинстве случаев улучшается. Ухудшение точности может быть вызвано неудачной инициализацией сети. Можно видеть, что чем больше размер набора, тем сильнее улучшается точность сети.

Также видно, что значительное улучшение точности происходит при увеличении количества слоев с 1 до 2. Дальнейшее увеличение числа слоев почти не влияет на точность в лучшую сторону.

#### Зависимость от количества нейронов на слое

Построим графики зависимости ошибки нейросети от количества нейронов на слое для различного количества слоев и различных наборов данных для обучения.

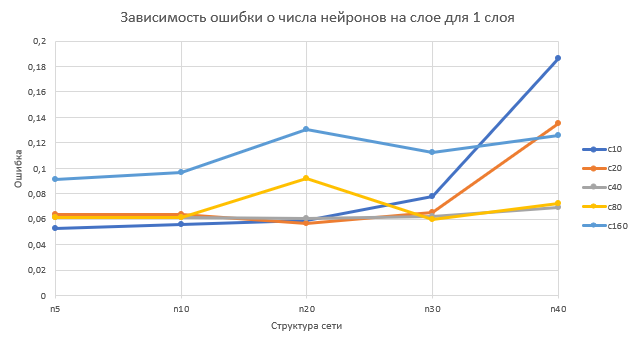
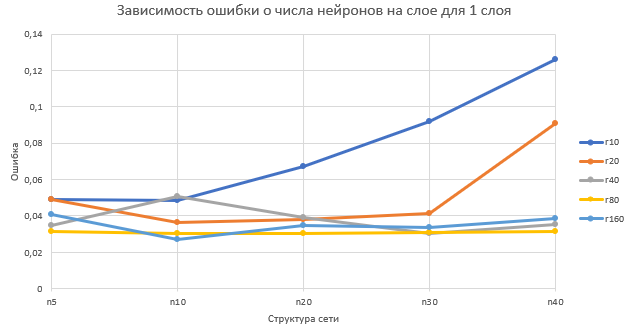
 

Рисунок – Зависимость ошибки от количества нейронов на слое для 1 слоя

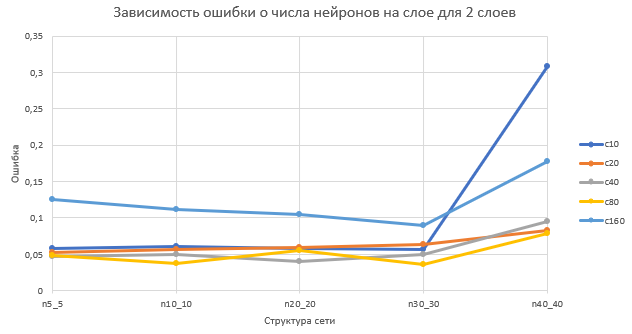
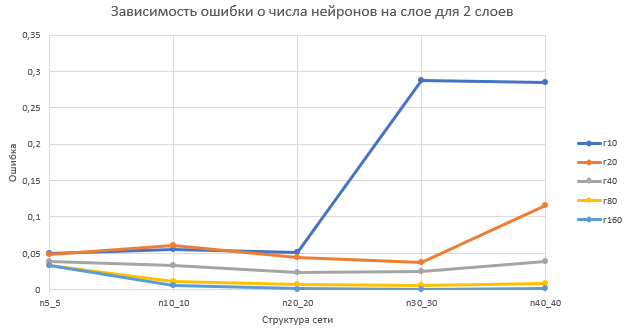
 

Рисунок – Зависимость ошибки от количества нейронов на слое для 2 слоев

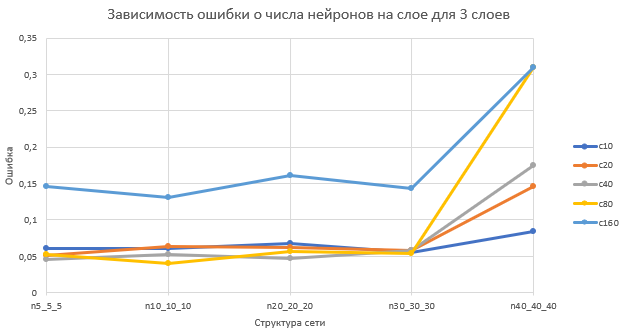
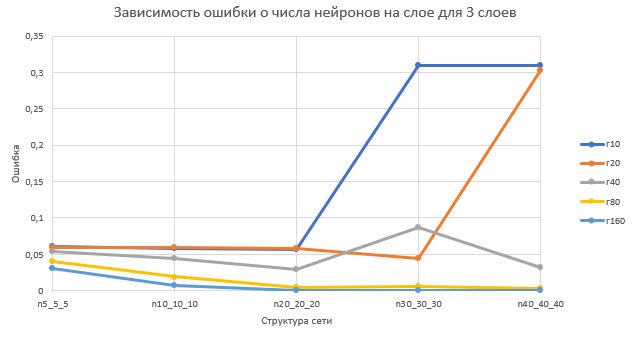
 

Рисунок – Зависимость ошибки от количества нейронов на слое для 3 слоев

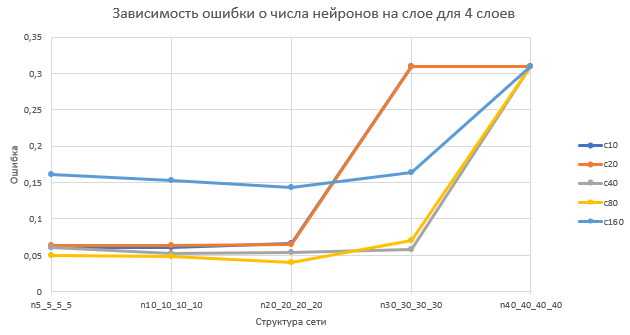
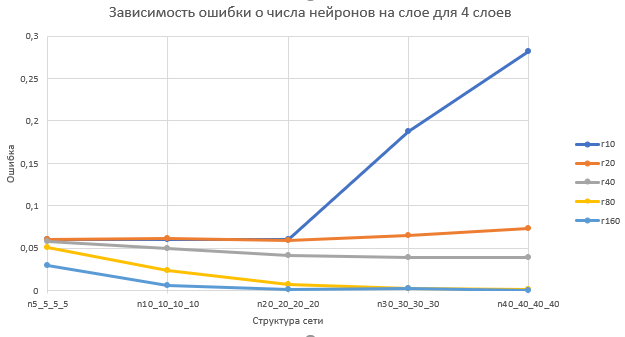
 

Рисунок – Зависимость ошибки от количества нейронов на слое для 4 слоев

По данным графикам можно сделать вывод, что при увеличении количества нейронов на слое точность нейросети может как улучшаться, так и ухудшаться.

При обучении на сеточном распределении увеличение количества нейронов на слое либо почти не влияет на точность сети, либо сильно ухудшает точность (в случае, если получается неудачная сеть).

При обучении на случайном распределении увеличение количества нейронов на слое в большинстве случаев улучшает или не изменяет точность сети. Можно видеть, что чем больше размер набора, тем сильнее улучшается точность сети.

#### Зависимость от размера тренировочного набора

Построим графики зависимости ошибки нейросети от размера тренировочного набора для различных нейросетей.

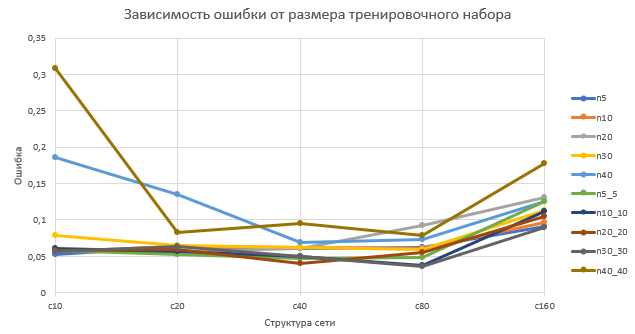
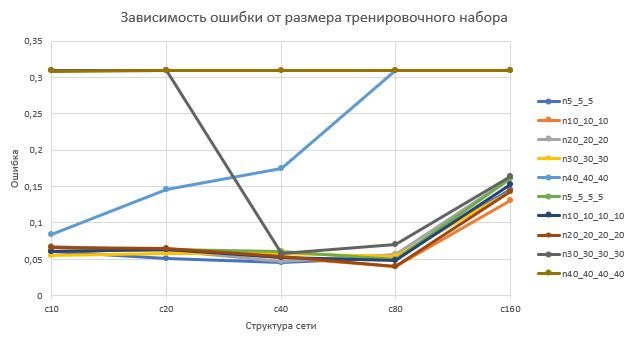
 

Рисунок – Зависимость ошибки от размера сеточного набора

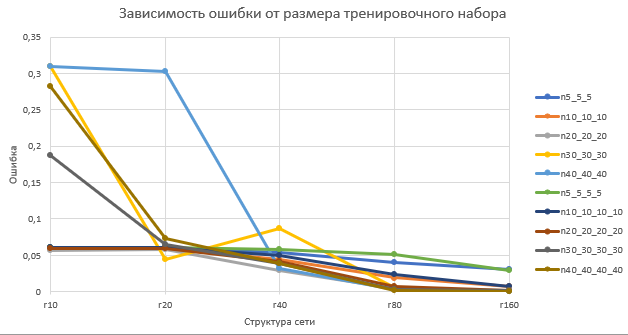
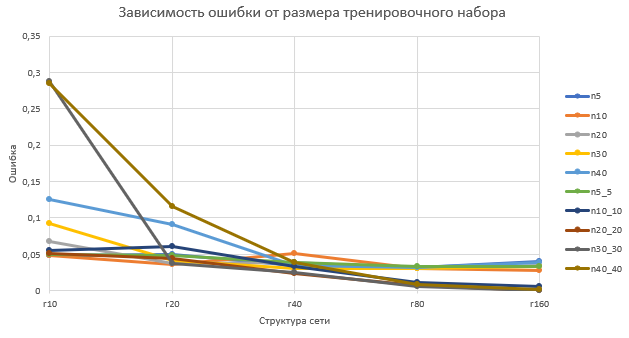


Рисунок – Зависимость ошибки от размера случайного набора

По данным графикам можно сделать вывод, что при увеличении размера сеточного набора точность нейросети сначала улучшается, а затем ухудшается. Это может быть вызвано тем, что при обучении на упорядоченном сеточном наборе сеть переобучается для последних точек набора, в то время как для первых точек набора точность уменьшается.

При увеличении размера случайного набора точность нейросети в общем улучшается для всех рассмотренных нейросетей. Колебания точности для некоторых нейросетей можно объяснить случайностью начальных весов сети.

#### Зависимость от вида распределения

Построим гистограммы ошибки нейросети для сравнения двух видов распределения тренировочного набора для различных нейросетей.



Рисунок 19 – Зависимость ошибки от вида распределения для набора размера 121

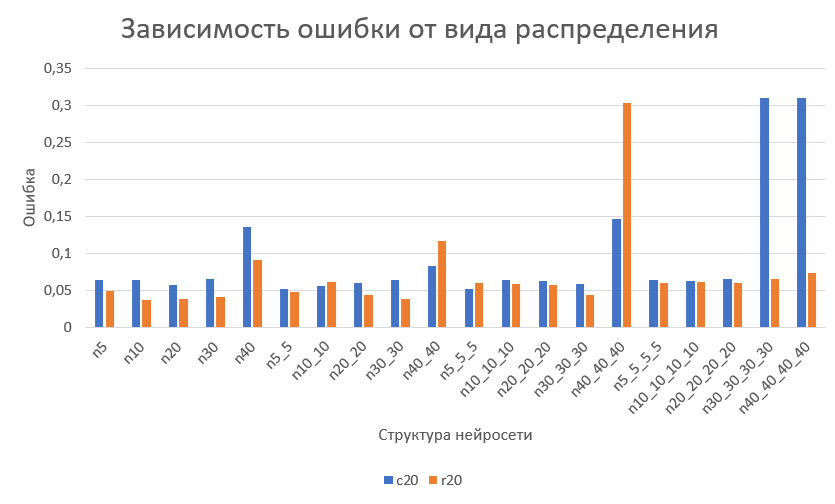


Рисунок 20 – Зависимость ошибки от вида распределения для набора размера 441



Рисунок 21 – Зависимость ошибки от вида распределения для набора размера 1681



Рисунок 22 – Зависимость ошибки от вида распределения для набора размера 6561



Рисунок 23 – Зависимость ошибки от вида распределения для набора размера 25921

По данным гистограммам видно, что для наборов размером 121 и 441 элемент разница между обучением на случайном и сеточном распределениях незначительна. Для больших по объему наборов случайное распределение имеет преимущество перед сеточным. Видно, что при увеличении объема набора увеличивается преимущество случайного распределения.

#### Зависимость от количества эпох

При вычислении ошибки нейросети на каждой эпохе использовался тот же набор данных, что и при обучении, поэтому ошибки в данном разделе могут отличаться от ошибок, приведенных в других разделах. Не смотря на использование тренировочного набора для тестирования, полученные показатели ошибки позволяют исследовать зависимость точности нейросети от количества пройденных эпох.

Построим графики зависимости ошибки от количества пройденных эпох для некоторых нейросетей.

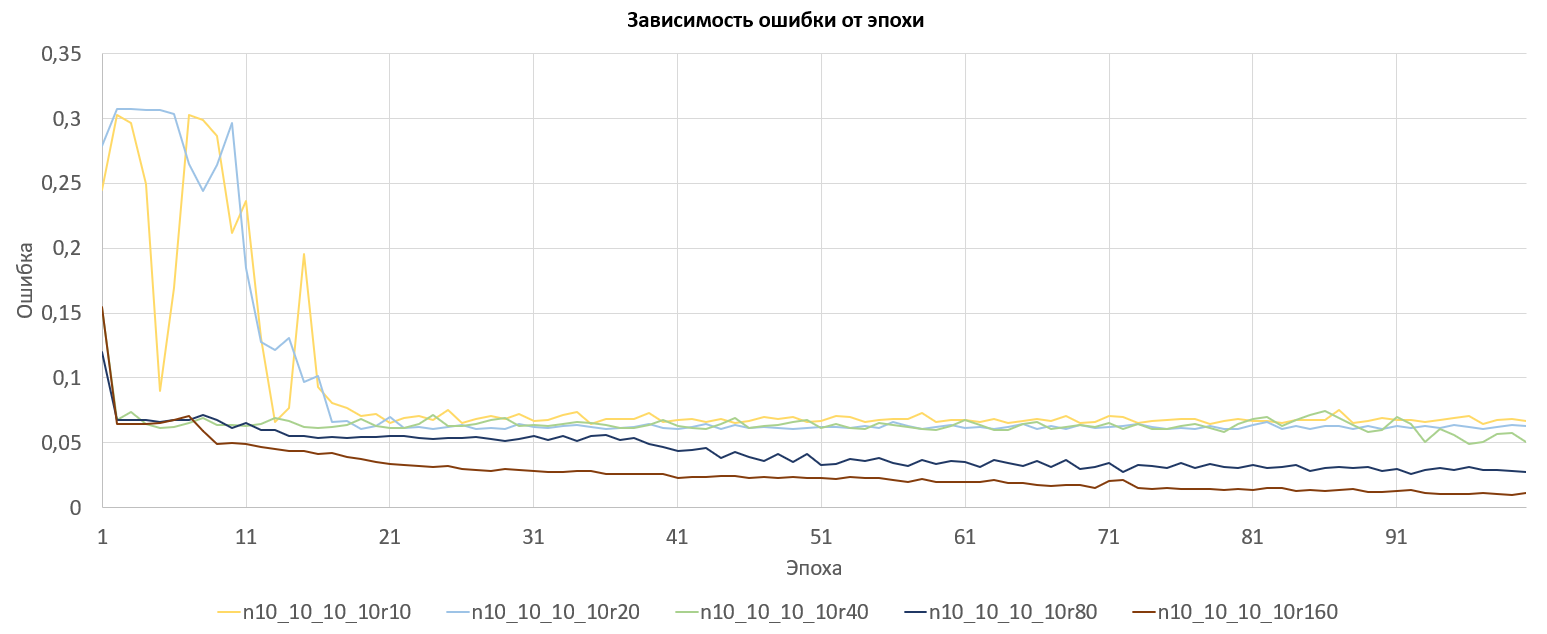


Рисунок 24 – Зависимость ошибки от эпохи для некоторых сетей

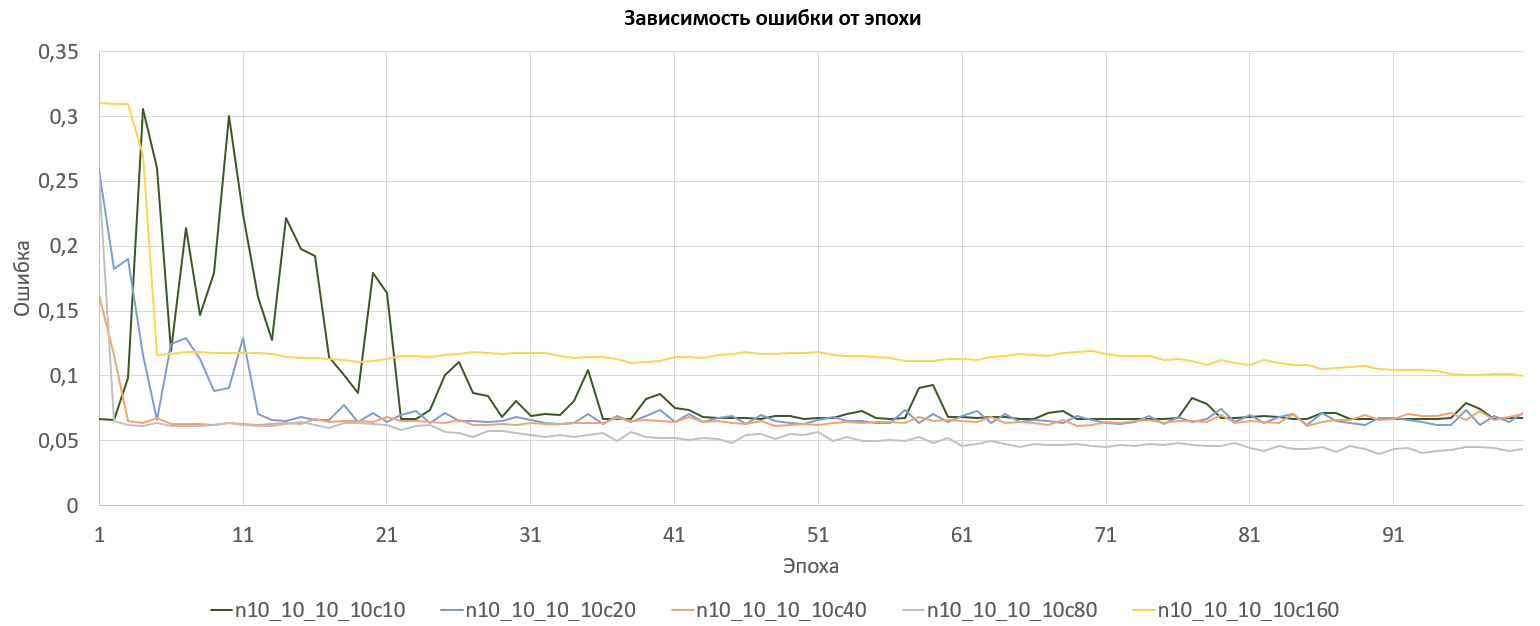


Рисунок 25 – Зависимость ошибки от эпохи для некоторых сетей

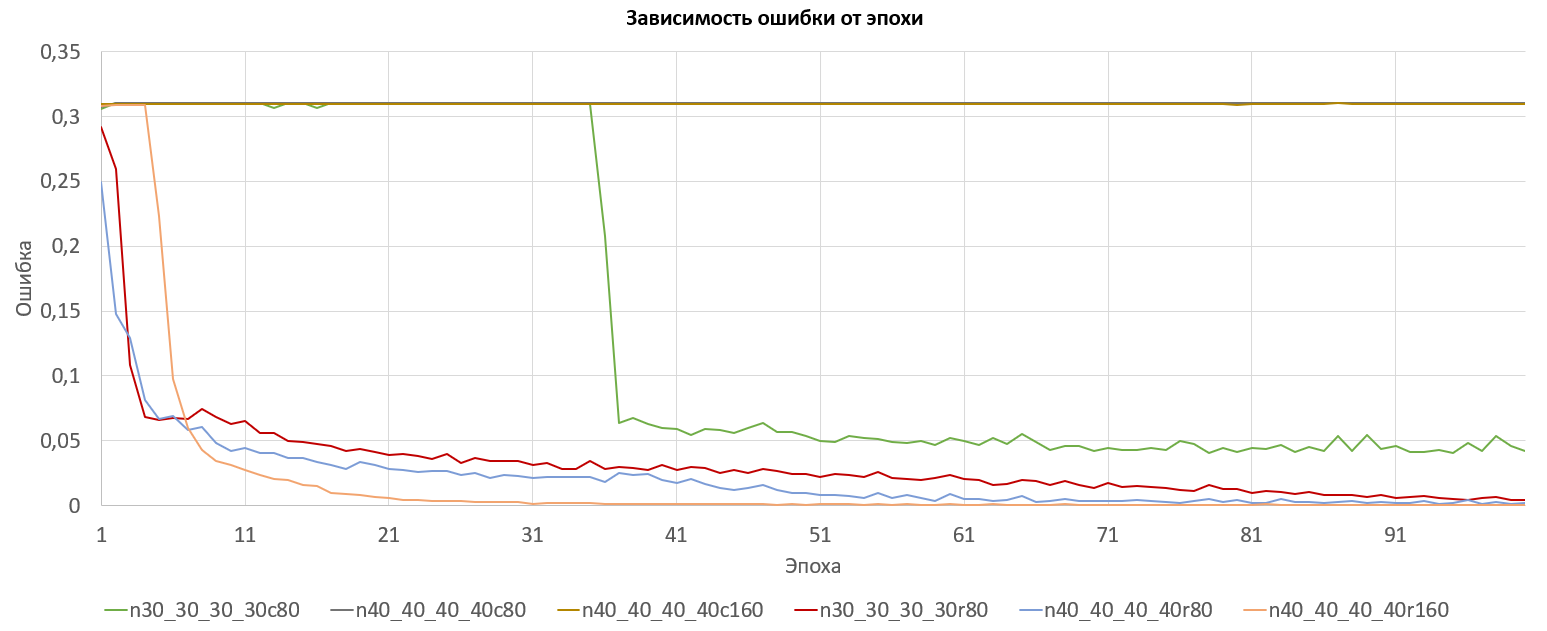


Рисунок 26 – Зависимость ошибки от эпохи для некоторых сетей

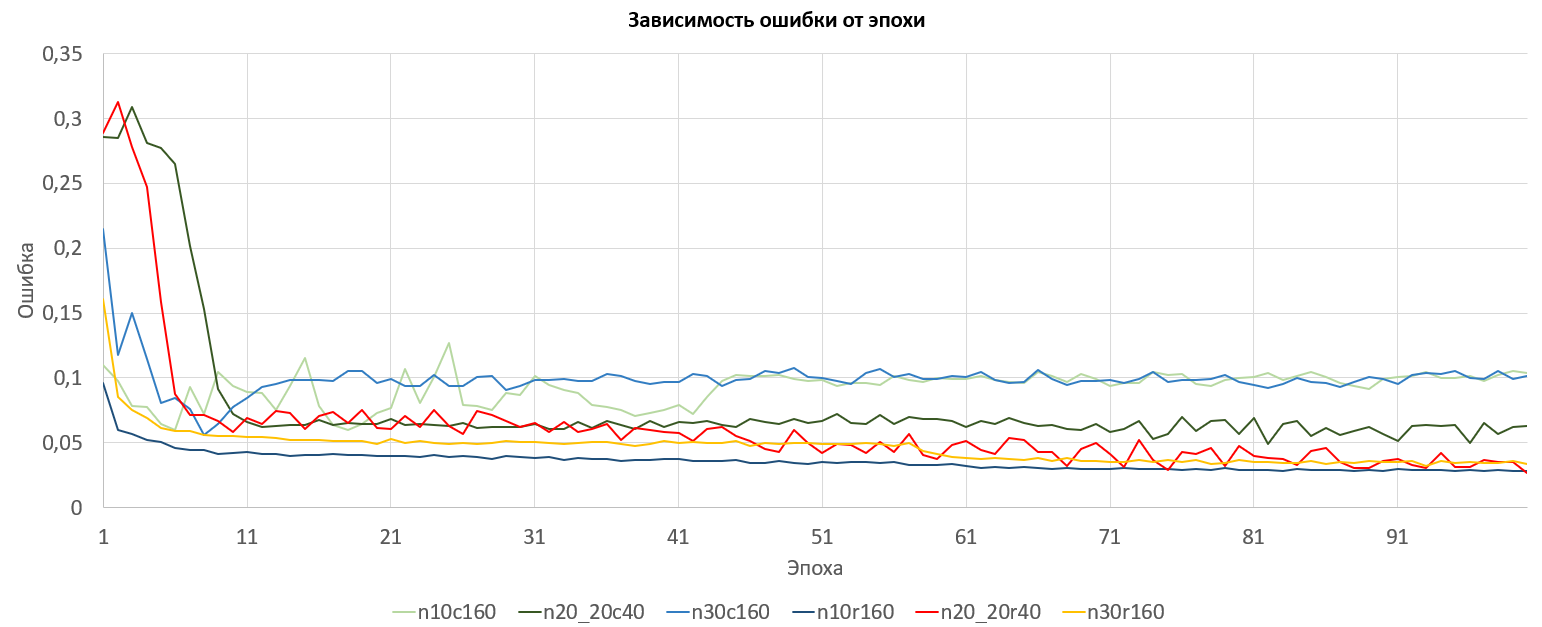


Рисунок 27 – Зависимость ошибки от эпохи для некоторых сетей

По данным графикам видно, что для большинства нейросетей ошибка резко уменьшается в первые несколько эпох (примерно 10-15), после чего либо плавно продолжает уменьшаться (например, сети n30\_30\_30\_30r80, n10\_10\_10\_10c60), либо совершает небольшие колебания в окрестности устоявшегося значения (например, сети n30\_30\_30\_30с80, n10\_10\_10\_10c10, n20\_20c40).

График зависимости, построенный для сети n30\_30\_30\_30c80, показывает, что ошибка может сохраняться значительное количество эпох без изменения, после чего за несколько эпох (примерно 3-4) резко уменьшается до устоявшегося значения (или близкого к нему).

График зависимости, построенный для сети n10\_10\_10\_10c10, показывает, что ошибка может вырасти в начале процесса обучения (первые 10-20 эпох), а затем снова снизиться до изначального уровня или ниже. Это может происходить из-за большой скорости обучения, из-за чего веса изменяются слишком сильно, чтобы оказаться в минимуме.

Для сети n40\_40\_40\_40c160 ошибка держится на высоком уровне на протяжении всех 100 эпох, что можно объяснить неудачными начальными весами или переобучением сети в части области обучения.

### Результаты обучения

Графики, построенные на основе нейросетей должны быть похожи на график заданной функции (Рисунок 28).

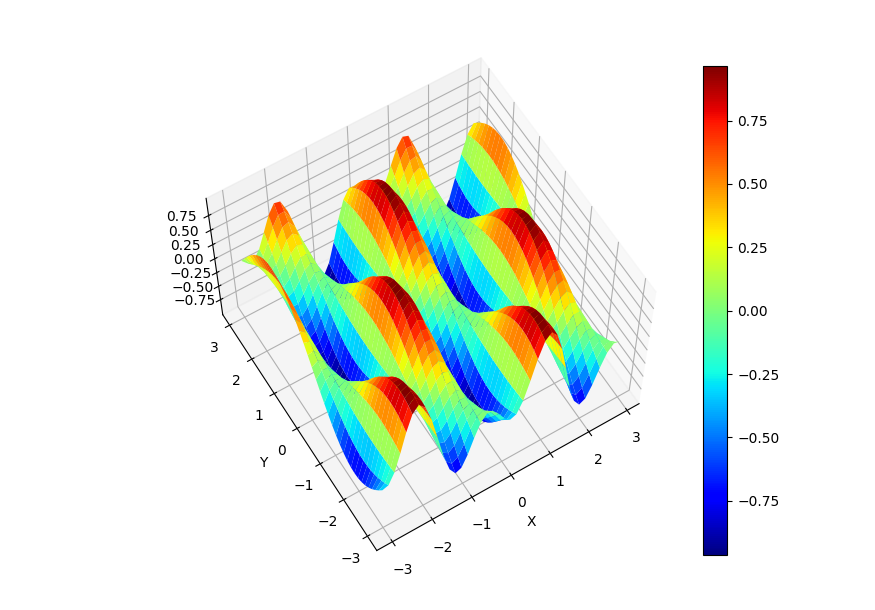
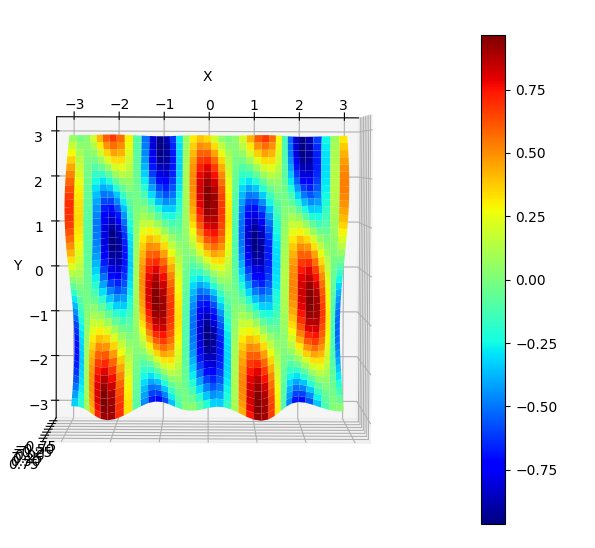
 

Рисунок 28 – График функции

На следующих рисунках приведены графики некоторых нейросетей. Некоторые графики (Рисунок 29, Рисунок 30, Рисунок 31) являются близкими к правильному, некоторые (Рисунок 32, Рисунок 33) – нет.

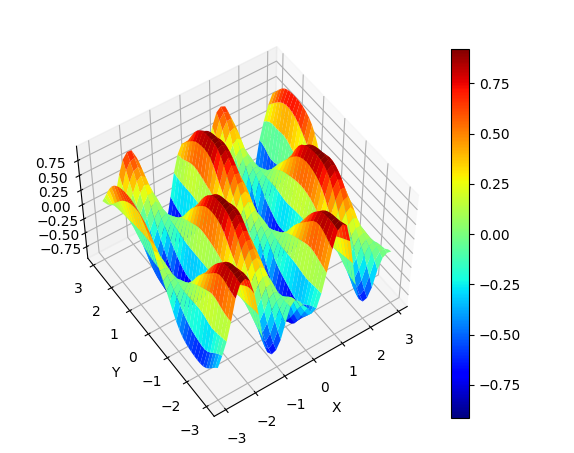
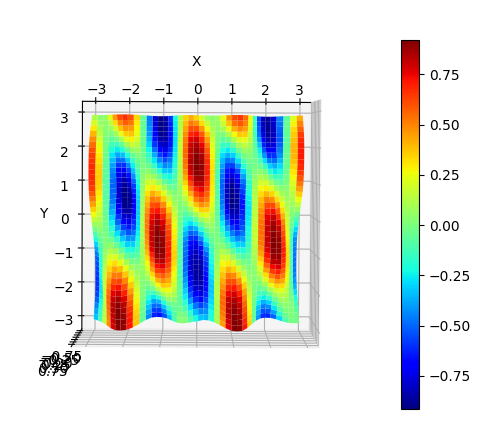
 

Рисунок 29 – График сети n40\_40\_40\_40r160

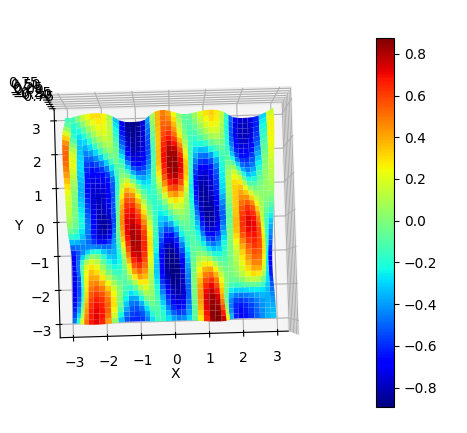
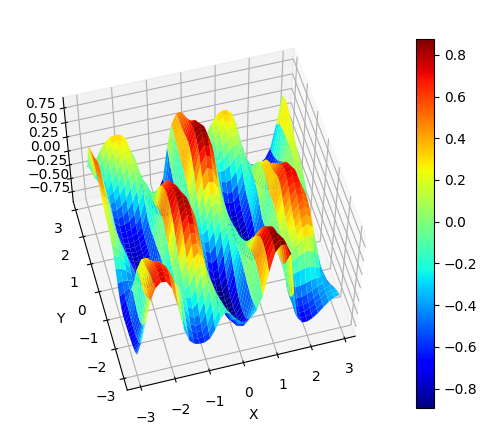


Рисунок 30 – График сети n20\_20r80

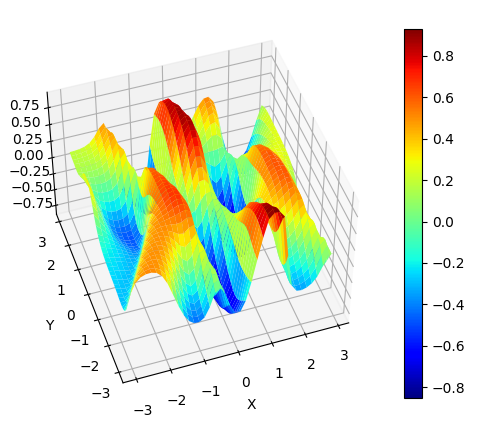
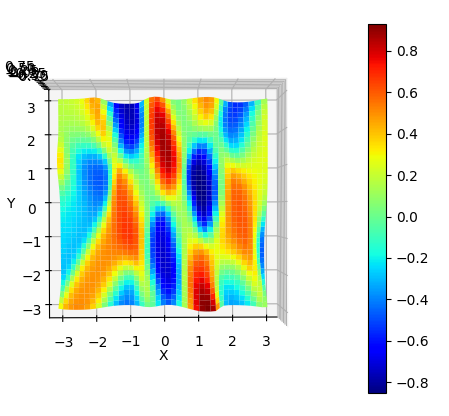
 

Рисунок 31 – График сети n10\_10r80

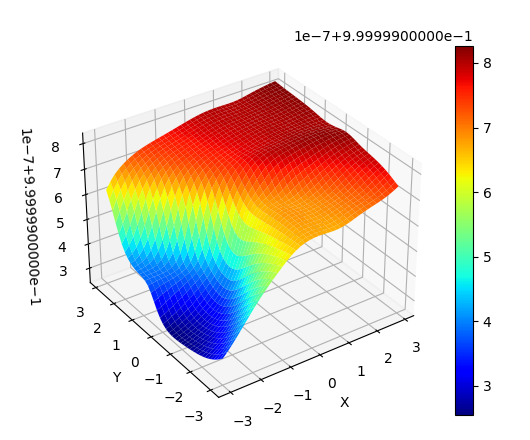


Рисунок 32 – График сети n40\_40\_40\_40c160

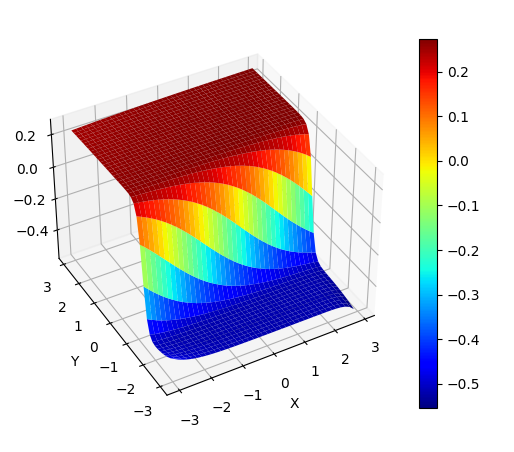


Рисунок 33 – График сети n10c160

Графики были построены с помощью скрипта на языке Python 3 (Приложение 4). Данные для построения графиков были сгенерированы основной программой.

# Вывод

В ходе работы была разработана программа, позволяющая работать с нейросетями и наборами данных определенного вида. Данная программа имеет интерфейс командной строки, что позволяет использовать текстовые команды для управления. Программа была написана на языке C# на платформе .NET, что позволяет запускать ее на различных операционных системах.

С помощью данной программы были созданы и обучены различные нейросети для исследования определенных зависимостей. В результате анализа полученных данных было обнаружено, что:

* с увеличением числа нейронов и слоев, уменьшается шанс получения удачной сети
* при обучении на сеточном наборе увеличение числа слоев ухудшает точность
* при обучении на случайном наборе точность, близкая к лучшей, достигается при двух слоях
* при обучении на сеточном наборе увеличение числа нейронов на слое не улучшает точность
* при обучении на случайном наборе увеличение числа нейронов на слое обычно улучшает точность
* при обучении на сеточном наборе существует конечный размер набора, при котором достигается лучшая точность
* при обучении на случайном наборе увеличение размера набора улучшает точность
* при малых размерах набора сеточное и случайное распределения приводят к примерно одинаковой точности
* при больших размерах набора случайное распределение приводит к значительно лучшей точности, чем сеточное
* обычно точность, близкая к наилучшей, достигается на первых нескольких эпохах; существуют исключения из данного правила

Были построены графики некоторых сетей в качестве демонстрации.

# Приложение 1. Код класса, реализующего нейросеть

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using MathNet.Numerics.LinearAlgebra;

namespace MatrixNeuralNetwork {

public class MatrixNN {

public string Structure {

get {

StringBuilder str = new StringBuilder("{");

foreach (int layerc in NetStr) {

str.AppendFormat("{0},", layerc);

}

str.Remove(str.Length - 1, 1);

str.Append("}");

return str.ToString();

}

}

public int LayerAmount { get; private set; }

public int[] NetStr { get; private set; }

private Vector<double>[] dNeurons { get; set; }

public Matrix<double>[] Weights { get; set; }

private Matrix<double>[] dWeights { get; set; }

public Vector<double>[] Shift { get; set; }

private Vector<double>[] dShift { get; set; }

public double[] errs {get; private set; }

public int CurrentEra { get => era; }

private int era;

public int CurrentElement { get => el; }

private int el;

public int CountEra { get; private set; }

public int CountElement { get; private set; }

public MatrixNN(int[] NNstructure) {

NetStr = NNstructure;

LayerAmount = NNstructure.Length;

dNeurons = new Vector<double>[LayerAmount];

Shift = new Vector<double>[LayerAmount];

dShift = new Vector<double>[LayerAmount];

Weights = new Matrix<double>[LayerAmount - 1];

dWeights = new Matrix<double>[LayerAmount - 1];

for (int i = 0; i < LayerAmount - 1; i++) {

dNeurons[i] = Vector<double>.Build.Dense(NNstructure[i], 0);

Weights[i] = Matrix<double>.Build.Random(NNstructure[i + 1], NNstructure[i]);

dWeights[i] = Matrix<double>.Build.Dense(NNstructure[i + 1], NNstructure[i], 0);

Shift[i] = Vector<double>.Build.Random(NNstructure[i + 1], NNstructure[i]);

dShift[i] = Vector<double>.Build.Dense(NNstructure[i + 1], 0);

}

}

public MatrixNN(int[] NNstructure, double? value) {

LayerAmount = NNstructure.Length;

dNeurons = new Vector<double>[LayerAmount];

Shift = new Vector<double>[LayerAmount];

dShift = new Vector<double>[LayerAmount];

Weights = new Matrix<double>[LayerAmount - 1];

dWeights = new Matrix<double>[LayerAmount - 1];

for (int i = 0; i < LayerAmount - 1; i++) {

dNeurons[i] = Vector<double>.Build.Dense(NNstructure[i], 0);

dWeights[i] = Matrix<double>.Build.Dense(NNstructure[i + 1], NNstructure[i], 0);

dShift[i] = Vector<double>.Build.Dense(NNstructure[i + 1], 0);

}

if (value is not null) {

for (int i = 0; i < LayerAmount - 1; i++) {

Weights[i] = Matrix<double>.Build.Dense(NNstructure[i + 1], NNstructure[i], (double)value);

Shift[i] = Vector<double>.Build.Dense(NNstructure[i + 1], (double)value);

}

}

}

private double ActivationFunction(double value) {

return 1 / (1 + Math.Exp(-value));

}

private double ActFuncDiff(double power) {

return power \* (1 - power);

}

private Vector<double>[] ForwardPass(double[] inputs) {

Vector<double>[] Neurons = new Vector<double>[LayerAmount];

Neurons[0] = Vector<double>.Build.DenseOfArray(inputs);

for (int i = 1; i < LayerAmount; i++) {

Neurons[i] = (Weights[i - 1] \* Neurons[i - 1] + Shift[i - 1]).Map(ActivationFunction);

}

return Neurons;

}

public double[] EvalValue(double[] inputs) {

return ForwardPass(inputs)[LayerAmount - 1].AsArray();

}

private ValueTriple<double, Matrix<double>[], Vector<double>[]> TrainCaseBackpropagation(double[] input, double[] idealOutput, double eduSpeed) {

Vector<double>[] Neurons = ForwardPass(input);

double[] output = Neurons[LayerAmount - 1].AsArray();

// find error

double error = 0;

for (int i = 0; i < output.Length; i++) {

error += (output[i] - idealOutput[i]) \* (output[i] - idealOutput[i]);

}

// go backward

// count deltas

Vector<double>[] dN = new Vector<double>[LayerAmount];

dN[LayerAmount - 1] = (Neurons[LayerAmount - 1] - Vector<double>.Build.DenseOfArray(idealOutput)).PointwiseMultiply(Neurons[LayerAmount - 1]).PointwiseMultiply(1 - Neurons[LayerAmount - 1]);

for (int i = LayerAmount - 2; i > 0; i--) {

dN[i] = Neurons[i].PointwiseMultiply(1 - Neurons[i]).PointwiseMultiply(Weights[i].Transpose() \* dN[i + 1]);

}

// count weight changes

Matrix<double>[] dW = new Matrix<double>[LayerAmount - 1];

Vector<double>[] dS = new Vector<double>[LayerAmount - 1];

for (int i = 0; i < LayerAmount - 1; i++) {

dW[i] = -eduSpeed \* dN[i + 1].OuterProduct(Neurons[i]);

dS[i] = -eduSpeed \* dN[i + 1] \* 1;

// change weights in outer function for batched training

}

return new ValueTriple<double, Matrix<double>[], Vector<double>[]>(error, dW, dS);

}

public double[] TrainNet(CasesSet trainSet, int eraAmount = 10000, int batchSize = 1, double eduSpeed = 0.3) {

CountElement = trainSet.Count;

CountEra = eraAmount;

double[] meanError = new double[eraAmount];

int counter = 0;

var batches = trainSet.GroupBy(x => (int)(counter++ / batchSize));

for (era = 0; era < eraAmount; era++) {

meanError[era] = 0;

el = 0;

foreach (var batch in batches) {

foreach (var item in batch) {

ValueTriple<double, Matrix<double>[], Vector<double>[]> v = TrainCaseBackpropagation(item.Key, item.Value, eduSpeed);

meanError[era] += v.Value1;

for (int i = 0; i < LayerAmount - 1; i++) {

dWeights[i] += v.Value2[i];

dShift[i] += v.Value3[i];

}

el++;

}

for (int i = 0; i < LayerAmount - 1; i++) {

Weights[i] = Weights[i] + dWeights[i];

Shift[i] = Shift[i] + dShift[i];

dWeights[i].Clear();

dShift[i].Clear();

}

}

meanError[era] /= CountElement;

}

this.errs = meanError;

return meanError;

}

public double TestNet(CasesSet testSet) {

double error = 0;

double errorMean = 0;

double[] output;

foreach (var item in testSet) {

output = ForwardPass(item.Key)[LayerAmount - 1].ToArray();

error = 0;

for (int i = 0; i < output.Length; i++) {

error += (output[i] - item.Value[i]) \* (output[i] - item.Value[i]);

}

errorMean += error;

}

errorMean /= testSet.Count;

return errorMean;

}

public double TestNet(double[] input, double[] output) {

double error = 0;

double[] out1 = ForwardPass(input)[LayerAmount - 1].AsArray();

for (int i = 0; i < output.Length; i++) {

error += (out1[i] - output[i]) \* (out1[i] - output[i]);

}

return error;

}

public CasesSet GraphNet(CasesSet input){

CasesSet result = new CasesSet();

double[] output;

foreach (var item in input) {

output = ForwardPass(item.Key)[LayerAmount - 1].ToArray();

result.Add(item.Key, output);

}

return result;

}

public CasesSet GraphNet(List<double[]> input){

CasesSet result = new CasesSet();

double[] output;

foreach (var item in input) {

output = ForwardPass(item)[LayerAmount - 1].ToArray();

result.Add(item, output);

}

return result;

}

struct ValueTriple<T, G, H> {

public T Value1 { get; set; }

public G Value2 { get; set; }

public H Value3 { get; set; }

public ValueTriple(T v1, G v2, H v3) {

Value1 = v1;

Value2 = v2;

Value3 = v3;

}

}

public MatrixNN Copy() {

MatrixNN res = new MatrixNN(NetStr);

for (int i = 0; i < LayerAmount - 1; i++) {

res.Weights[i] = Weights[i];

res.Shift[i] = Shift[i];

}

return res;

}

}

}

# Приложение 2. Список команд пользователя

| **Синтаксис команды** | **Описание параметров** | **Описание команды** |
| --- | --- | --- |
| exit |  | Завершение работы программы |
| exec *file* | *file* – путь к файлу | Выполнение команд пользователя построчно из файла |
| net new *name str* | *name* – имя сети  *str* – структура сети в виде {n1,n2,…,nk} без пробельных символов | Создание нейросети с заданным именем и структурой.  В структуре указывается количество нейронов на каждом слое |
| net ls |  | Вывод всех хранящихся в памяти сетей в виде  *имя структура* |
| net rm *name* | *name* – имя сети | Удаление сети с заданным именем |
| net import *name file* | *name* – имя сети  *file* – путь к файлу | Импорт сети из файла и присвоение этой сети имени |
| net export *name file* | *name* – имя сети  *file* – путь к файлу | Экспорт сети в файл |
| net copy *src new* | *src* – имя копируемой сети  *new* – имя новой сети | Создание новой сети как копии существующей сети |
| case new uniform *name nx x1 x2 ny y1 y2* | *name* – имя набора  *nx, x1, x2* – количество точек на отрезке [x1, x2]  *ny, y1, y2* – количество точек на отрезке [y1, y2] | Создание равномерного сеточного распределения точек на прямоугольной области  По оси *Ox* область разбивается *nx* линиями, включая граничные  По оси *Oy* область разбивается *ny* линиями, включая граничные |
| case new random name n x1 x2 y1 y2 | *name* – имя набора  *n* – количество точек  *x1, x2, y1, y2* – границы области по *x* и *y* | Создание равномерного случайного распределения на прямоугольной области  В области выбираются ­*n* точек |
| case ls |  | Вывод всех хранящихся в памяти наборов в виде  *имя количество точек* |
| case rm *name* | *name* – имя набора | Удаление набора с заданным именем |
| case import *name file* | *name* – имя набора  *file* – путь к файлу | Импорт набора из файла и присвоение этому набору имени |
| case export *name file* | *name* – имя набора  *file* – путь к файлу | Экспорт сети в файл |
| train *net set [new] [epoch batch edu]* | *net* – имя сети  *s*et – имя набора  *new* – имя новой сети  *epoch* – количество эпох (по умолчанию 100)  *batch* – размер пакета (по умолчанию 100)  *edu* – скорость обучения (по умолчанию 0,3) | Обучение сети на наборе данных.  Можно указать необходимое количество эпох, размер пакета и скорость обучения.  Можно указать параметр *new*. При этом будет создана копия сети *net*, названная именем *new*, которая будет затем обучена. |
| eval *net data* | *net* – имя сети  *data* – входные значения в виде {i1, i2, …, in} | Применение нейросети к заданному входному вектору. |
| testcase *net case* | *net* – имя сети  *case* – тестовый случай в виде  {i1, i2, …, in} => {o1, o2, …, om} | Тестирование нейросети на заданных входных данных {i1, i2, …, in}, для которых известны правильные выходные данные {o1, o2, …, om} |
| test *net set* | *net* – имя сети  *s*et – имя набора | Тестирование нейросети на наборе данных |
| graphnet *net set file* | *net* – имя сети  *s*et – имя набора  *file* – путь к файлу | Вычисление значения нейросети для каждого элемента набора. Пары (входные данные, значение нейросети) записываются в указанный файл.  Из набора данных *set* используются только входные данные |

# Приложение 3. Ошибки сетей для всех попыток

| **Сеть \ Попытка** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **min** | **max** | **norm** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **n5c80** | 0,097647 | 0,100237 | 0,101017 | 0,097720 | 0,097668 | 0,100715 | 0,099553 | 0,113256 | 0,061428 | 0,061461 | 0,061428 | 0,113256 | 10 |
| **n5\_5c80** | 0,068216 | 0,048386 | 0,077782 | 0,051496 | 0,056548 | 0,090995 | 0,060250 | 0,080068 | 0,146931 | 0,060242 | 0,048386 | 0,146931 | 10 |
| **n5\_5\_5c80** | 0,061026 | 0,070422 | 0,066310 | 0,051897 | 0,058445 | 0,061252 | 0,162622 | 0,057371 | 0,051950 | 0,055246 | 0,051897 | 0,162622 | 10 |
| **n5\_5\_5\_5c80** | 0,049943 | 0,061106 | 0,070772 | 0,060356 | 0,060666 | 0,056243 | 0,058470 | 0,069321 | 0,059133 | 0,074653 | 0,049943 | 0,074653 | 10 |
| **n10c80** | 0,110592 | 0,096356 | 0,099679 | 0,138472 | 0,194373 | 0,096526 | 0,198395 | 0,061049 | 0,061528 | 0,061211 | 0,061049 | 0,198395 | 10 |
| **n10\_10c80** | 0,051332 | 0,049208 | 0,066125 | 0,049301 | 0,061459 | 0,084945 | 0,037574 | 0,075403 | 0,065659 | 0,047454 | 0,037574 | 0,084945 | 10 |
| **n10\_10\_10c80** | 0,039561 | 0,060271 | 0,082914 | 0,053633 | 0,088385 | 0,088366 | 0,044996 | 0,042568 | 0,053700 | 0,054731 | 0,039561 | 0,088385 | 10 |
| **n10\_10\_10\_10c80** | 0,052622 | 0,073754 | 0,053275 | 0,048493 | 0,050557 | 0,052750 | 0,047673 | 0,054867 | 0,054611 | 0,064835 | 0,047673 | 0,073754 | 10 |
| **n20c80** | 0,091725 | 0,182654 | 0,192096 | 0,095122 | 0,117739 | 0,138741 | 0,116857 | 0,179590 | 0,132345 | 0,095515 | 0,091725 | 0,192096 | 10 |
| **n20\_20c80** | 0,072831 | 0,057690 | 0,075725 | 0,309977 | 0,309982 | 0,059589 | 0,056640 | 0,057724 | 0,055692 | 0,309982 | 0,055692 | 0,309982 | 7 |
| **n20\_20\_20c80** | 0,309982 | 0,056958 | 0,309981 | 0,058458 | 0,057510 | 0,309981 | 0,309981 | 0,076114 | 0,309982 | 0,059906 | 0,056958 | 0,309982 | 5 |
| **n20\_20\_20\_20c80** | 0,058802 | 0,309982 | 0,061168 | 0,056768 | 0,309982 | 0,055922 | 0,070799 | 0,309981 | 0,046351 | 0,039770 | 0,03977 | 0,309982 | 7 |
| **n30c80** | 0,059900 | 0,123005 | 0,143429 | 0,099439 | 0,221635 | 0,123147 | 0,123927 | 0,129994 | 0,059593 | 0,105425 | 0,059593 | 0,221635 | 10 |
| **n30\_30c80** | 0,068878 | 0,309982 | 0,111689 | 0,309982 | 0,309981 | 0,309980 | 0,309982 | 0,067304 | 0,076497 | 0,036015 | 0,036015 | 0,309982 | 5 |
| **n30\_30\_30c80** | 0,309982 | 0,309982 | 0,309982 | 0,070208 | 0,309981 | 0,054178 | 0,309982 | 0,309982 | 0,309982 | 0,309976 | 0,054178 | 0,309982 | 2 |
| **n30\_30\_30\_30c80** | 0,309981 | 0,069886 | 0,309982 | 0,309981 | 0,309982 | 0,309981 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,069886 | 0,309982 | 1 |
| **n40c80** | 0,122530 | 0,309970 | 0,207186 | 0,113788 | 0,099502 | 0,309979 | 0,120448 | 0,309980 | 0,115335 | 0,072756 | 0,072756 | 0,30998 | 7 |
| **n40\_40c80** | 0,309979 | 0,309982 | 0,309982 | 0,309981 | 0,309982 | 0,309976 | 0,078824 | 0,309981 | 0,309982 | 0,195851 | 0,078824 | 0,309982 | 2 |
| **n40\_40\_40c80** | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309980 | 0,309982 | 0,309978 | 0,309978 | 0,309982 | 0 |
| **n40\_40\_40\_40c80** | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0,309982 | 0 |
| **n5r80** | 0,073566 | 0,073300 | 0,073879 | 0,073291 | 0,074007 | 0,031330 | 0,073672 | 0,074204 | 0,073832 | 0,064625 | 0,03133 | 0,074204 | 10 |
| **n5\_5r80** | 0,056166 | 0,033121 | 0,044395 | 0,052412 | 0,056554 | 0,048043 | 0,040607 | 0,049541 | 0,048745 | 0,047901 | 0,033121 | 0,056554 | 10 |
| **n5\_5\_5r80** | 0,039997 | 0,051792 | 0,039660 | 0,062644 | 0,059741 | 0,045009 | 0,050166 | 0,044189 | 0,048596 | 0,059745 | 0,03966 | 0,062644 | 10 |
| **n5\_5\_5\_5r80** | 0,051598 | 0,073679 | 0,057826 | 0,074024 | 0,052320 | 0,073830 | 0,058727 | 0,051195 | 0,073197 | 0,064010 | 0,051195 | 0,074024 | 10 |
| **n10r80** | 0,054210 | 0,054501 | 0,066506 | 0,034847 | 0,073149 | 0,030248 | 0,073464 | 0,059569 | 0,052220 | 0,053057 | 0,030248 | 0,073464 | 10 |
| **n10\_10r80** | 0,013441 | 0,040653 | 0,028665 | 0,040843 | 0,040241 | 0,039120 | 0,042518 | 0,016066 | 0,032291 | 0,011659 | 0,011659 | 0,042518 | 10 |
| **n10\_10\_10r80** | 0,043670 | 0,046557 | 0,018794 | 0,039223 | 0,028997 | 0,043959 | 0,029466 | 0,025845 | 0,050703 | 0,039210 | 0,018794 | 0,050703 | 10 |
| **n10\_10\_10\_10r80** | 0,023792 | 0,045814 | 0,061479 | 0,045758 | 0,040306 | 0,040009 | 0,046638 | 0,066056 | 0,037781 | 0,035134 | 0,023792 | 0,066056 | 10 |
| **n20r80** | 0,030140 | 0,031165 | 0,052939 | 0,031012 | 0,030996 | 0,031550 | 0,049174 | 0,031034 | 0,054456 | 0,073533 | 0,03014 | 0,073533 | 10 |
| **n20\_20r80** | 0,014326 | 0,012510 | 0,016394 | 0,029311 | 0,014250 | 0,006954 | 0,039141 | 0,028744 | 0,013246 | 0,028288 | 0,006954 | 0,039141 | 10 |
| **n20\_20\_20r80** | 0,013910 | 0,020042 | 0,009181 | 0,015252 | 0,013286 | 0,010233 | 0,309981 | 0,014617 | 0,004616 | 0,009769 | 0,004616 | 0,309981 | 9 |
| **n20\_20\_20\_20r80** | 0,020178 | 0,019841 | 0,011070 | 0,008399 | 0,017432 | 0,036988 | 0,018585 | 0,045383 | 0,015064 | 0,007507 | 0,007507 | 0,045383 | 10 |
| **n30r80** | 0,032515 | 0,065734 | 0,033070 | 0,036928 | 0,051811 | 0,054045 | 0,031088 | 0,031689 | 0,030966 | 0,054071 | 0,030966 | 0,065734 | 10 |
| **n30\_30r80** | 0,009710 | 0,009053 | 0,032086 | 0,309982 | 0,005415 | 0,007477 | 0,021989 | 0,009661 | 0,013402 | 0,006954 | 0,005415 | 0,309982 | 9 |
| **n30\_30\_30r80** | 0,008081 | 0,005543 | 0,042864 | 0,006597 | 0,309981 | 0,309982 | 0,309981 | 0,009903 | 0,309982 | 0,309981 | 0,005543 | 0,309982 | 5 |
| **n30\_30\_30\_30r80** | 0,013671 | 0,309980 | 0,309980 | 0,309982 | 0,309982 | 0,309900 | 0,309982 | 0,309982 | 0,002735 | 0,309981 | 0,002735 | 0,309982 | 2 |
| **n40r80** | 0,031233 | 0,052750 | 0,032866 | 0,034557 | 0,048930 | 0,051784 | 0,053891 | 0,079500 | 0,066770 | 0,031462 | 0,031233 | 0,0795 | 10 |
| **n40\_40r80** | 0,009087 | 0,309981 | 0,009996 | 0,034047 | 0,011314 | 0,309980 | 0,309982 | 0,309980 | 0,309982 | 0,034975 | 0,009087 | 0,309982 | 5 |
| **n40\_40\_40r80** | 0,309982 | 0,004169 | 0,309981 | 0,309982 | 0,309981 | 0,002992 | 0,017808 | 0,309982 | 0,309982 | 0,007348 | 0,002992 | 0,309982 | 4 |
| **n40\_40\_40\_40r80** | 0,001040 | 0,309981 | 0,309982 | 0,309981 | 0,309982 | 0,309981 | 0,309982 | 0,309982 | 0,006517 | 0,309982 | 0,00104 | 0,309982 | 2 |

# Приложение . Код скрипта для построения графиков

**import** math  
**import** scipy  
**from** scipy **import** linalg  
**import** numpy **as** np  
**import** re  
  
**from** pylab **import** meshgrid  
**from** matplotlib **import** cm  
**import** matplotlib.pyplot **as** plt  
  
**def** NetToReal(value):  
 teta = (value-0)/1  
 **return** -1+teta\*2  
  
filename = **"graphs\\n10\_10r80"**n = 161  
  
f = open(filename, **"r"**)  
line = f.readline()  
num = line[line.find(**" "**)+1:]  
num = int(num)  
  
X = list()  
Y = list()  
Z = list()  
  
**for** i **in** range(num):  
 line = f.readline()  
 match = re.search(**r'{((?:[\d\,\.\-]+\s?)+)} => {((?:[\d\,\.\-]+\s?)+)}'**, line)  
 X.append(float(match.group(1).split(**' '**)[0]))  
 Y.append(float(match.group(1).split(**' '**)[1]))  
 Z.append(float(match.group(2).split(**' '**)[0]))  
  
X = np.array(X).reshape((n,n))  
Y = np.array(Y).reshape((n,n))  
Z = np.array(Z).reshape((n,n))  
Z = NetToReal(Z)  
  
X = X[1:161:4, 1:161:4]  
Y = Y[1:161:4, 1:161:4]  
Z = Z[1:161:4, 1:161:4]  
  
fig1 = plt.figure(1)  
ax1 = fig1.add\_subplot(111, projection=**'3d'**)  
  
color = **lambda** Z: cm.jet((Z-np.amin(Z))/(np.amax(Z)-np.amin(Z)))  
  
p1 = ax1.plot\_surface(X, Y, Z,  
 rstride = 1,  
 cstride = 1,  
 cmap=cm.jet)  
fig1.colorbar(p1)  
  
plt.xlabel(**'X'**)  
plt.ylabel(**'Y'**)  
  
plt.show()