МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского (ННГУ)»

Институт информационных технологий, математики и механики

Кафедра: Алгебры, геометрии и дискретной математики (АГДМ)

Направление подготовки: **«Прикладная математика и информатика»**

Профиль подготовки: **«Прикладная математика и информатика (общий профиль)»**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему:

«Поиск путей в лабиринте с помощью полносвязных нейронных сетей»

Выполнила:
студентка группы
3821Б1ПМоп1 Храмова А.А.
Проверил:
доцент каф. АГДМ, д-р
физ.-мат. наук Золотых Н.Ю.

Нижний Новгород
2025

Содержание

1. Введение

В современном мире методы машинного обучения, использующие искусственные нейронные сети, стали крайне востребованными. Их ключевая способность предсказывать результаты на основе больших данных имеет решающее значение для развития многих научных и технических направлений.
На основе первой теоремы Розенблатта[2] было показано, как элементарный перцептрон может решить эту задачу. Розенблатт не накладывал ограничений на связи между нейронами - каждый А-элемент может "видеть"всё изображение целиком. Хотя такой перцептрон универсален, было выяснено, что этот алгоритм использует экспоненциально большое количество нейронов на скрытом слое, тогда как для глубокой сети достаточно полиномиальной сложности второго порядка, поэтому он наиболее эффективен [1].
Ученые предлагали различные определения ограничений на выбор ассоциативных элементов[3]. Если каждый А-элемент имеет ограниченное рецептивное поле, то есть обрабатывается только некоторая локальная область изображения, то перцептрон не сможет решать определенные задачи: понять, все ли черные пиксели относятся к одной области; определить, сколько пикселей закрашено. Эти задачи требуют полного анализа всего изображения, что невозможно при ограничении на "локальность"А-элементов. Общим выводом из всех предположений служит утверждение: если человек не может решить задачу сразу с первого взгляда и ему нужно применить некоторые последовательные операции, такие как подсчет пикселей или следование запутанной траектории, то эта задача не может быть решена ограниченным элементарным перцептроном, так как ограниченные перцептроны принципиально не могут решать задачи, требующие последовательного анализа. Поэтому следует ожидать, что неограниченный элементарный перцептрон может решить эту задачу, но ценой большой сложности, а глубокие сети с последовательной природой, такие как лабиринт, дают экспоненциальный выигрыш в эффективности [1].
Предметом данной работы является реализация двух нейросетевых архитектур для версии задачи о перемещении по лабиринту: для полносвязной глубокой сети и для неглубокой сети. Исследование посвящено проведению практических экспериментов при различных параметрах задачи с целью подтверждения теоретических результатов оригинальной статьи, а также сравнительного анализа производительности и эффективности каждого из подходов.

2. Постановка задачи

Имеется n людей и n объектов с однозначным соответствием, где каждый объект принадлежит ровно одному человеку (см. рисунок 1). Между ними задано L промежуточных этапов, каждый из которых описывается перестановкой. Составляется матрица перестановок, которая описывает диаграмму из n ломаных линий с L вершинами, характеризующих связи между людьми и объектами. $X_i, i = 1, 2, .., L$ - это матрица перестановок на i-том шаге, $X_1 \cdot X_2 \cdot .. \cdot X_L$ - тоже матрица перестановок, которая уже описывает связь «человек-объект». В этой работе под лабиринтом понимается его упрощенная модель без петель, шагов назад или пересечений путей [1].
Требуется построить неглубокую и глубокую нейронные сети, определяющие соответствие «человек-объект» по диаграмме.

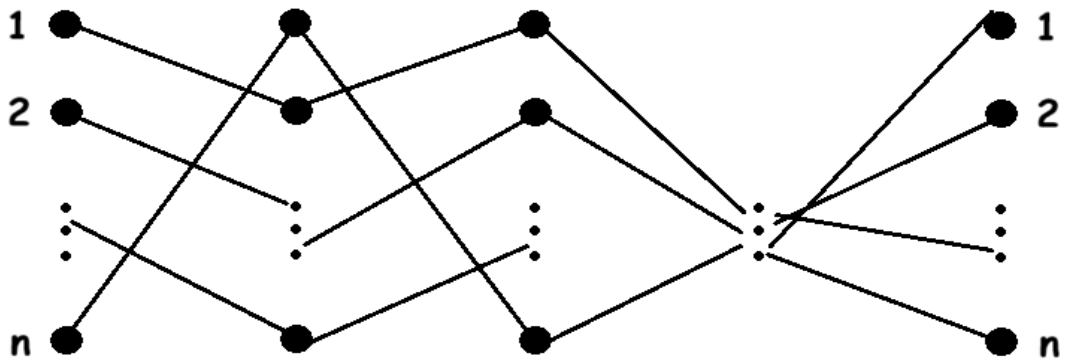

Рис. 1: Диаграмма соответствия между n людьми и n объектами

Каждая задача лабиринта путешествий представляет собой L-кортеж $(f_1, f_2, .., f_L)$ биективных функций (перестановок) из $\{1, ..., n\}$ в $\{1, ..., n\}$. $F(n, L)$ содержит все возможные L-кортежи перестановок. Композиция функций из L-кортежа также является биективной функцией из $\{1, ..., n\}$ в $\{1, ..., n\}$, что как раз биективно связывает объекты с людьми. Композиция всех перестановок f_i в кортеже будет определять итоговую перестановку из множества входных вершин в множество выходных. Выход каждого слоя вычисляется как булева функция от значений предыдущего слоя, поэтому каждый нейрон в сети, полученный в результате этой функции, имеет булевое значение [1].

3. Цель исследования

1. Программно реализовать две различные нейросетевые архитектуры для решения задачи поиска путей в лабиринте, которая сводится к задаче композиции перестановок
2. Проверить на практике их эффективность
3. Сравнить их производительность при различных значениях параметров размера перестановки n и количества перемножаемых матриц L
4. Подтвердить теоретические выкладки статьи экспериментальными данными

4. Решение модельной задачи. Архитектура и принцип работы неглубокой сети.

4.1. Генерация данных

Для данной задачи сгенерируем датасет, состоящий из картинок лабиринтов - то есть диаграмм соответствий между n входами и n выходами. Диаграмма состоит из n ломанных линий с L рёбрами, характеризующими связи между входными и выходными объектами.
Каждое изображение в наборе представляет собой график, созданный при использовании библиотеки "matplotlib". Для визуализации диаграммы нам не нужны оси, поэтому отключаем их построение. Линии создаются путем соединения точек, координаты которых генерируются с помощью функции "random.sample"из модуля "random". Она перемешивает значения в списке из уникальных чисел, диапазон которых определяется параметрами n и L задачи (см. рисунки 2 и 3).
Сгенерированные данные представляются в черно-белом цвете - черные линии на белом фоне. Для улучшения эффективности распознавания пикселей применяется следующая визуальная оптимизация: увеличение толщины линий и размера вершинных маркеров, сокращение площади незанятого белого пространства, а также максимальное заполнение пространства чёрными линиями. Однако принято решение инвертировать цвета, так как это позволяет лучше распознавать полезные признаки и соответствует ожиданиям сверточных нейросетей, потому что для них 0=фон. Также мы используем режим открытия картинки в оттенках серого (convert("L")) для того, чтобы не терять тонкие черные линии. Все предпринятые действия улучшают контрастность и снижают уровень шумов при компьютерном анализе изображений.

Все данные преобразуются в единый формат для того, чтобы эффективно использовать память и обрабатывать данные пачками параллельно.
В названии картинки содержится итоговая перестановка, то есть реальный ответ в виде "$N^{\underline{o}}_{of\ image}[2.3.1.]$". Это в итоге преобразуется в тензор $[1., 2., 0.]$, который необходим для работы нейросети.

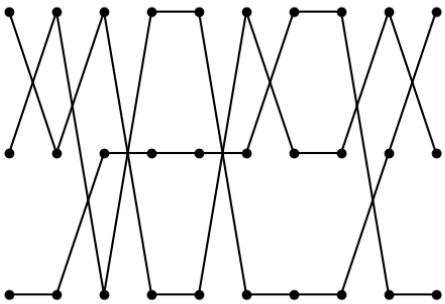

Рис. 2: Пример лабиринта с $n = 3$, $L = 9$

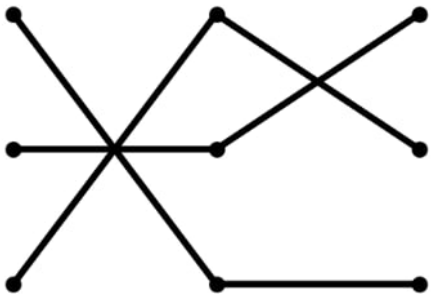

Рис. 3: Пример лабиринта с $n = 3$, $L = 2$

4.2. Практическая реализация неглубокой полносвязной сети для работы с изображениями

Архитектура состоит из двух основных блоков: два сверточных слоя и три линейных полносвязных слоя с функциями активации, добавляющими нелинейность.
В сверточной части на первом слое Conv2d выделяет 32 простых признака (например, линии, углы) с помощью специальных фильтров, каждый из которых отвечает за конкретный признак. MaxPool2d на первом слое уменьшает размерность изображения в 2 раза, сохраняя только важные особенности. На втором слое Conv2d расширяет количество признаков до 64, анализируя более сложные паттерны. После второго пулинга размерность изображения сокращается еще в 2 раза, а итоговая размерность сокращается в 4 раза. Следующие за сверточными линейные слои в полносвязном блоке превращают трехмерный вектор, полученный после сверточных слоев, в одномерный вектор и постепенно сжимают признаки, выделяя самые важные из них, пока не получим n, что соответствует размерности вектора перестановки. Затем пропускаем выход через адаптированный для задачи softmax, который преобразует его в вероятности, но преобразует их в масштабируемый числовой диапазон для перестановок. Также данные проходят через функцию активации ReLU, которая добавляет нелинейность, делая сеть более гибкой и способной к обучению.

Данная модель обучается стандартным способом с использованием методов оптимизации на основе градиента. Обучаемые параметры (веса) обновляются во время обучения, стремясь минимизировать функцию потерь. В качестве оптимизатора обучения был выбран встроенный в "pytorch"алгоритм Adam. Функция ошибки - MSELoss. В качестве оценки производительности на каждой эпохе обучения вычислялись такие метрики, как: loss, accuracy. Все метрики вычислялись на обучающей и тестовой выборках. Так как задача выполняется как задача регрессии, точность вычисляется не совсем стандартным способом. Для вычисления accuracy сортируются массив настоящих ответов и ответов предсказаний и запоминаются их индексы. Далее отсортированные индексы поочередно сравниваются, и если массивы полностью совпадают, то такой ответ засчитывается как True.

4.3. Результаты экспериментов при различных параметрах задачи

Проведём эксперименты на разных входных данных.

4.3.1. Результаты при $L = 2$

1. $n = 3$, $L = 2$

   Параметры обучения: learning rate = 10^{-5}, batch size = 32, количество эпох = 5.

   Время обучения: 12.24 сек.

Таблица 1: Результаты обучения модели по эпохам

Epoch	Train Loss	Test Loss	Train Accuracy	Test Accuracy
0	0.6750	0.6564	0.3404	0.3705
1	0.6268	0.5618	0.4300	0.4613
2	0.4119	0.2724	0.6288	0.7158
3	0.1414	0.0599	0.8667	1.0000
4	0.6319	0.6143	1.0000	1.0000

$$\left[\begin{array}{ccc} 2 & 1 & 0 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \\ 2 & 1 & 0 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{ccc} 2.0654 & 0.8464 & 0.0882 \\ 2.0863 & 0.7858 & 0.1279 \\ 0.9818 & 1.9084 & 0.1098 \\ 2.0654 & 0.8464 & 0.0882 \end{array}\right]$$

Предсказания

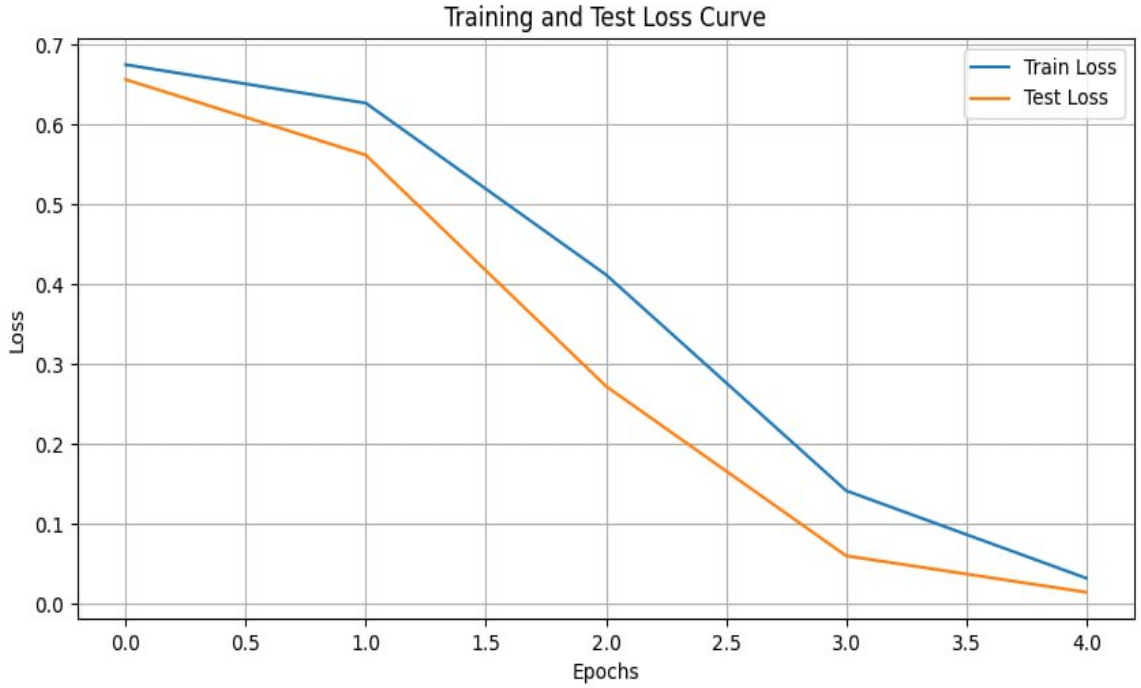

Рис. 4: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 2$

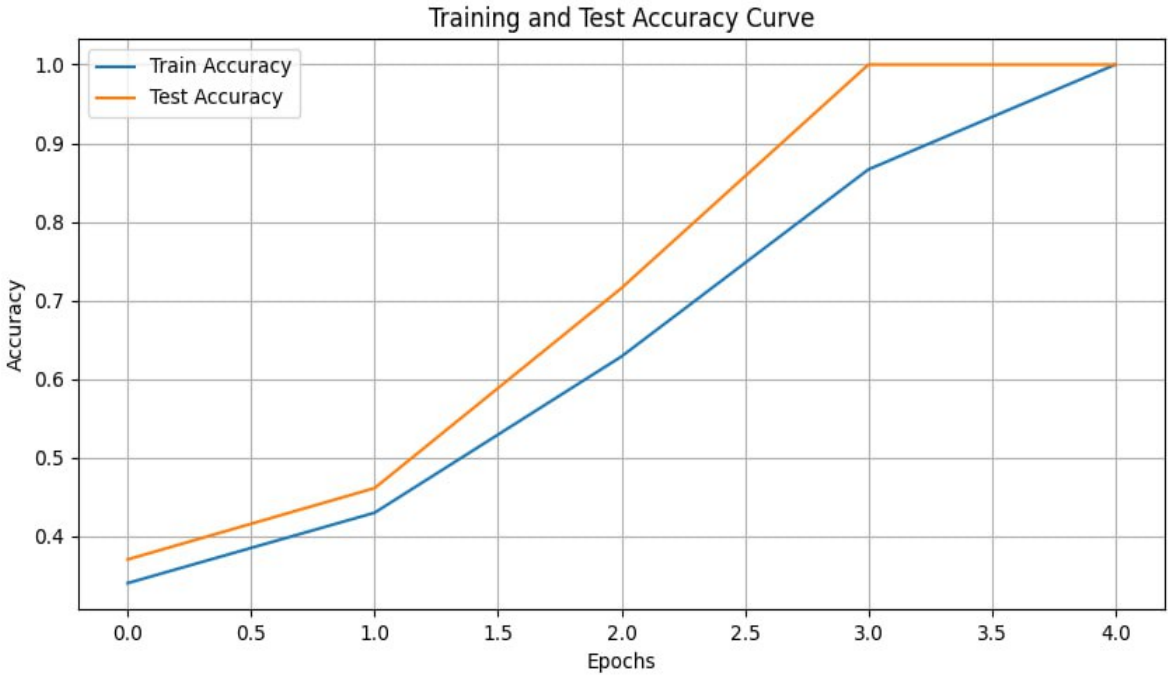

Рис. 5: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 2$

2. $n = 4, L = 2$

   Параметры обучения: learning rate = 10^{-5}, batch size = 32, количество эпох = 30

   Время выполнения: 71.01 сек.

Таблица 2: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	1.2547	1.2597	0.2462	0.2634
1	1.2439	1.2372	0.2816	0.2779
2	1.2332	1.2417	0.2897	0.2757
3	1.2142	1.2221	0.3075	0.3114
4	1.1927	1.2198	0.3428	0.2723
5	1.1664	1.1907	0.3603	0.3203
		⋮		
26	0.0785	0.1312	0.9888	0.9542
27	0.0720	0.0962	0.9931	0.9732
28	0.0642	0.0915	0.9931	0.9777
29	0.0610	0.0920	1.0000	1.0000

$$\begin{bmatrix} 2 & 0 & 1 & 3 \\ 2 & 1 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 1 & 2 & 0 & 3 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 1.7827 & 0.4774 & 0.7726 & 2.9673 \\ 1.7523 & 0.9907 & 2.8729 & 0.3840 \\ 1.8591 & 3.0181 & 0.2457 & 0.8772 \\ 0.8138 & 1.8532 & 0.2528 & 3.0802 \end{bmatrix}$$

Предсказания

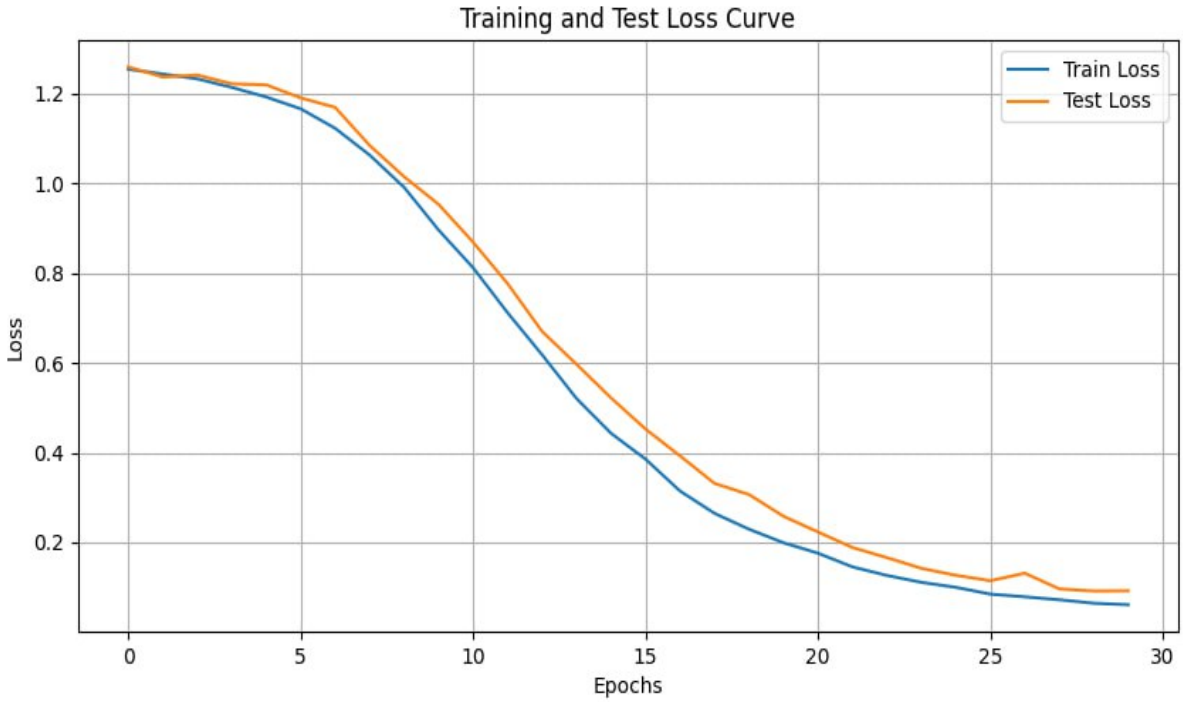

Рис. 6: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 2$

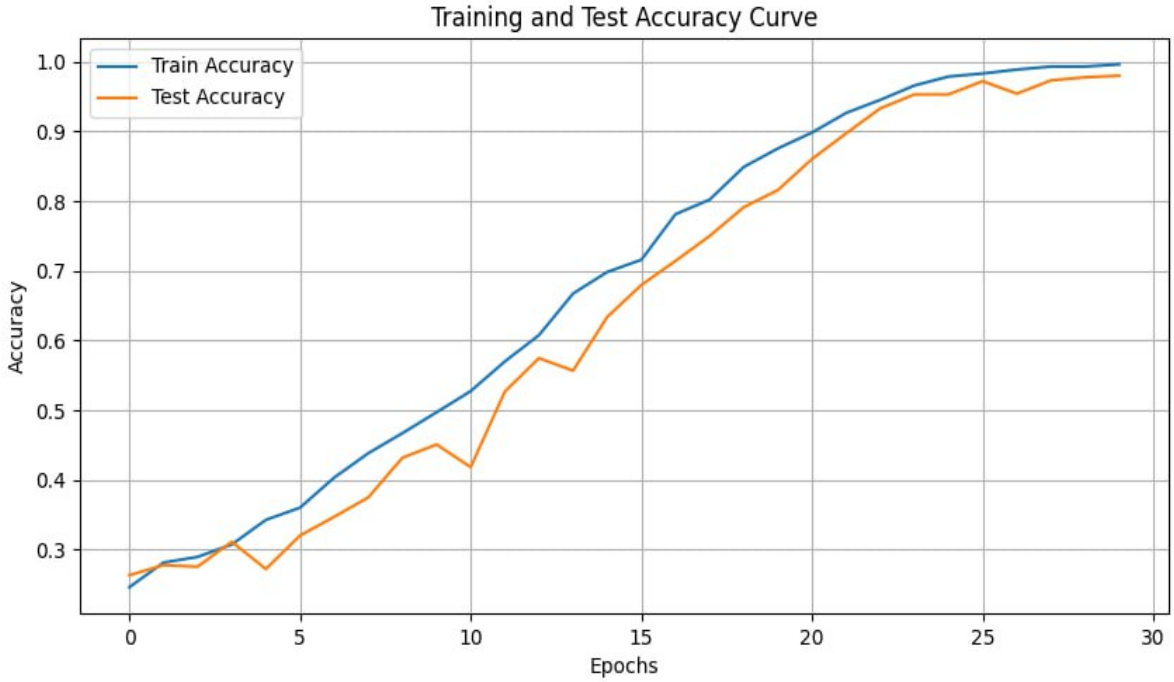

Рис. 7: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 2$

3. $n = 5, L = 2$

   Параметры обучения:

   learning rate = 0.000055, batch size = 32, количество эпох = 55

   Время обучения: 131.24 сек.

Таблица 3: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Acc.**	**Test Acc.**
0	2.0349	2.0065	0.1973	0.1848
1	1.9958	2.0073	0.2048	0.2134
2	1.9904	2.0084	0.2125	0.2250
3	1.9792	2.0128	0.2275	0.1973
4	1.9707	2.0264	0.2358	0.1688
		⋮		
50	0.0506	0.2152	0.9945	0.8705
51	0.0426	0.1943	0.9995	0.8777
52	0.0559	0.1910	0.9995	0.8777
53	0.0344	0.1908	1.0000	0.8696
54	0.0307	0.1921	1.0000	0.8732

$$\begin{bmatrix} 4 & 3 & 1 & 0 & 2 \\ 1 & 3 & 2 & 4 & 0 \\ 0 & 1 & 2 & 4 & 3 \\ 2 & 1 & 4 & 0 & 3 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 4.3714 & 2.6762 & 1.0355 & 0.1896 & 1.7272 \\ 1.3216 & 3.5878 & 1.6319 & 3.2452 & 0.2135 \\ 0.1921 & 0.9811 & 2.1498 & 4.1048 & 2.5721 \\ 3.0450 & 1.4471 & 2.3975 & 0.2902 & 2.8203 \end{bmatrix}$$

Предсказания

Последующие графики будут в приложении, так как они выглядят идентично тем, что уже представлены в предыдущих результатах. Графики: 15 и 16.

4.3.2. Результаты при $L = 3$

1. $n = 3$, $L = 3$

   Параметры обучения: learning rate = 0.000055, batch size = 32, количество эпох = 15.

   Время обучения: 36.28 сек.

Таблица 4: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	0.8747	0.6667	0.3475	0.3720
1	0.6663	0.6691	0.3433	0.2783
2	0.6655	0.6649	0.3433	0.3631
3	0.6623	0.6673	0.3604	0.3318
			⋮	
11	0.0903	0.0940	0.9400	0.9479
12	0.0609	0.0583	0.9671	0.9821
13	0.0411	0.0555	0.9850	0.9851
14	0.0323	0.0382	0.9942	0.9911

$$\left[\begin{array}{ccc} 1 & 2 & 0 \\ 1 & 2 & 0 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{ccc} 1.1894 & 1.6593 & 0.1513 \\ 1.0684 & 1.8237 & 0.1079 \\ 2.2640 & 0.6329 & 0.1031 \\ 2.0178 & 0.1562 & 0.8260 \end{array}\right]$$

Предсказания

Графики: 17 и 18.

2. $n = 4$, $L = 3$

   Параметры обучения: learning rate = 0.0005, batch size = 32, количество эпох = 17.

   Время обучения: 517.96 сек.

Таблица 5: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	1.2530	1.2515	0.2562	0.2551
1	1.2510	1.2499	0.2447	0.2447
2	1.2503	1.2502	0.2585	0.2601
3	1.2494	1.2513	0.2536	0.2594
4	1.2476	1.2509	0.2654	0.2568
5	1.2432	1.2467	0.2731	0.2650
		⋮		
12	0.2186	0.2448	0.8386	0.8086
13	0.1594	0.1699	0.8940	0.8882
14	0.1188	0.1218	0.9415	0.9426
15	0.0927	0.1033	0.9707	0.9586
16	0.0752	0.0859	0.9859	0.9724

$$\begin{bmatrix} 0 & 2 & 3 & 1 \\ 0 & 2 & 3 & 1 \\ 2 & 1 & 0 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 0.2336 & 2.4956 & 2.4356 & 0.8352 \\ 0.3397 & 1.8496 & 2.7495 & 1.0612 \\ 2.1699 & 1.2554 & 0.3343 & 2.2403 \\ 0.3339 & 0.7828 & 1.8382 & 3.0451 \end{bmatrix}$$

Предсказания

Графики: 19 и 20.

3. $n = 5, L = 3$

   Параметры обучения:

   learning rate = 0.000055, batch size = 32, количество эпох = 50

   Время обучения: 3078.82 сек.

$$\begin{bmatrix} 1 & 4 & 0 & 2 & 3 \\ 4 & 1 & 3 & 2 & 0 \\ 2 & 1 & 0 & 3 & 4 \\ 2 & 4 & 0 & 3 & 1 \end{bmatrix}$$

Настоящие ответы

Таблица 6: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Acc.**	**Test Acc.**
0	2.0029	2.0001	0.2008	0.1995
1	2.0007	2.0005	0.2027	0.2011
2	2.0003	2.0003	0.2013	0.1978
3	1.9990	2.0019	0.2081	0.2023
4	1.9988	1.9990	0.2056	0.2008
5	1.9968	1.9971	0.2077	0.2031
		⋮		
45	0.0375	0.0893	0.9973	0.9579
46	0.0347	0.0988	0.9978	0.9471
47	0.0333	0.0871	0.9986	0.9602
48	0.0333	0.0836	0.9986	0.9631
49	0.0305	0.0829	0.9988	0.9619

$$\left[\begin{array}{ccccc} 1.1239 & 3.8779 & 0.3544 & 1.8385 & 2.8053 \\ 4.1471 & 0.6920 & 3.0398 & 1.9823 & 0.1387 \\ 2.0401 & 1.0329 & 0.1602 & 2.9800 & 3.7869 \\ 2.0139 & 3.9830 & 0.1987 & 2.7792 & 1.0252 \end{array}\right]$$

Предсказания

Графики: 21 и 22.

4.4. Анализ результатов

Время обучения модели при различных параметрах n и L.

L	n=3	n=4	n=5
2	12.24	71.01	131.24
3	36.28	517.96	3078.82

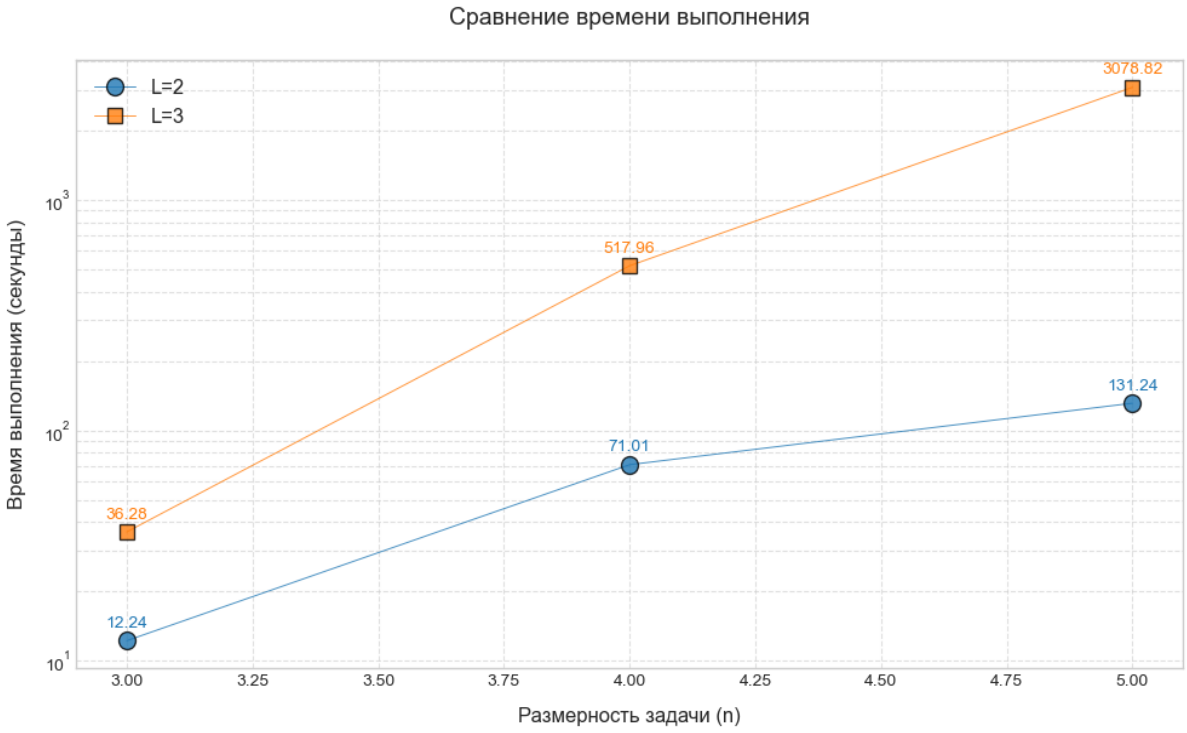

Рис. 8: График зависимости времени обучения нейросети от параметра n при фиксированном L

Сравнение алгоритмов проводится на столь ограниченном диапазоне параметров, так как при дальнейшем их увеличении объем набора данных нужно также увеличить, что требует более мощных вычислительных ресурсов.
Проведя сравнительный анализ, видим, что при фиксированном параметре $L = 2$ алгоритм демонстрирует увеличение времени в 11 раз при переходе с $n = 3$ к $n = 5$ и в 85 раз при $L = 3$ и таком же изменении n. Наблюдается рост сложности, близкий к экспоненциальному, откуда можем сделать вывод, что обучение такой нейросети крайне аппаратно затратно и выполнимо только на специализированных машинах, позволяющих вычислительное ускорение и располагающих большими запасами памяти.

5. Решение модельной задачи для глубокой сети. Её архитектура и принцип работы.

5.1. Генерация данных

Данные генерируются каждый раз при запуске программы перед началом обучения с помощью функции "generate_permutation_dataset". Для начала получим список всех перестановок для n объектов, длина которого, очевидно, будет $n!$. Все перестановки преобразуются в тензоры, так как это необходимо для работы с библиотекой "pytorch"и для обучения нейросети. Цикл запускается 5 тысяч раз, где случайным образом выбирается L штук перестановок из списка всевозможных перестановок, которые объединяются в один тензор. Целевая перестановка вычисляется как композиция, то есть последовательное применение всех L перестановок в одном тензоре. Затем последовательность из L перестановок и соответствующий ей ответ формируются в кортеж, который добавляется в датасет в качестве одного элемента. Таких элементов будет 5 тысяч. В зависимости от задачи количество элементов в наборе можно менять, то есть увеличивать или уменьшать размер набора.

5.2. Полносвязная сеть для двух матриц перестановок. Практическая реализация.

Задача решается с помощью базового компонента многих архитектур нейронных сетей - полносвязного линейного слоя. В такой сети каждый нейрон текущего слоя соединён с каждым нейроном следующего слоя. Каждый нейрон выполняет линейное преобразование, а именно вычисляет взвешенную сумму входов с коэффициентами и добавляет смещение. Функция активации ReLU зануляет отрицательные значения и делает нейросеть нелинейной, без чего невозможно обучение.
На входе получаем два вектора перестановки и преобразуем их в единый вектор. Далее он проходит через линейный слой и расширяет количество нейронов на скрытом слое до $3n^2$ (опираясь на теорему из статьи [1]). После чего данные пропускаются через ReLU и второй линейный слой, на выходе у которого n нейронов, что соответствует результату композиции перестановок.
Модель обучается также с использованием градиента. В качестве оптимизатора также выбран алгоритм Adam. Функция ошибки - MSELoss. В качестве оценки производительности на каждой эпохе обучения вычислялись такие метрики, как: loss, accuracy. Все метрики вычислялись на обучающей и тестовой выборках. Точность вычисляется так же, как и для неглубокой нейросети.

5.3. Динамическая полносвязная сеть для L матриц перестановок. Практическая реализация.

Эта задача решается также с использованием полносвязных линейных слоёв и функции активации ReLU. Данная реализация представляет глубокую нейронную сеть для обработки последовательности из L векторов. Сеть постепенно преобразовывает данные, пропуская их через серию линейных слоёв. Для каждого L-того элемента последовательности добавляется пара линейных слоёв: первый резко расширяет пространство признаков из $2n$ в $3n^2$ (опираясь на теорему из статьи [1]), второй сжимает представление обратно в количество значений в векторе перестановки n.
На первом шаге производится умножение двух векторов $Y_1 = X_1 \cdot X_2$ так, как это было реализовано в предыдущем варианте. Потом помещаем на новый слой результат от первого умножения Y_1 вместе с новым вектором X_3, получая новый вектор перестановки Y_2. В качестве ответа нейросети выдаём последний результат Y_{L-2} композиции всех L векторов.
Модель обучается также с использованием градиента. В качестве оптимизатора также выбран алгоритм Adam. Функция ошибки - MSELoss. В качестве оценки производительности на каждой эпохе обучения вычислялись такие метрики, как: loss, accuracy. Все метрики вычислялись на обучающей и тестовой выборках. Точность вычисляется так же, как и для неглубокой нейросети.

5.4. Результаты экспериментов при различных параметрах задачи

5.4.1. Результаты при $L = 2$

1. $n = 3, L = 2$

   Параметры обучения: learning rate = 0.01, batch size = 32, количество эпох = 3

   Время обучения: 3.54 сек.

Таблица 7: Результаты обучения модели по эпохам

Epoch	Train Loss	Test Loss	Train Accuracy	Test Accuracy
0	0.3375	0.1068	0.6525	0.9243
1	0.0344	0.0037	0.9848	1.0000
2	0.0017	0.0008	1.0000	1.0000
3	0.0005	0.0003	1.0000	1.0000

$$\left[\begin{array}{ccc} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{ccc} 1.9802 & 0.0110 & 0.9982 \\ 0.0182 & 1.9793 & 1.0191 \\ 1.0100 & 1.9974 & -0.0165 \\ 0.9835 & 0.0058 & 2.0226 \end{array}\right]$$

Предсказания

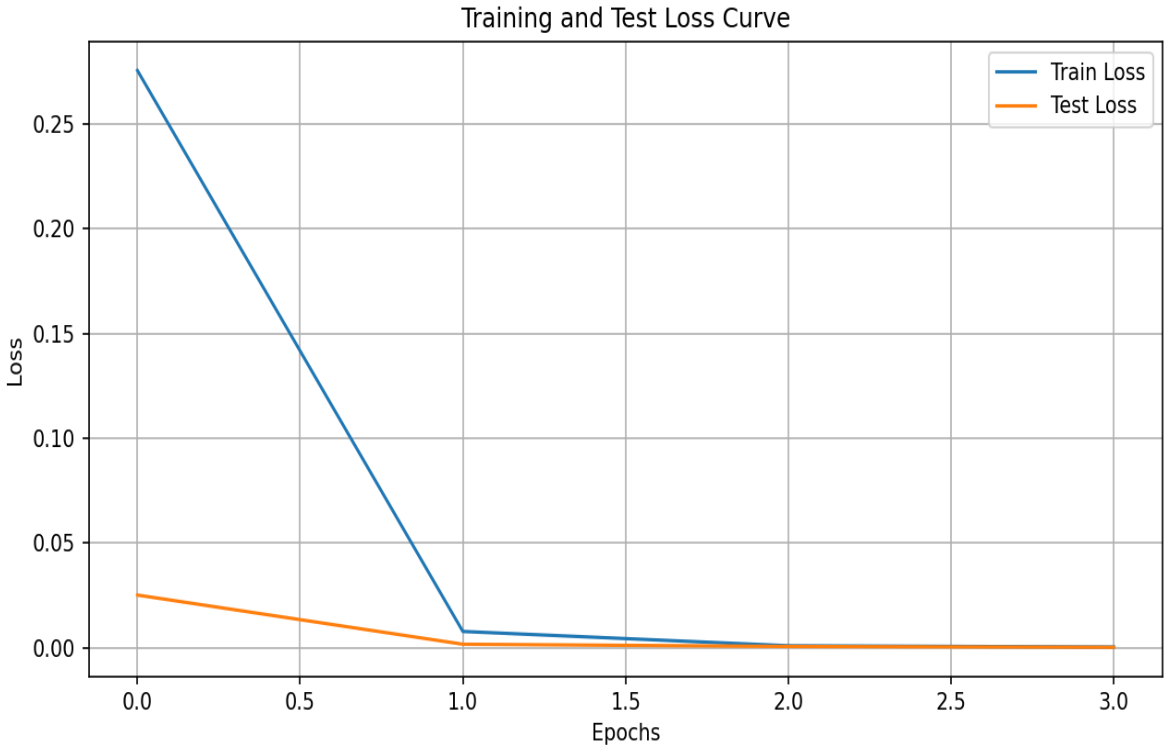

Рис. 9: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 2$

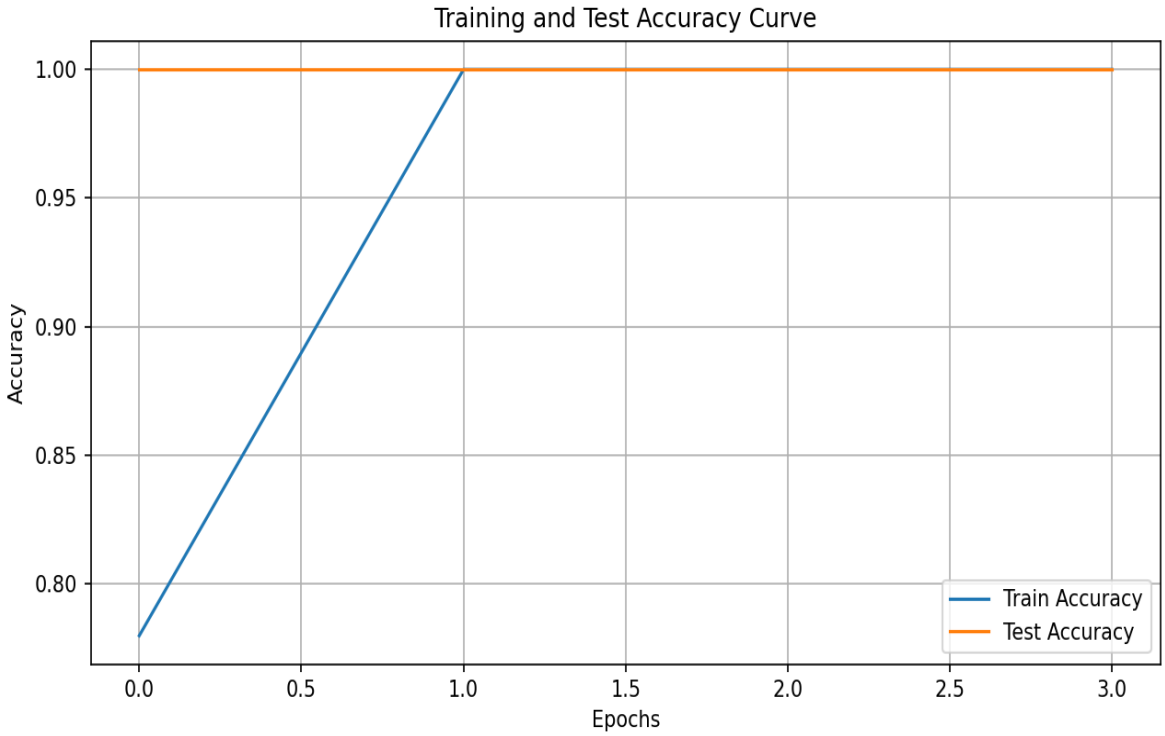

Рис. 10: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 2$

2. $n = 4, L = 2$

   Параметры обучения: learning rate = 0.01, batch size = 32, количество эпох = 5

   Время обучения: 6.03 сек.

Таблица 8: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	0.9640	0.6132	0.3777	0.5332
1	0.3982	0.2030	0.6803	0.8562
2	0.1153	0.0471	0.9330	0.9960
3	0.0281	0.0131	0.9975	1.0000
4	0.0111	0.0073	1.0000	1.0000

$$\left[\begin{array}{cccc} 1 & 3 & 2 & 0 \\ 0 & 3 & 1 & 2 \\ 0 & 3 & 2 & 1 \\ 0 & 3 & 2 & 1 \end{array}\right] \qquad \left[\begin{array}{cccc} 1.0387 & 2.9495 & 1.8861 & 0.1080 \\ 0.0439 & 3.0270 & 1.0702 & 1.8618 \\ 0.0040 & 3.0385 & 2.0130 & 0.8648 \\ 0.0256 & 3.0235 & 2.0596 & 0.9747 \end{array}\right]$$

Настоящие ответы Предсказания

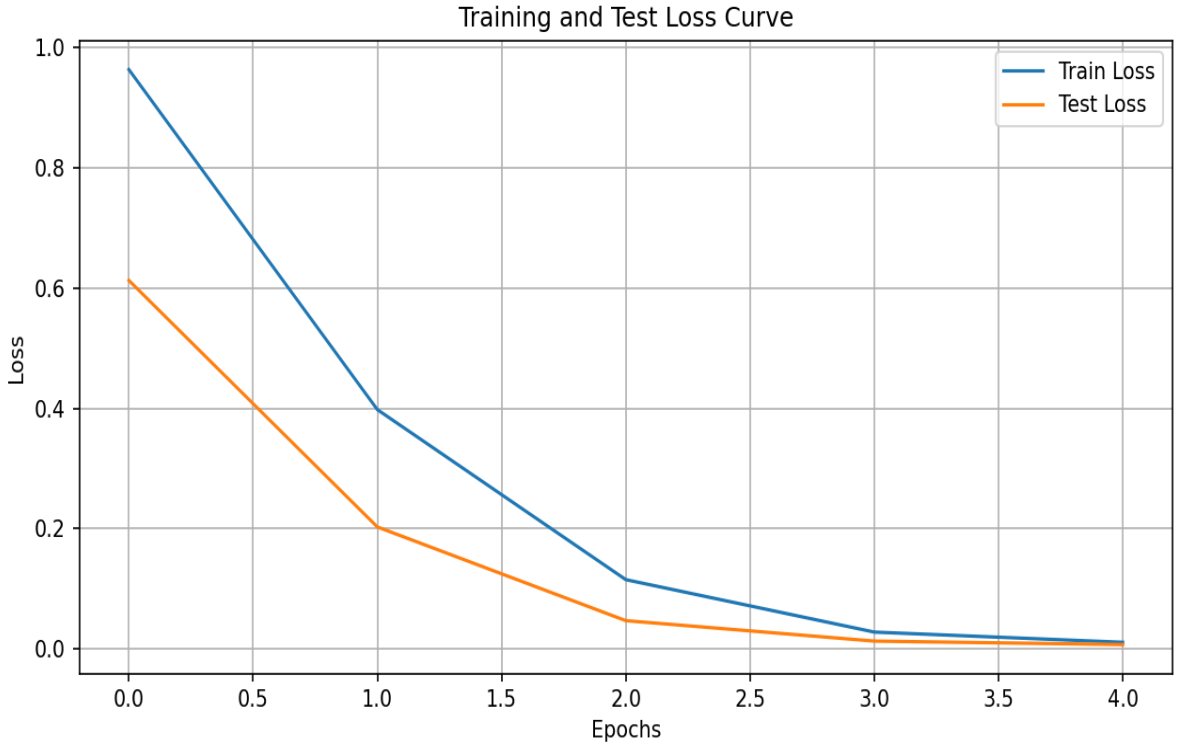

Рис. 11: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 2$

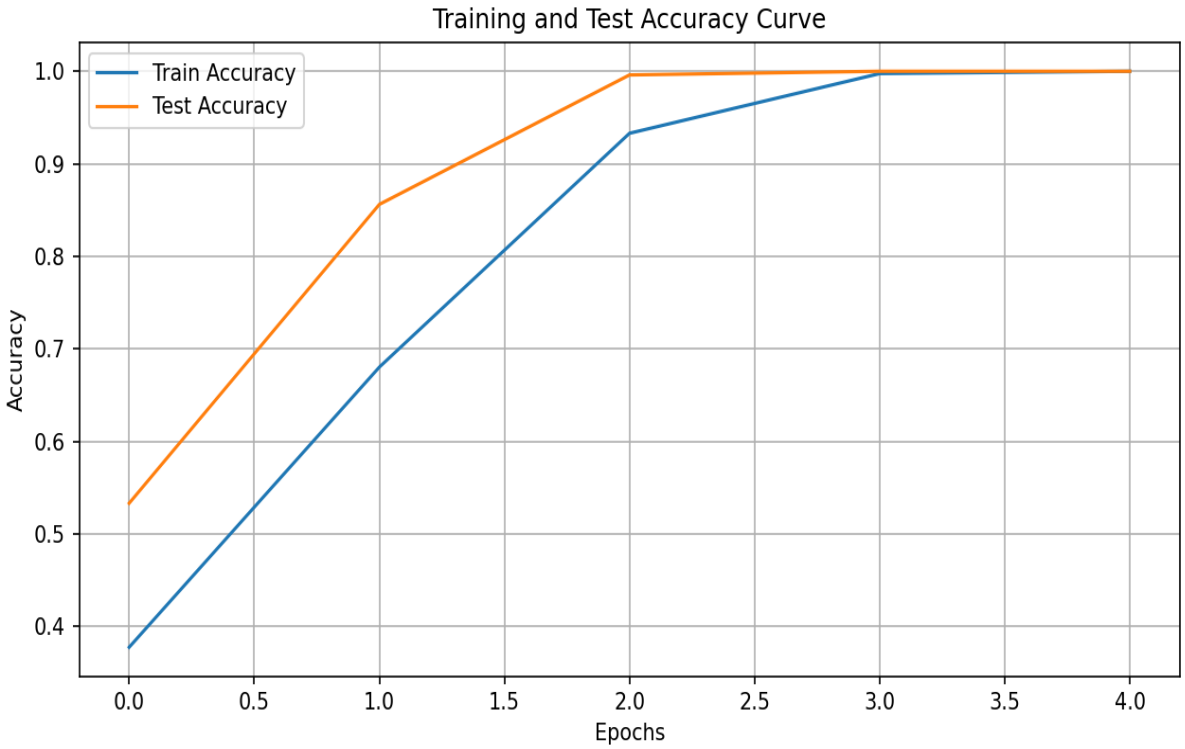

Рис. 12: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 2$

3. $n = 5, L = 2$

   Параметры обучения: learning rate = 0.0055, batch size = 32, количество эпох = 11.

   Время обучения: 13.42 сек.

Таблица 9: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	1.8251	1.5288	0.2654	0.2963
1	1.3502	1.1972	0.3272	0.3599
2	1.0130	0.8982	0.4189	0.4742
3	0.7320	0.6142	0.5343	0.5889
⋮	⋮	⋮	⋮	⋮
8	0.0450	0.0360	0.9902	0.9927
9	0.0283	0.0313	0.9964	0.9972
10	0.0182	0.0155	0.9994	0.9998
11	0.0124	0.0133	1.0000	1.0000

$$\left[\begin{array}{ccccc} 2 & 3 & 4 & 1 & 0 \\ 4 & 1 & 2 & 0 & 3 \\ 1 & 3 & 4 & 2 & 0 \\ 0 & 4 & 2 & 1 & 3 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{ccccc} 2.0139 & 2.9784 & 3.9727 & 1.0392 & 0.0142 \\ 4.0706 & 1.1129 & 1.8418 & 0.0066 & 2.9877 \\ 1.1016 & 2.7675 & 4.0397 & 2.0344 & 0.0433 \\ 0.0154 & 4.0829 & 2.0197 & 0.8927 & 2.9775 \end{array}\right]$$

Предсказания

Последующие графики будут в приложении, так как они выглядят идентично тем, что уже представлены в предыдущих результатах. Графики: 23 и 24.

4. $n = 6, L = 2$

   Параметры обучения: learning rate = 0.0055, batch size = 32, количество эпох = 21.

   Время обучения: 48.48 сек.

Таблица 10: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	2.7334	2.4419	0.2095	0.2201
1	2.3187	2.2034	0.2402	0.2482
2	1.9946	1.8575	0.2706	0.2822
3	1.7069	1.6092	0.3120	0.3247
4	1.4515	1.3457	0.3575	0.3745
5	1.2007	1.0922	0.4115	0.4484
6	1.0003	0.8922	0.4706	0.5074
⋮	⋮	⋮	⋮	⋮
17	0.0394	0.0488	0.9928	0.9924
18	0.0323	0.0240	0.9959	0.9970
19	0.0467	0.0213	0.9871	0.9987
20	0.0224	0.0156	0.9986	0.9987

$$\left[\begin{array}{cccccc} 3 & 4 & 2 & 0 & 5 & 1 \\ 1 & 5 & 2 & 3 & 4 & 0 \\ 3 & 2 & 4 & 0 & 1 & 5 \\ 4 & 0 & 2 & 1 & 5 & 3 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{cccccc} 2.8847 & 4.0597 & 1.9901 & 0.1853 & 4.9268 & 0.9794 \\ 1.1333 & 5.1555 & 1.8767 & 3.1787 & 3.8702 & 0.0657 \\ 3.0470 & 1.8880 & 4.1017 & 0.1487 & 1.1243 & 5.0339 \\ 3.9773 & 0.0875 & 2.0286 & 0.9635 & 4.9936 & 3.0887 \end{array}\right]$$

Предсказания

Графики: 25 и 26.

5. $n = 7, L = 2$

   Параметры обучения: learning rate = 0.0055, batch size = 32, количество эпох = 35.

   Время обучения: 98.05 сек.

$$\left[\begin{array}{ccccccc} 4 & 6 & 5 & 1 & 2 & 0 & 3 \\ 2 & 6 & 1 & 5 & 3 & 0 & 4 \\ 2 & 0 & 6 & 3 & 4 & 5 & 1 \\ 2 & 0 & 3 & 4 & 6 & 1 & 5 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{ccccccc} 4.021 & 5.348 & 5.117 & 1.267 & 1.945 & 0.341 & 2.988 \\ 1.974 & 6.138 & 1.144 & 5.031 & 2.950 & 0.192 & 3.984 \\ 1.956 & 0.015 & 6.14 & 3.115 & 3.55 & 4.985 & 1.436 \\ 1.927 & 0.225 & 2.974 & 3.995 & 6.029 & 0.994 & 5.075 \end{array}\right]$$

Предсказания

Графики: 27 и 28.

Таблица 11: Результаты обучения модели по эпохам

Epoch	Train Loss	Test Loss	Train Accuracy	Test Accuracy
0	3.9029	3.5220	0.1686	0.1901
1	3.4137	3.3378	0.1908	0.1946
2	3.1796	3.0892	0.2090	0.2100
3	2.9054	2.8389	0.2246	0.2268
4	2.6941	2.6874	0.2390	0.2390
⋮	⋮	⋮	⋮	⋮
26	0.1092	0.1207	0.9505	0.9427
27	0.1107	0.1003	0.9488	0.9588
28	0.0921	0.0806	0.9642	0.9737
29	0.0987	0.0880	0.9585	0.9680
30	0.0737	0.0971	0.9764	0.9649
31	0.0710	0.0657	0.9770	0.9847
32	0.0657	0.0990	0.9814	0.9580
33	0.0910	0.0570	0.9639	0.9876
34	0.0531	0.0592	0.9878	0.9840

5.4.2. Результаты при $L = 3$

1. $n = 3, L = 3$

   Параметры обучения: learning rate = 0.0055, batch size = 32, количество эпох = 3

   Время обучения: 4.17 сек.

Таблица 12: Результаты обучения модели по эпохам

Epoch	Train Loss	Test Loss	Train Accuracy	Test Accuracy
0	0.4420	0.0080	0.5814	1.0000
1	0.0010	0.0001	1.0000	1.0000
2	0.0001	0.0001	1.0000	1.0000

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 0.0043 & 0.9893 & 1.9976 \\ 1.0015 & 1.9946 & 0.0009 \\ 1.0034 & 0.0097 & 2.0020 \\ 1.0147 & 2.0029 & 0.0059 \end{bmatrix}$$

Предсказания

Графики: 29 и 30.

2. $n = 4, L = 3$

   Параметры обучения: learning rate = 0.001, batch size = 32, количество эпох = 9

   Время обучения: 19.71 сек.

Таблица 13: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Acc.**	**Test Acc.**
0	1.2945	1.2612	0.2549	0.2674
1	1.1843	1.1358	0.2943	0.3027
2	1.0416	0.9786	0.3434	0.3597
3	0.8804	0.8196	0.3957	0.4218
4	0.6846	0.5707	0.4856	0.5420
5	0.4320	0.3225	0.6403	0.7424
6	0.1961	0.1040	0.8614	0.9619
7	0.0595	0.0290	0.9865	0.9988
8	0.0176	0.0107	0.9998	1.0000

$$\left[\begin{array}{cccc} 2 & 3 & 0 & 1 \\ 2 & 0 & 1 & 3 \\ 2 & 0 & 3 & 1 \\ 0 & 3 & 2 & 1 \end{array}\right]$$

Настоящие ответы

$$\left[\begin{array}{cccc} 1.9942 & 2.9472 & 0.0592 & 0.9597 \\ 2.0065 & 0.0095 & 1.0408 & 2.9021 \\ 2.0703 & 0.0533 & 3.0108 & 0.8796 \\ 0.0213 & 2.8689 & 1.9730 & 1.1261 \end{array}\right]$$

Предсказания

Графики: 31 и 32.

3. $n = 5, L = 3$

   Параметры обучения: learning rate = 0.005, batch size = 32, количество эпох = 20

   Время обучения: 29.06 сек.

Таблица 14: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Acc.**	**Test Acc.**
0	2.1931	2.0411	0.1992	0.2080
1	2.0258	2.0020	0.2071	0.2200
2	1.9727	1.9482	0.2206	0.2313
3	1.8894	1.8391	0.2402	0.2459
4	1.7938	1.7630	0.2570	0.2705
5	1.6972	1.6600	0.2757	0.2796
⋮	⋮	⋮	⋮	⋮
16	0.1273	0.1087	0.9313	0.9497
17	0.1303	0.0829	0.9306	0.9687
18	0.0676	0.0907	0.9746	0.9580
19	0.0511	0.0598	0.9839	0.9782

$$\begin{bmatrix} 3 & 2 & 0 & 1 & 4 \\ 1 & 4 & 0 & 2 & 3 \\ 0 & 1 & 4 & 3 & 2 \\ 3 & 4 & 2 & 1 & 0 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 2.6773 & 1.9667 & 0.0794 & 1.0208 & 3.9423 \\ 1.1464 & 4.1975 & 0.0179 & 1.8267 & 2.7920 \\ 0.1804 & 1.2854 & 3.9514 & 2.9455 & 1.6753 \\ 2.8122 & 4.0659 & 2.1013 & 1.1358 & 0.0558 \end{bmatrix}$$

Предсказания

Графики: 33 и 34.

5.4.3. Результаты при $L = 4$

1. $n = 3, L = 4$ Параметры обучения: learning rate = 0.0007, batch size = 32, количество эпох = 13

   Время обучения: 27.2 сек.

Таблица 15: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	0.7019	0.6695	0.3341	0.3312
1	0.6674	0.6602	0.3410	0.3595
2	0.6191	0.5718	0.4024	0.4534
3	0.5409	0.4895	0.4582	0.4866
4	0.4207	0.3420	0.5528	0.5934
5	0.2972	0.2487	0.6780	0.7710
6	0.2094	0.1649	0.8165	0.8634
7	0.1387	0.1119	0.9028	0.9368
8	0.0916	0.0699	0.9523	0.9735
9	0.0545	0.0355	0.9721	0.9759
10	0.0242	0.0140	0.9905	1.0000
11	0.0089	0.0053	1.0000	1.0000
12	0.0038	0.0028	1.0000	1.0000

$$\begin{bmatrix} 0 & 2 & 1 \\ 2 & 0 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 0.0656 & 1.9487 & 1.0372 \\ 2.0368 & 0.0441 & 1.0161 \\ 0.0814 & 1.9573 & 0.9505 \\ 0.0697 & 0.9437 & 1.9826 \end{bmatrix}$$

Предсказания

   Графики: 35 и 36.

2. $n = 4, L = 4$ Параметры обучения: learning rate = 0.001, batch size = 32, количество эпох = 50

   Время обучения: 207.76 сек.

Таблица 16: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	1.2838	1.2617	0.2489	0.2496
1	1.2653	1.2721	0.2496	0.2547
2	1.2618	1.2547	0.2574	0.2621
3	1.2503	1.2368	0.2642	0.2700
4	1.2226	1.2000	0.2740	0.2838
⋮	⋮	⋮	⋮	⋮
46	0.0586	0.0608	0.9783	0.9764
47	0.0486	0.0584	0.9848	0.9796
48	0.0522	0.0783	0.9824	0.9611
49	0.0383	0.0528	0.9912	0.9788

$$\begin{bmatrix} 2 & 0 & 1 & 3 \\ 0 & 3 & 1 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 2 & 1 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 1.9313 & 0.0264 & 1.0437 & 2.9976 \\ 0.0268 & 2.9180 & 0.9936 & 1.9558 \\ 2.0028 & 2.9067 & 0.0451 & 0.9903 \\ 2.9037 & 0.0286 & 2.0377 & 1.0325 \end{bmatrix}$$

Предсказания

Графики: 37 и 38.

5.4.4. Результаты при $L = 5$

1. $n = 3, L = 5$

   Параметры обучения: learning rate = 0.001, batch size = 32, количество эпох = 31

   Время обучения: 142.22 сек.

Таблица 17: Результаты обучения модели по эпохам

Epoch	**Train Loss**	**Test Loss**	**Train Accuracy**	**Test Accuracy**
0	0.7103	0.6672	0.3285	0.3368
1	0.6703	0.6855	0.3327	0.3388
2	0.6706	0.6711	0.3347	0.3218
3	0.6712	0.6687	0.3338	0.3338
4	0.6712	0.6704	0.3302	0.3299
5	0.6691	0.6685	0.3375	0.3317
6	0.6706	0.6698	0.3303	0.3345
7	0.6652	0.6517	0.3464	0.3707
⋮	⋮	⋮	⋮	⋮
29	0.0071	0.0063	1.0000	1.0000
30	0.0049	0.0038	1.0000	1.0000

$$\begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

Настоящие ответы

$$\begin{bmatrix} 0.0512 & 2.0186 & 0.9559 \\ 0.0468 & 2.0171 & 0.8786 \\ 0.0282 & 1.0358 & 1.9122 \\ 2.0331 & 1.0005 & 0.0135 \end{bmatrix}$$

Предсказания

Графики: 39 и 40.

5.5. Анализ результатов

Время обучения модели при различных параметрах n и L.

L	n=3	n=4	n=5	n=6
2	3.54	6.03	13.42	48.48
3	4.17	19.71	29.06	
4	27.2	207.76		
5	142.22			

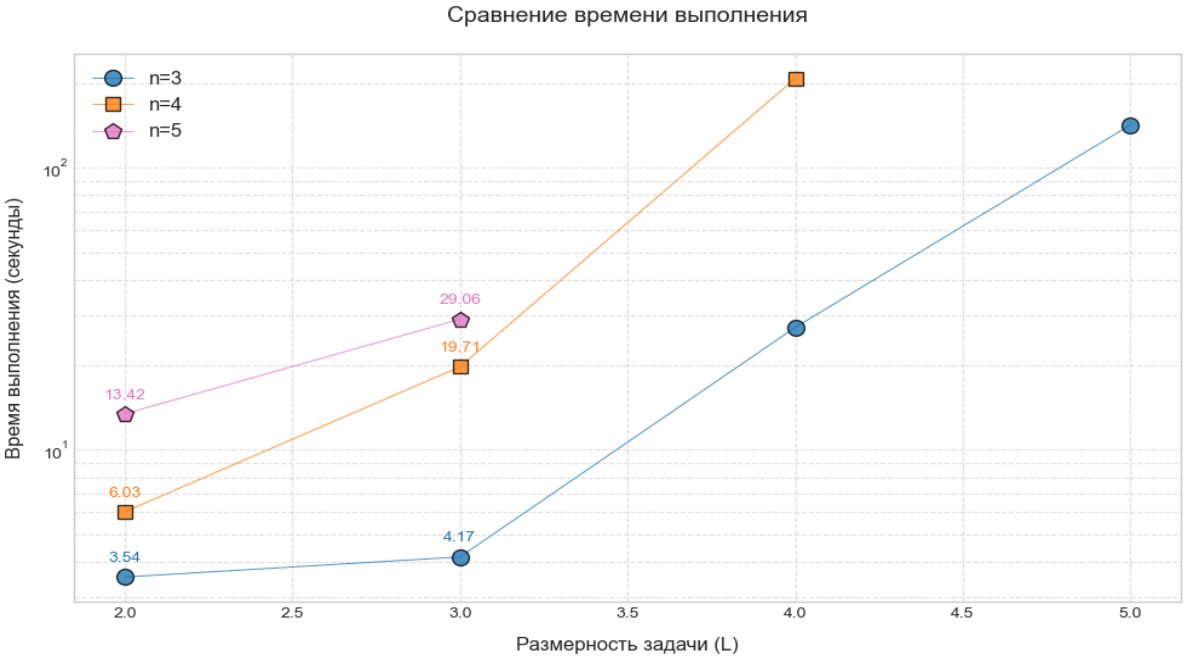

Рис. 13: График зависимости времени обучения нейросети от параметра L при фиксированном n

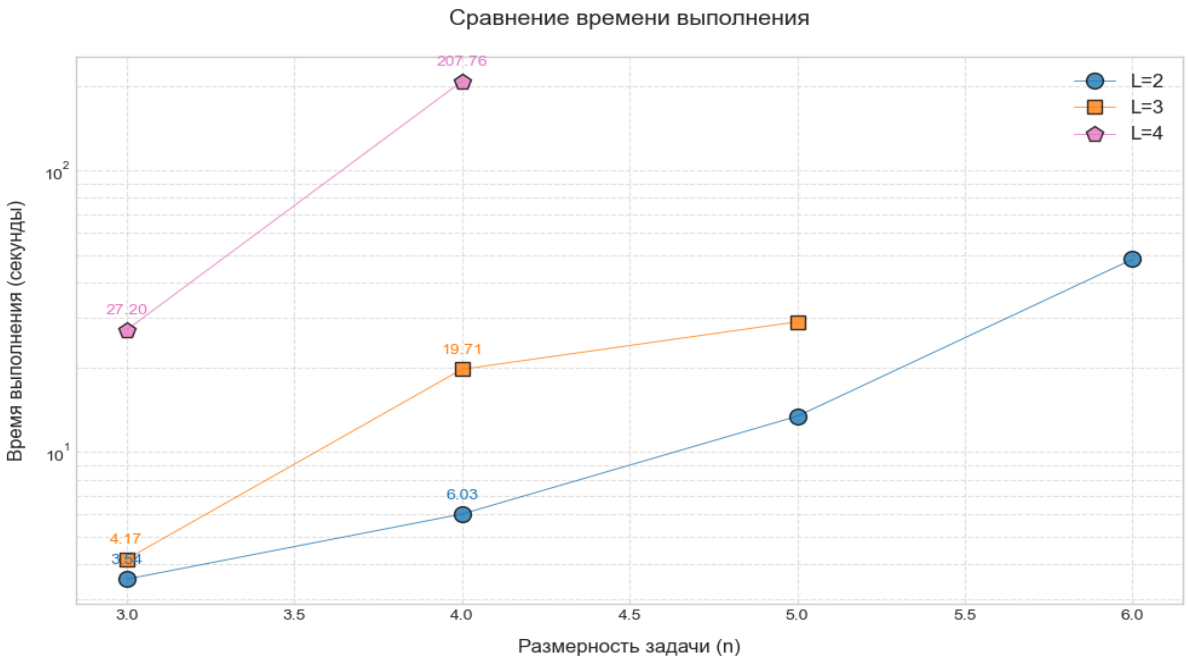

Рис. 14: График зависимости времени обучения нейросети от параметра n при фиксированном L

Сравнение алгоритмов проводится на представленном наборе параметров, так как, так же, как и для неглубокой нейросети, при дальнейшем их увеличении объем набора данных нужно также увеличить, что требует более мощных вычислительных ресурсов.
Проведя сравнительный анализ, видим, что при фиксированном параметре $n = 3$ алгоритм демонстрирует увеличение времени в 40 раз при переходе с $L = 2$ к $L = 5$, в 34 раза при $n = 4$ и изменении L от 2 до 4 и в 2 раза при изменении L от 2 до 3. Наблюдается рост сложности, близкий к полиномиальному или полиномиально-экспоненциальному, откуда можем сделать вывод, что обучение такой нейросети крайне аппаратно затратно и выполнимо только на специализированных машинах, позволяющих вычислительное ускорение и располагающих большими запасами памяти.

6. Выводы и рекомендации

В результате можем сделать вывод, что хотя обе нейросети довольно успешно обучаются и предсказывают результат перестановки, они оказались затратными, требуют много памяти и высокой мощности. Однако можно сделать ряд обобщений и предположений о дальнейшем использовании этих алгоритмов и их оптимизации, о том, какой нейросетью лучше пользоваться в зависимости от задачи.
Сложность глубокой сети наиболее возрастает с увеличением n, но не так сильно, как сложность глубокой сети с увеличением L. Соответственно, если более важен параметр n - количество объектов, то нужно выбирать неглубокую нейросеть, при этом если важно L - глубина лабиринта, лучше пользоваться глубокой нейросетью.
В качестве датасета для неглубокой сети используются картинки лабиринтов, а так как ресурсы ограничены, разрешение на этих картинках невелико, что замедляет обучение. Увеличивая параметры, мы сталкиваемся с большим набором признаков (features), что заставляет увеличивать датасет, иначе модель будет переобучаться. От этого ограничения, как и от многих других, можно избавиться, применив суперкомпьютеры с большими ЭВМ. Также можно добавить некоторые оптимизации к самим алгоритмам, например, использовать вместо полносвязных слоев более хитрые архитектуры, воспользоваться концепцией skip-connections, что помогает решить проблему затухания градиентов, которая появляется при увеличении глубины лабиринта. Есть также множество других распространённых методов для ускорения работы моделей (квантизация весов, прореживание и др.). В дальнейшем можно рассмотреть такие варианты, как переход на совсем другие архитектуры (рекуррентные сети, трансформеры).

7. Литература

Список литературы

[1] Alexander Kirdin; Sergey Sidorov; Nikolai Zolotykh *Rosenblatt's First Theorem and Frugality of Deep Learning*. Entropy 2022, 24, 1635. [MDPI]

[2] Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books: Washington, DC, USA, 1962. [Google Scholar]

[3] Minsky, M.; Papert, S. *Perceptrons*. MIT Press: Cambridge, MA, USA,1988. [Google Scholar]

[4] Gorban, A.N.; Mirkes, E. M.; Tyukin, I.Y. *How deep should be the depth of convolutional neural networks: A backyard dog case study.*. Cogn. Comput. 2020, 12, 388–397. [Google Scholar] [CrossRef][Green Version]

[5] Bianchini, M.; Scarselli, F. *On the complexity of neural network classifiers: A comparison between shallow and deep architectures.* IEEE Trans. Neural Netw. Learn. Syst. 2014, 25, 1553–1565. [Google Scholar] [CrossRef][PubMed]

8. Приложение

A. Графики

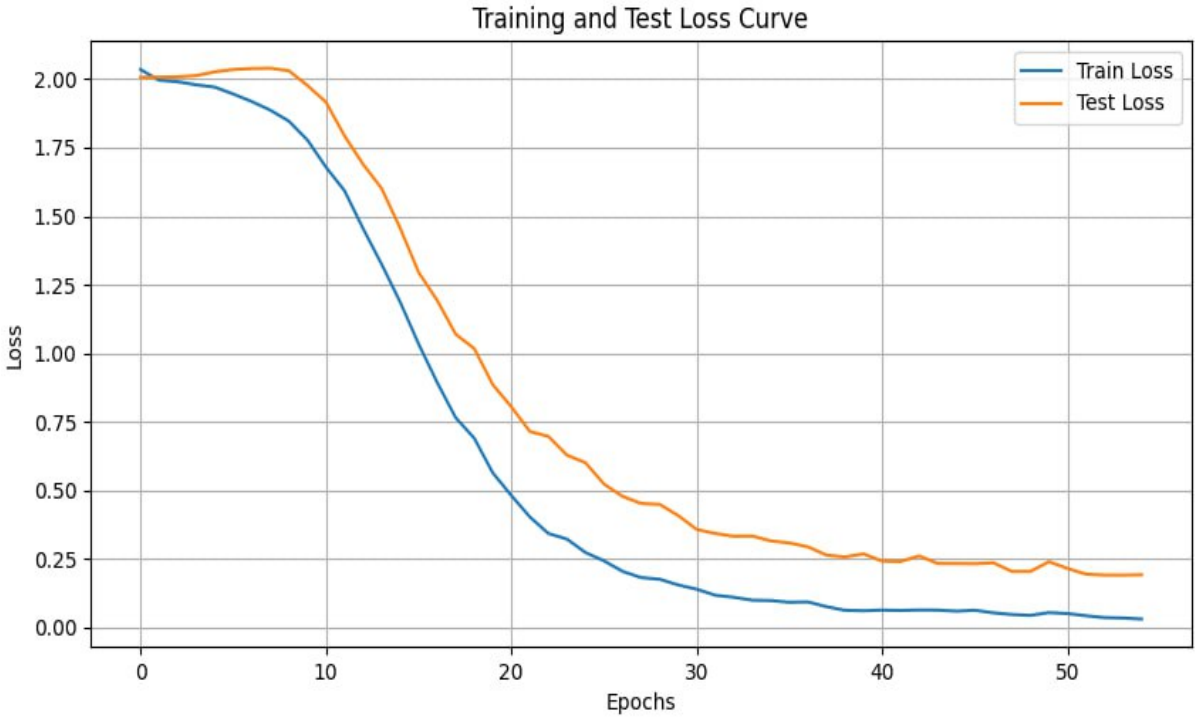

Рис. 15: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 2$

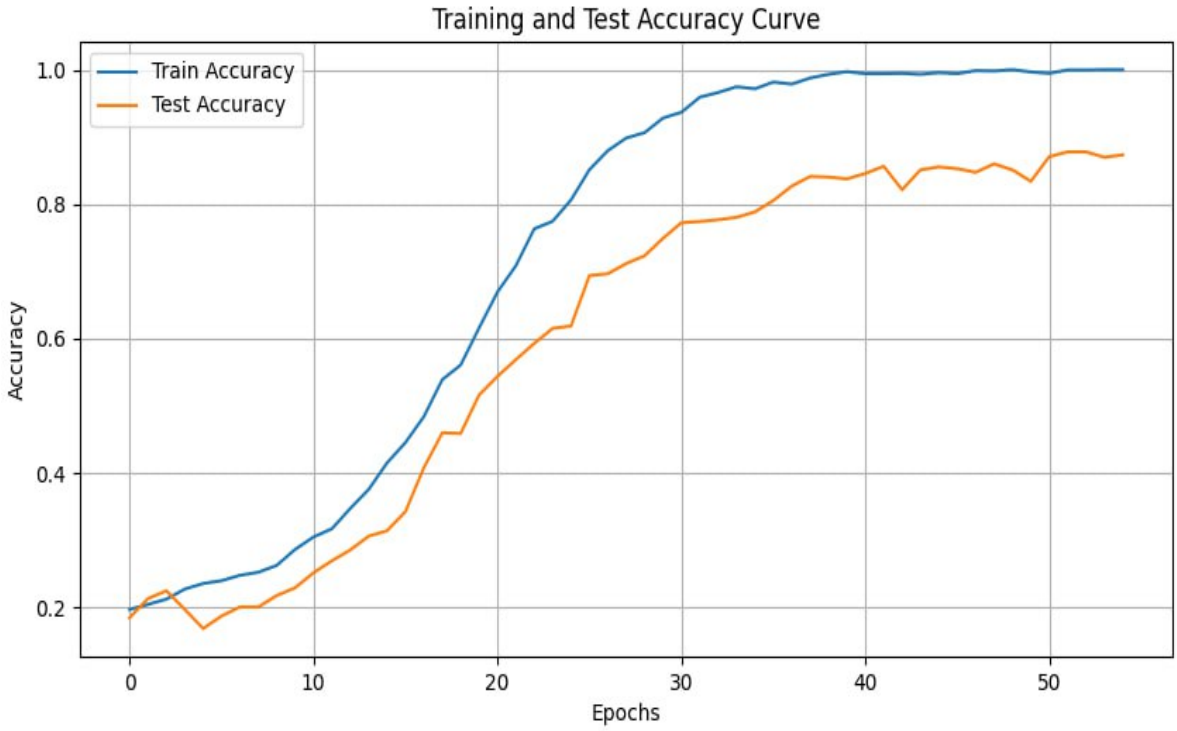

Рис. 16: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 2$

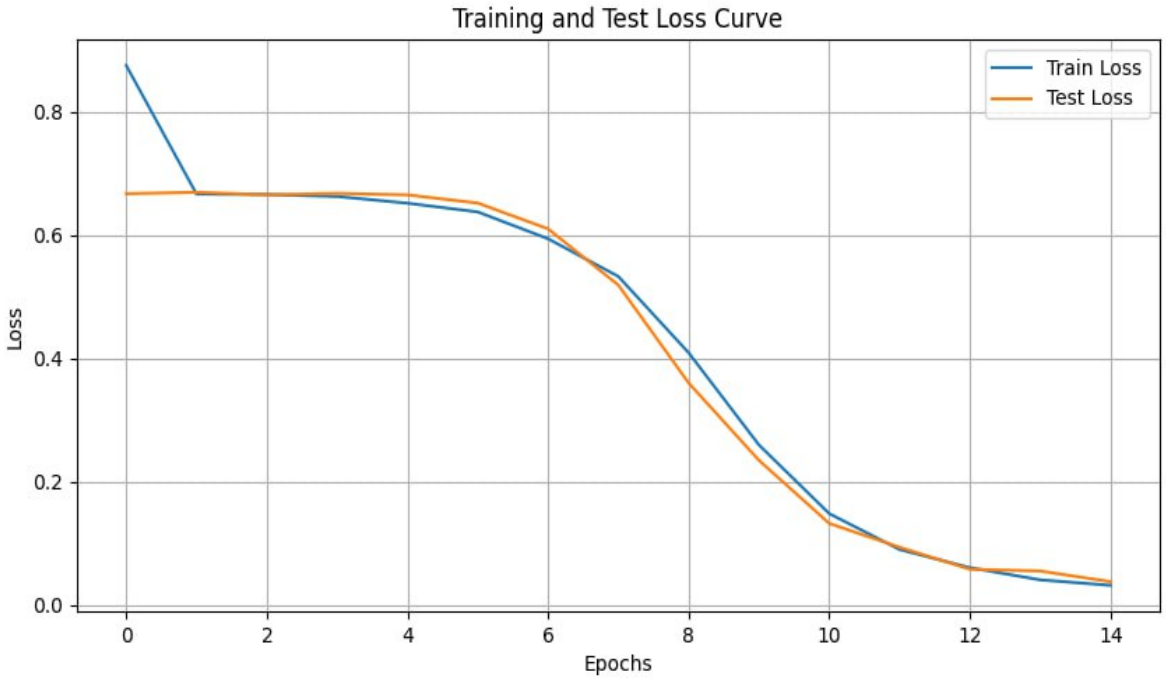

Рис. 17: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 3$

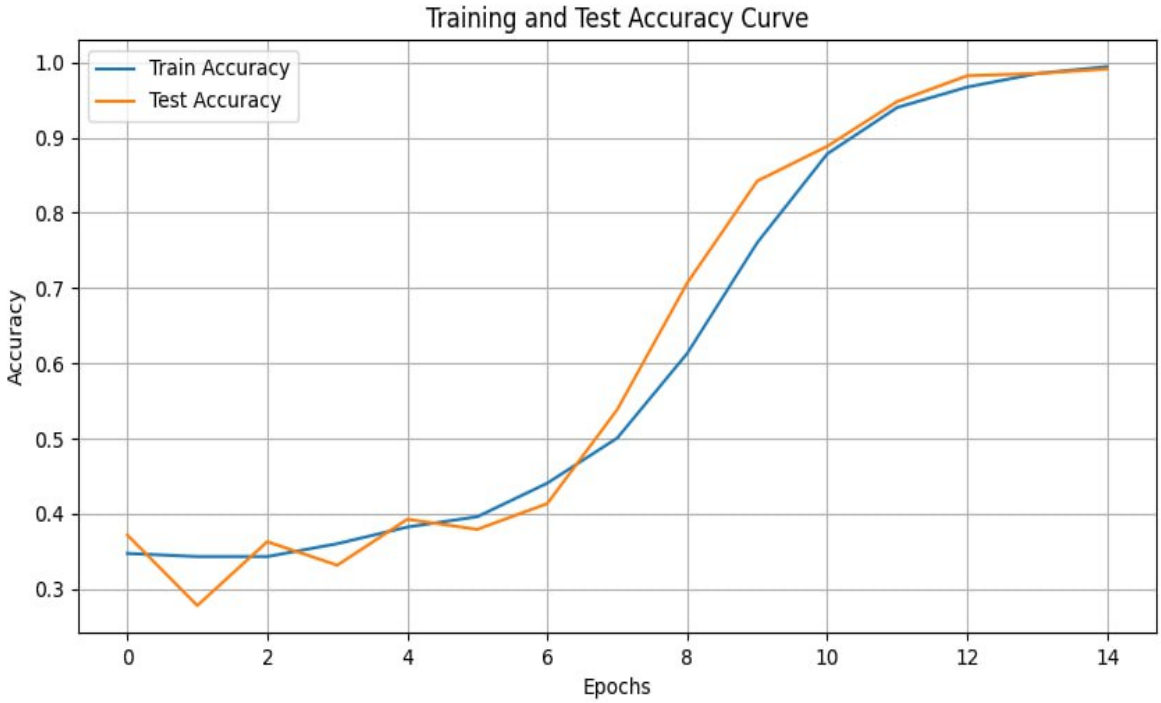

Рис. 18: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 3$

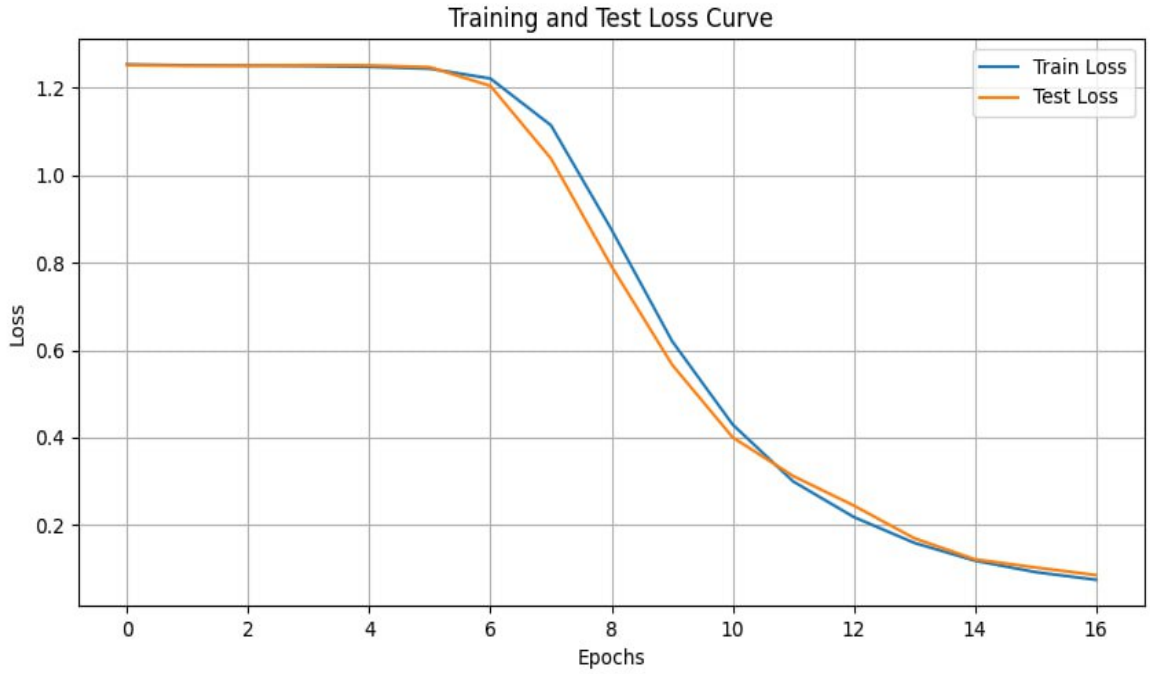

Рис. 19: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 3$

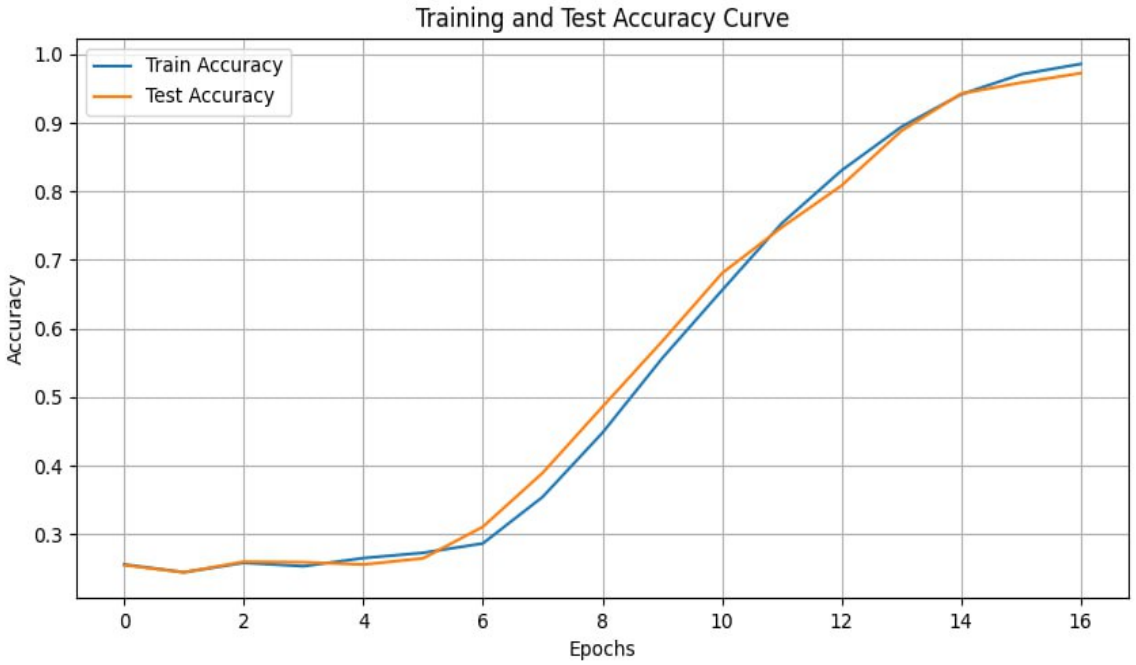

Рис. 20: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 3$

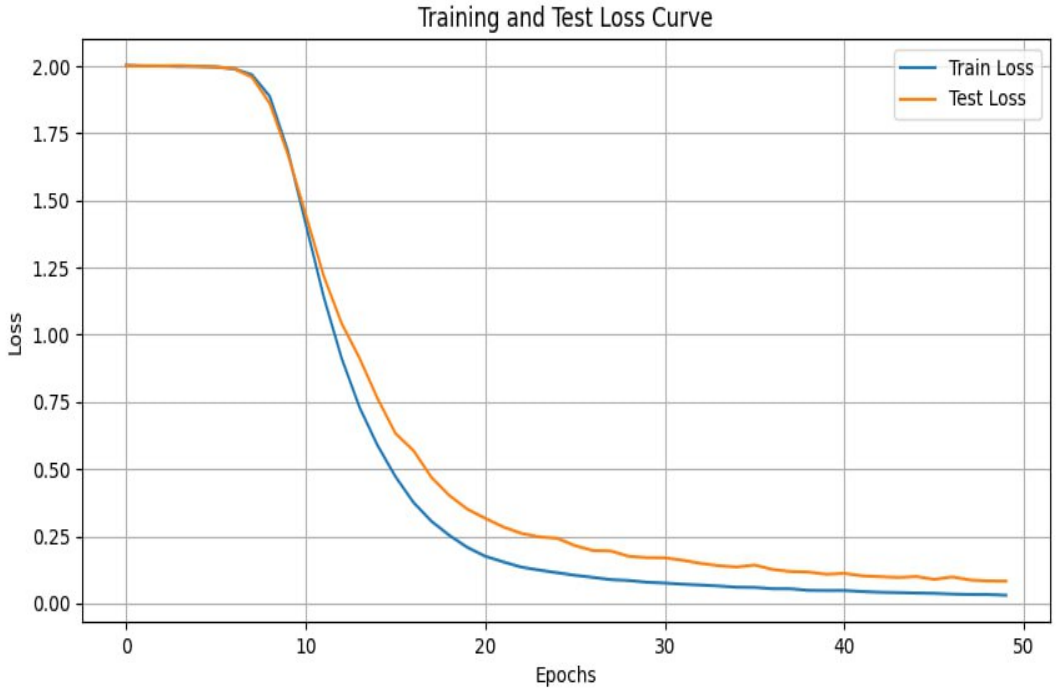

Рис. 21: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 3$

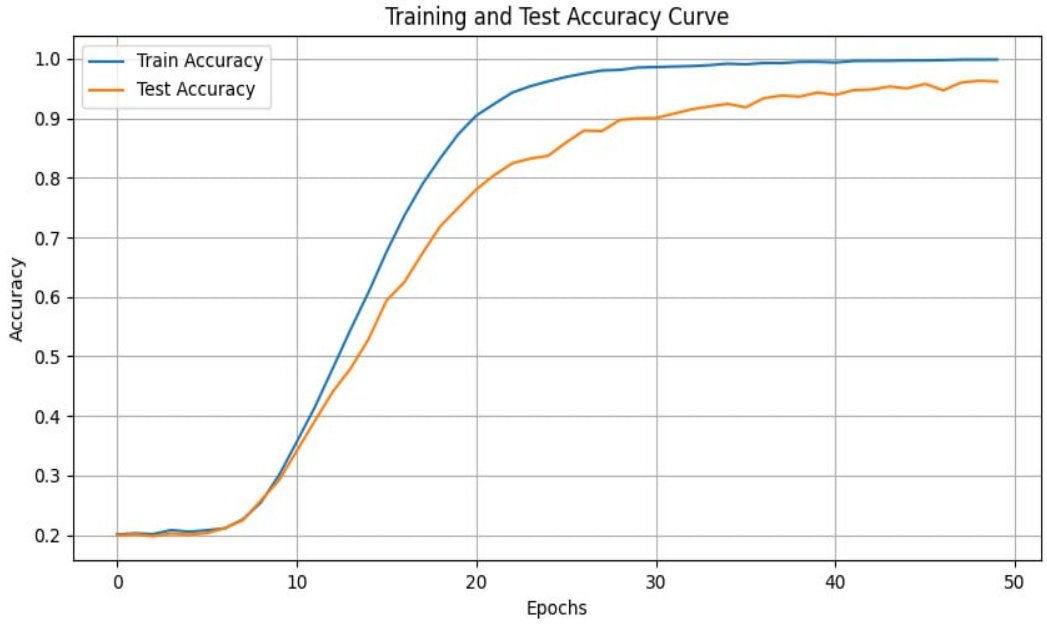

Рис. 22: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 3$

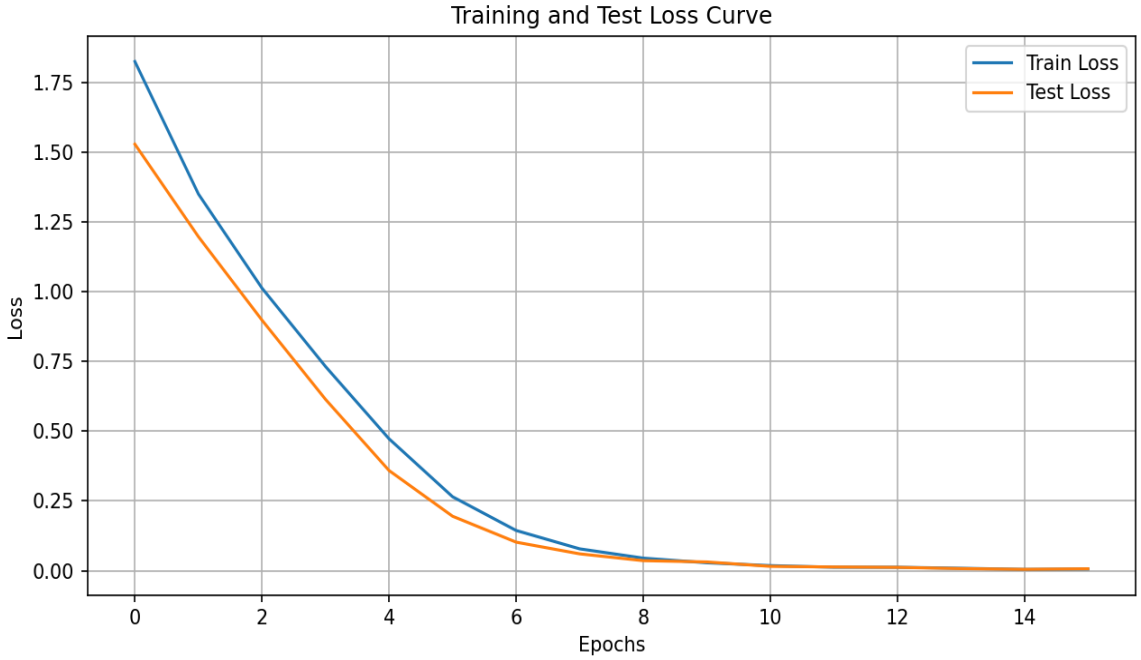

Рис. 23: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 2$

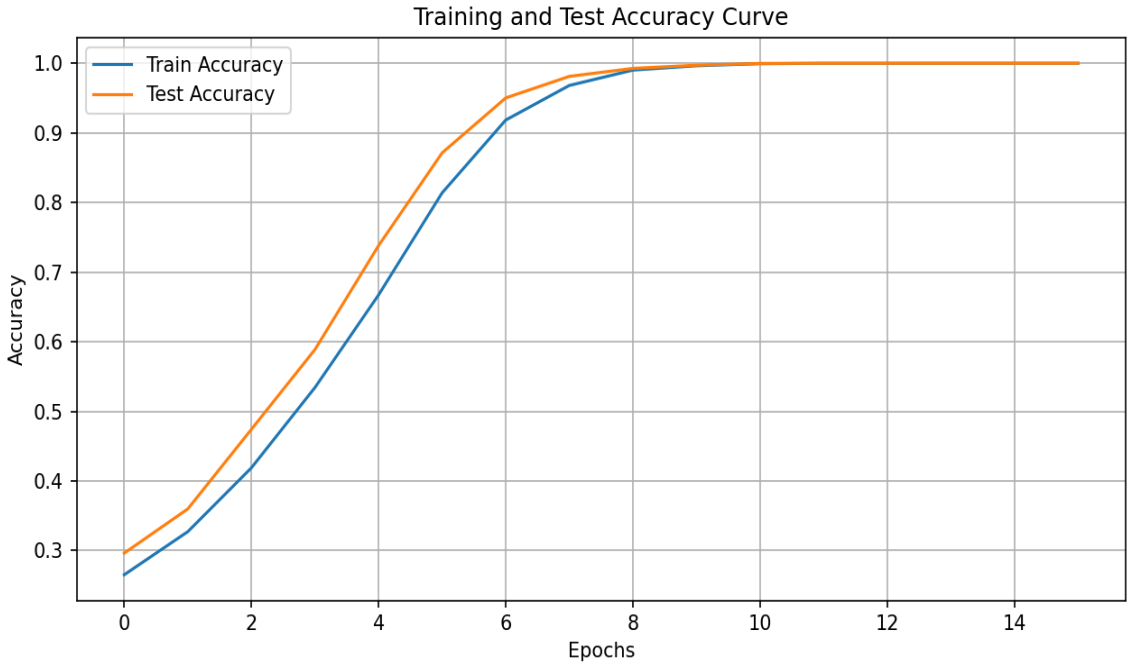

Рис. 24: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 2$

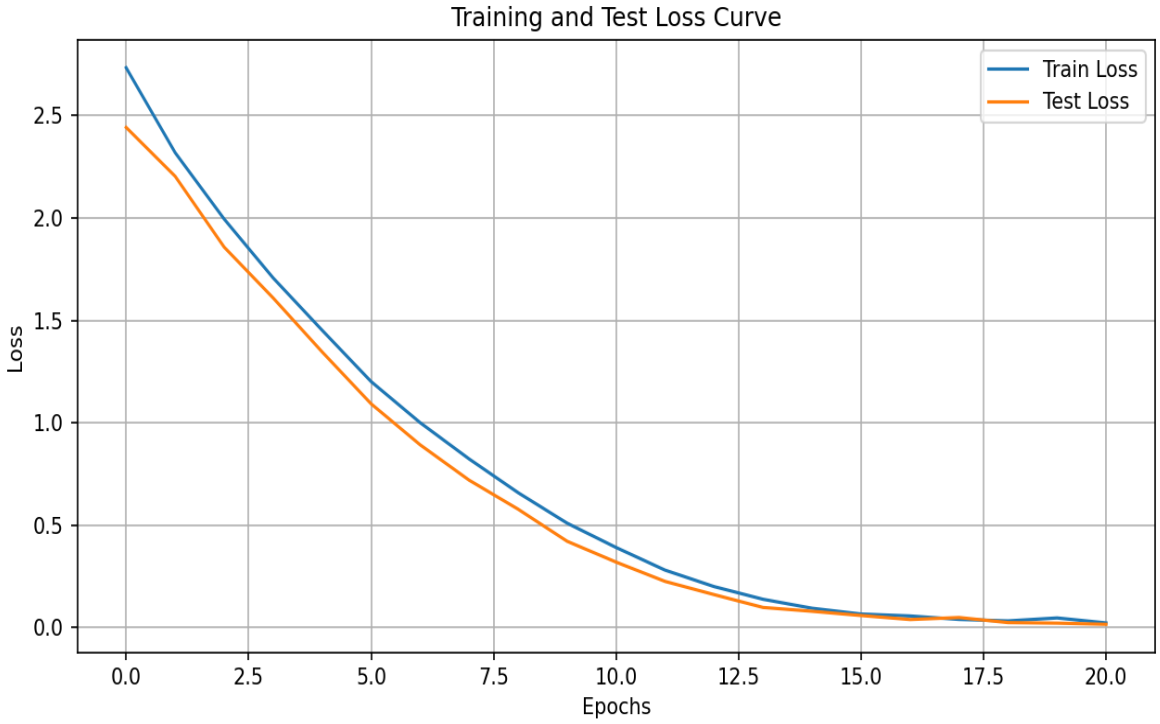

Рис. 25: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 6, L = 2$

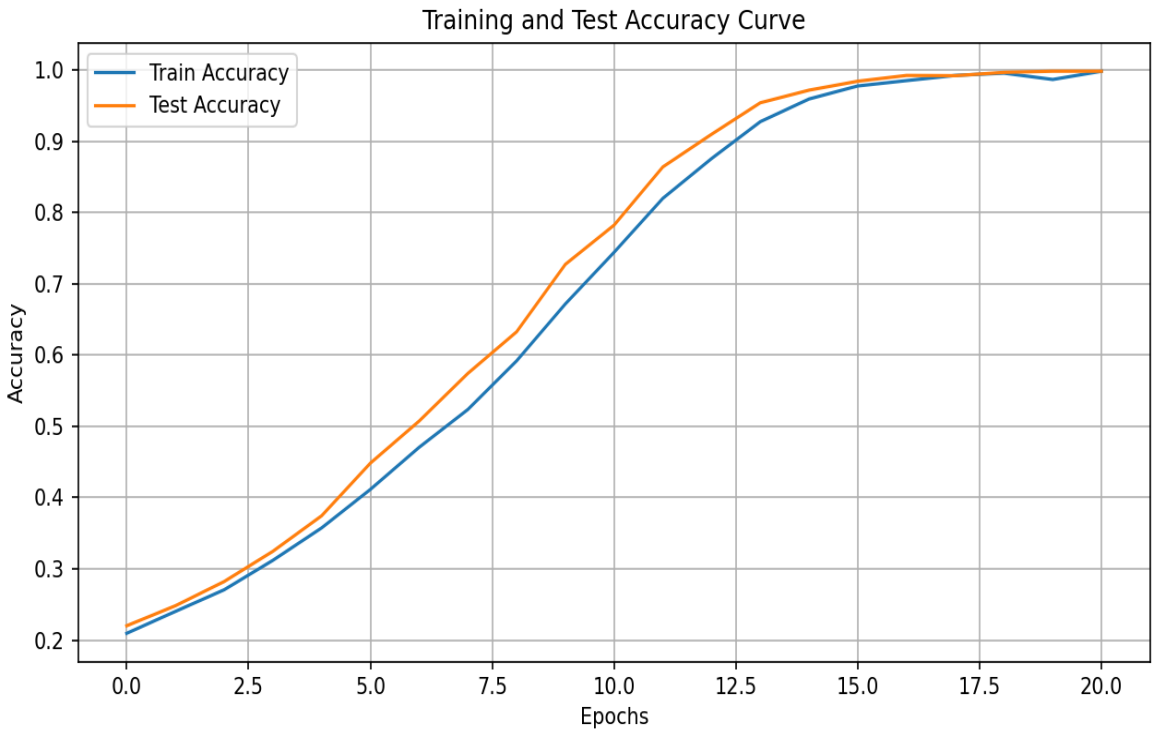

Рис. 26: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 6, L = 2$

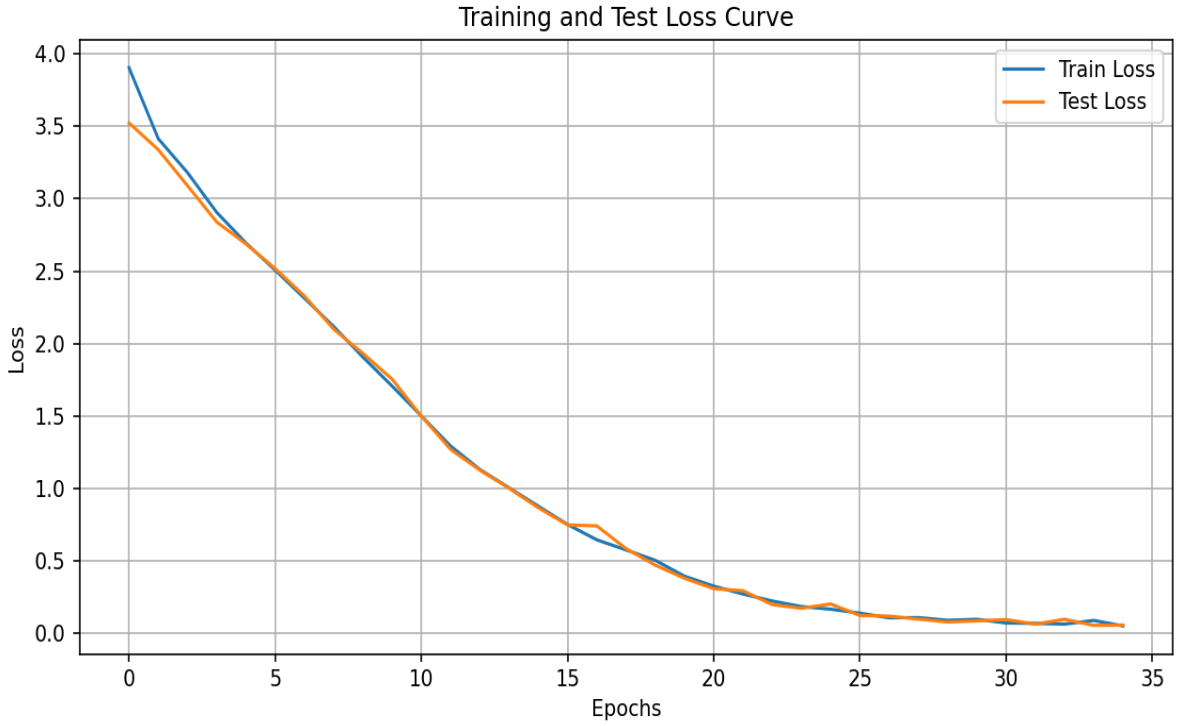

Рис. 27: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 7, L = 2$

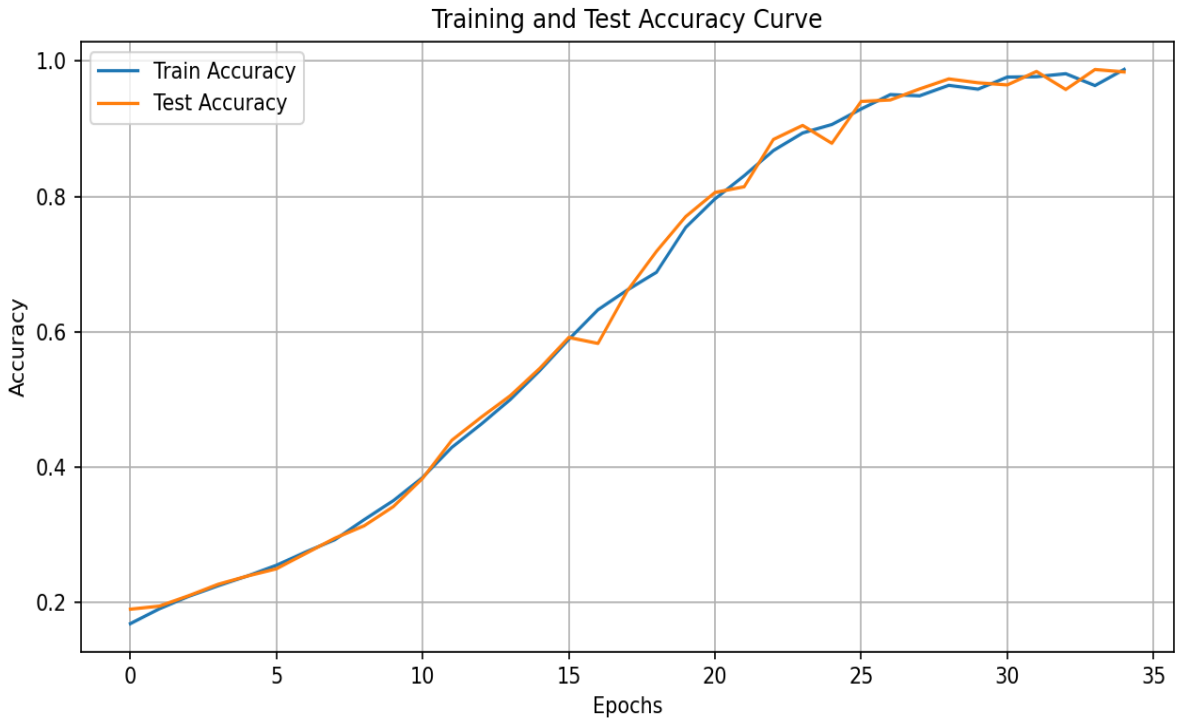

Рис. 28: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 7, L = 2$

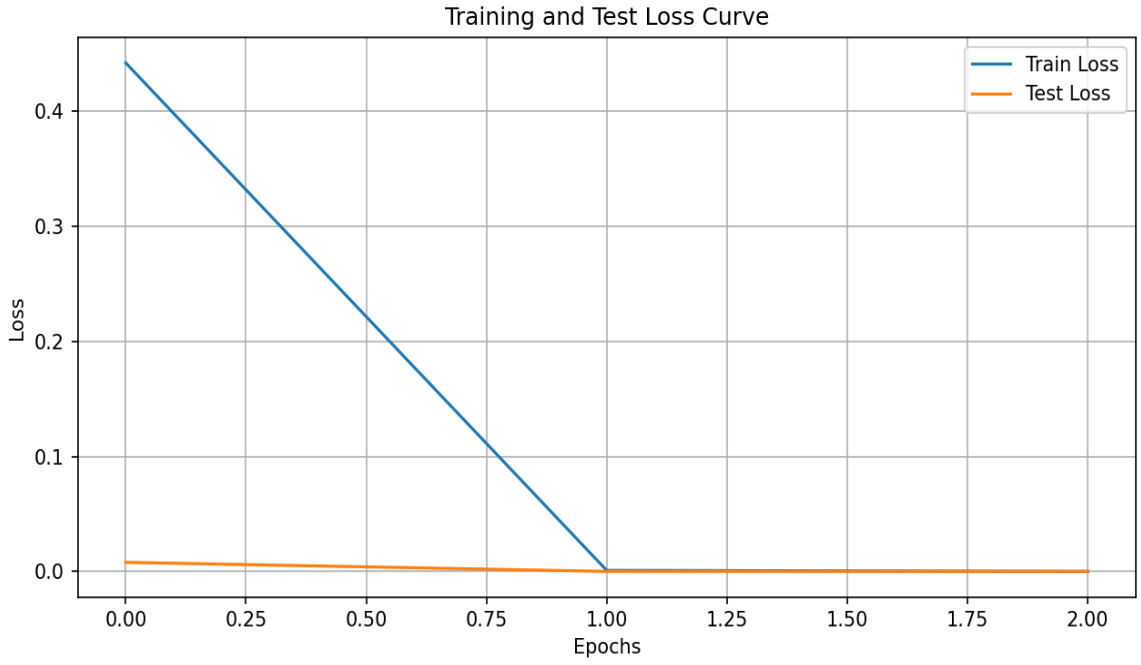

Рис. 29: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 3$

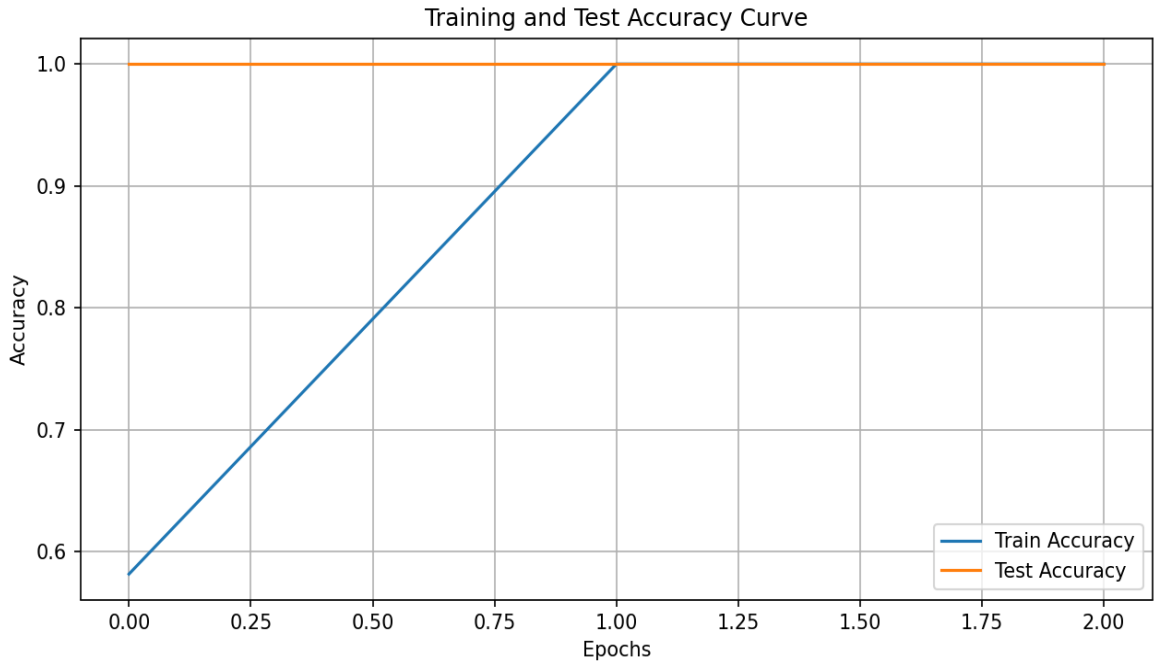

Рис. 30: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 3$

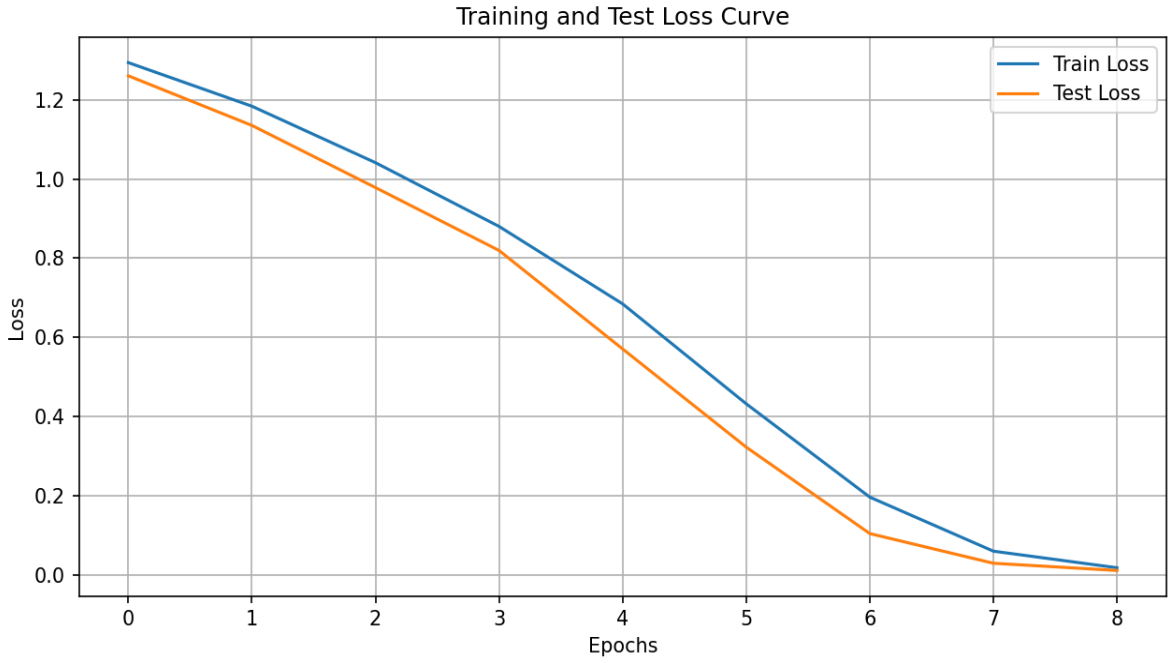

Рис. 31: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 3$

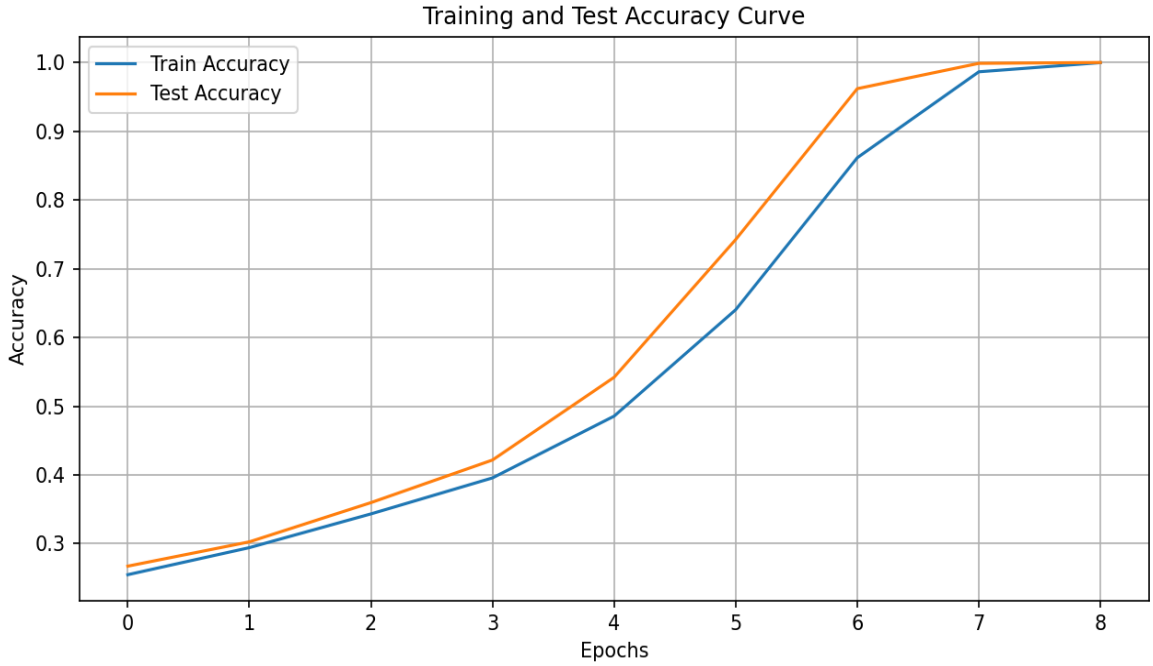

Рис. 32: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 3$

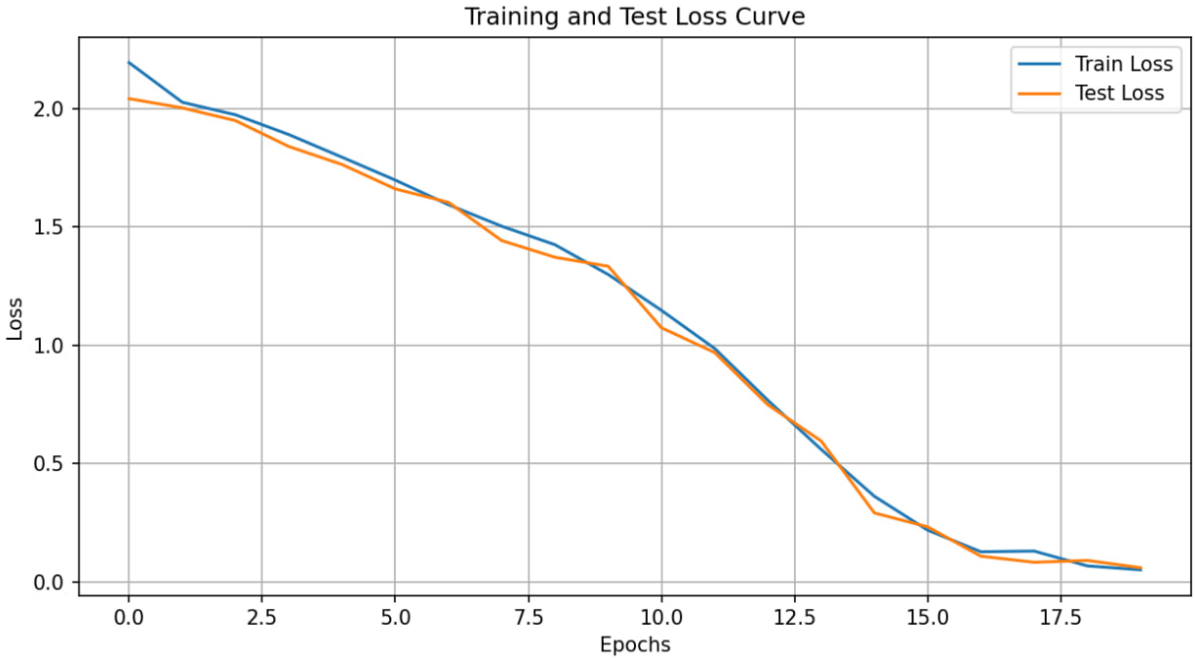

Рис. 33: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 3$

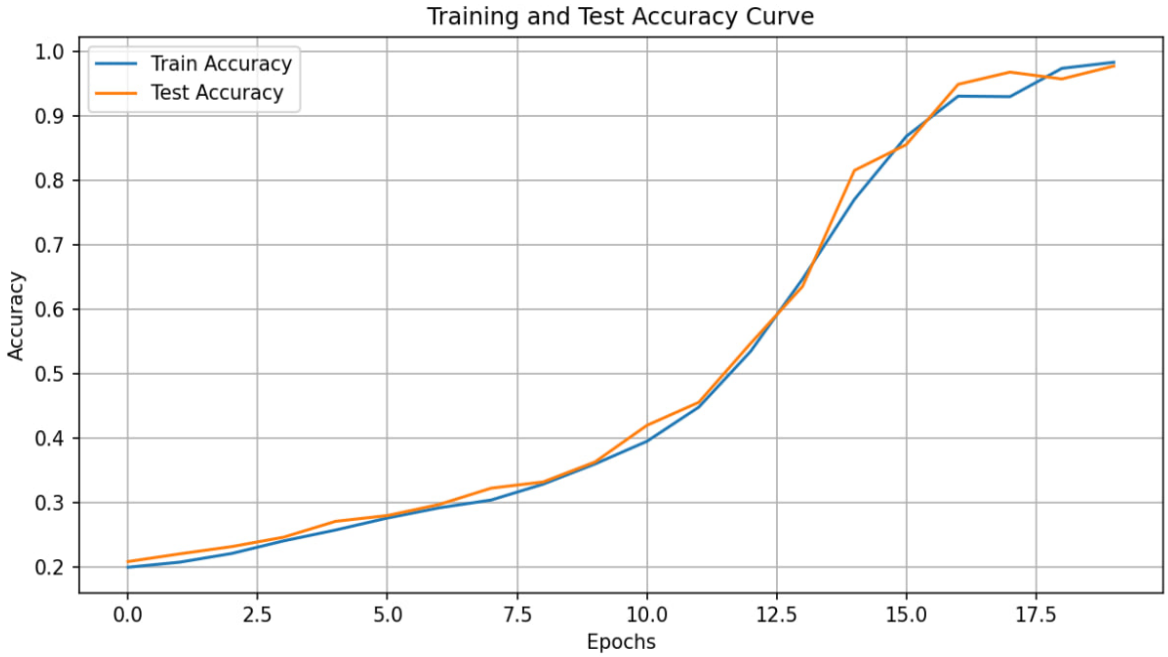

Рис. 34: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 5, L = 3$

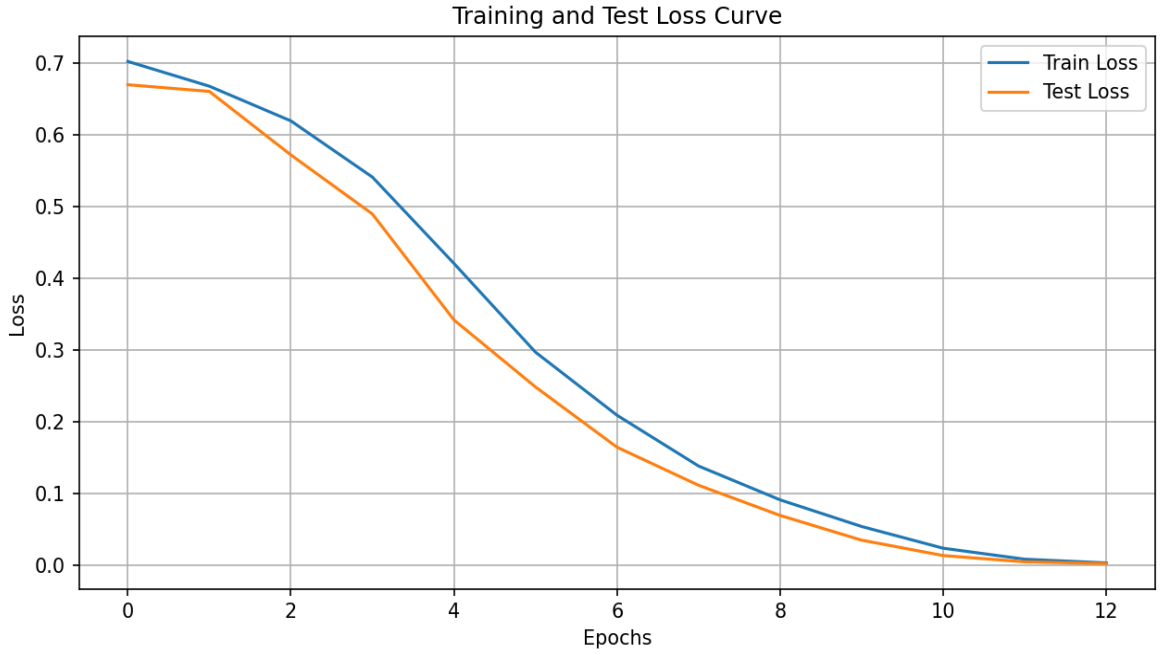

Рис. 35: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 4$

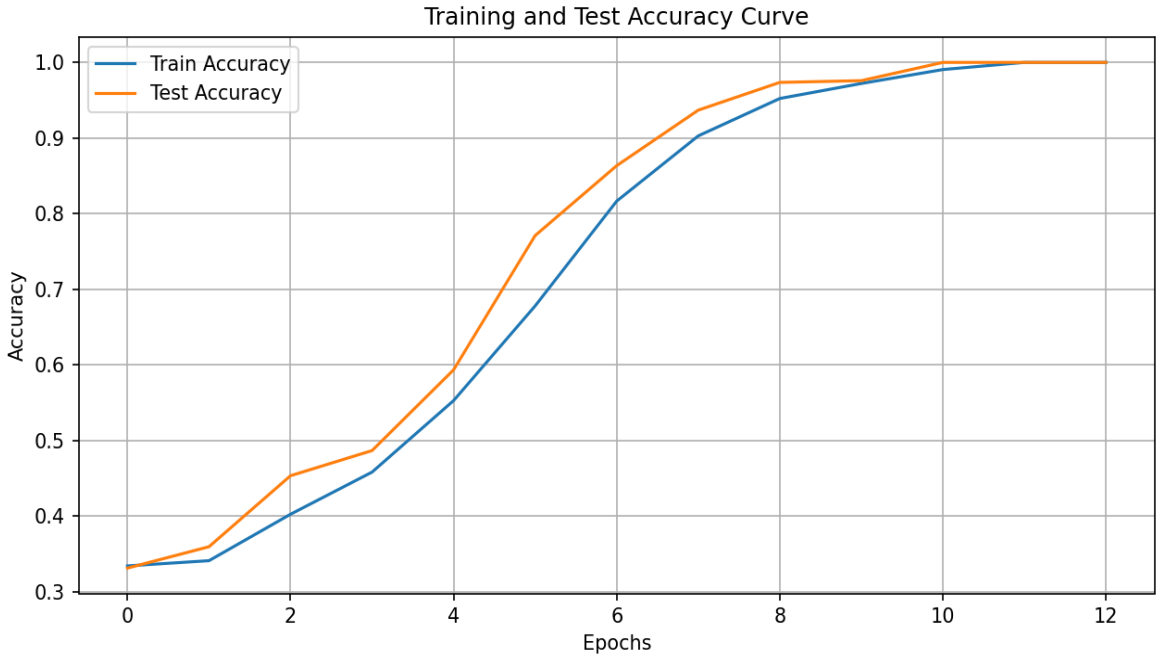

Рис. 36: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 4$

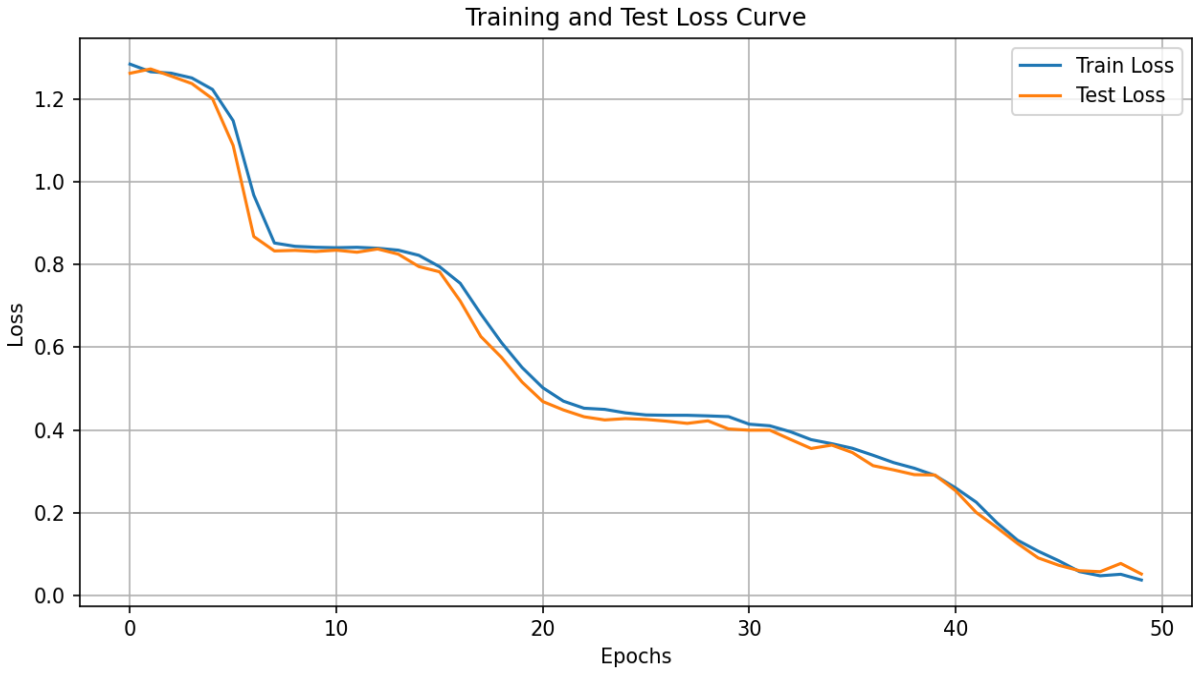

Рис. 37: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 4$

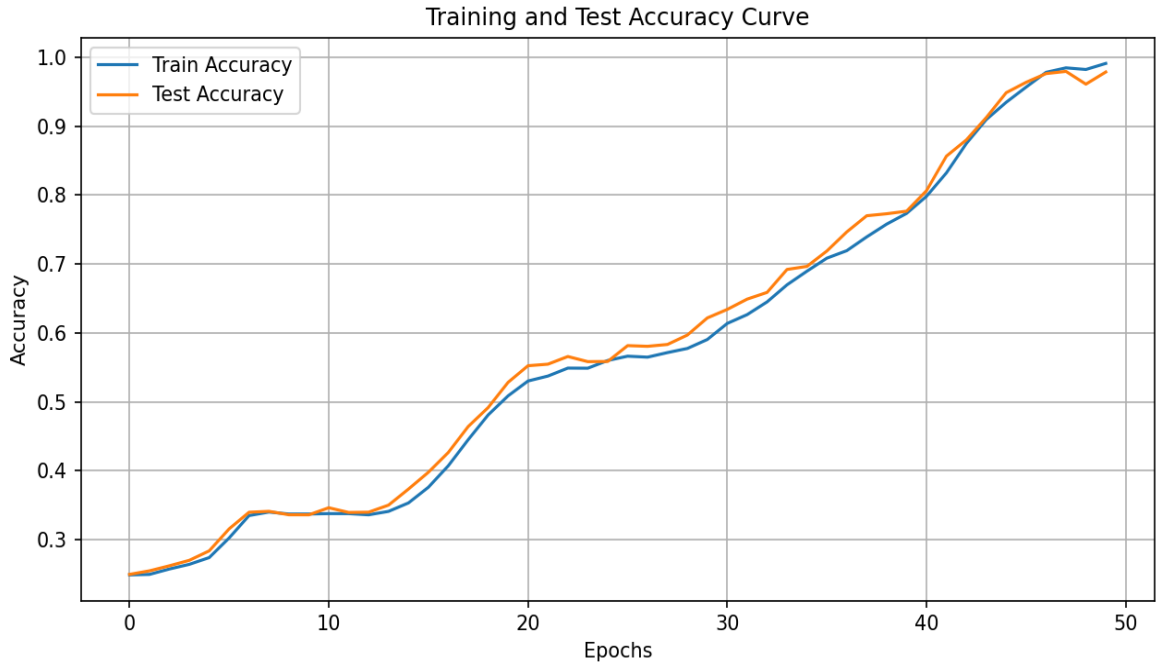

Рис. 38: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 4, L = 4$

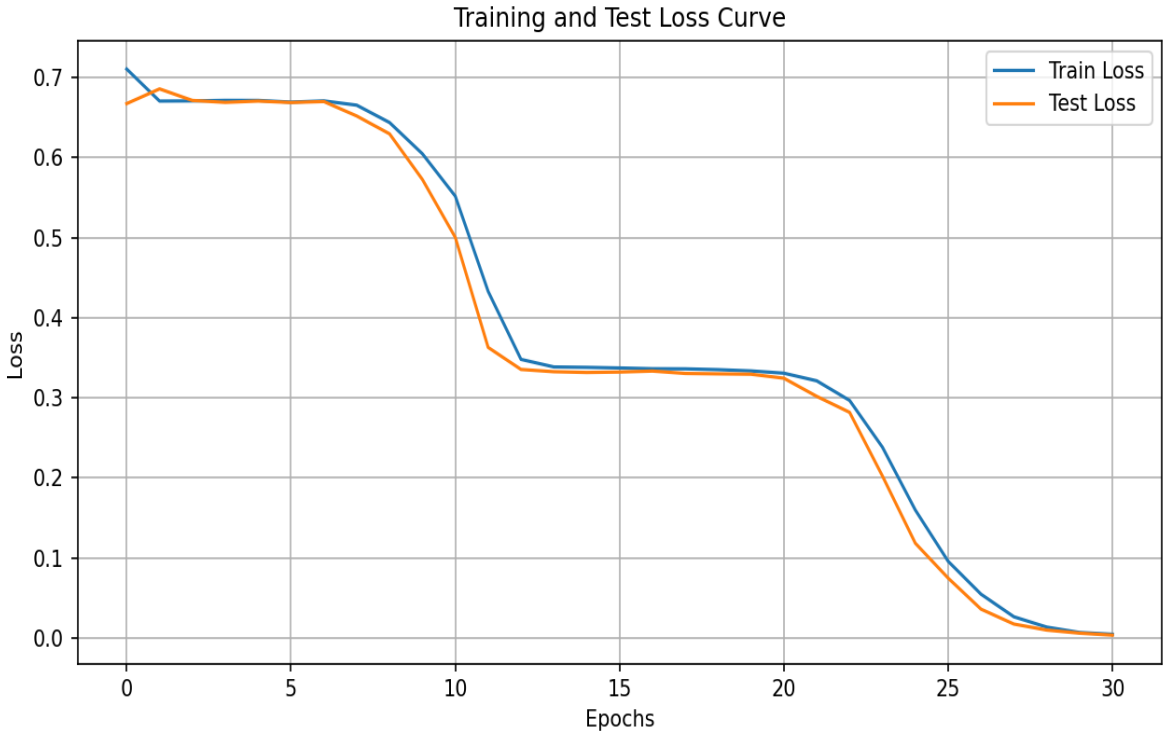

Рис. 39: График ошибки на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 5$

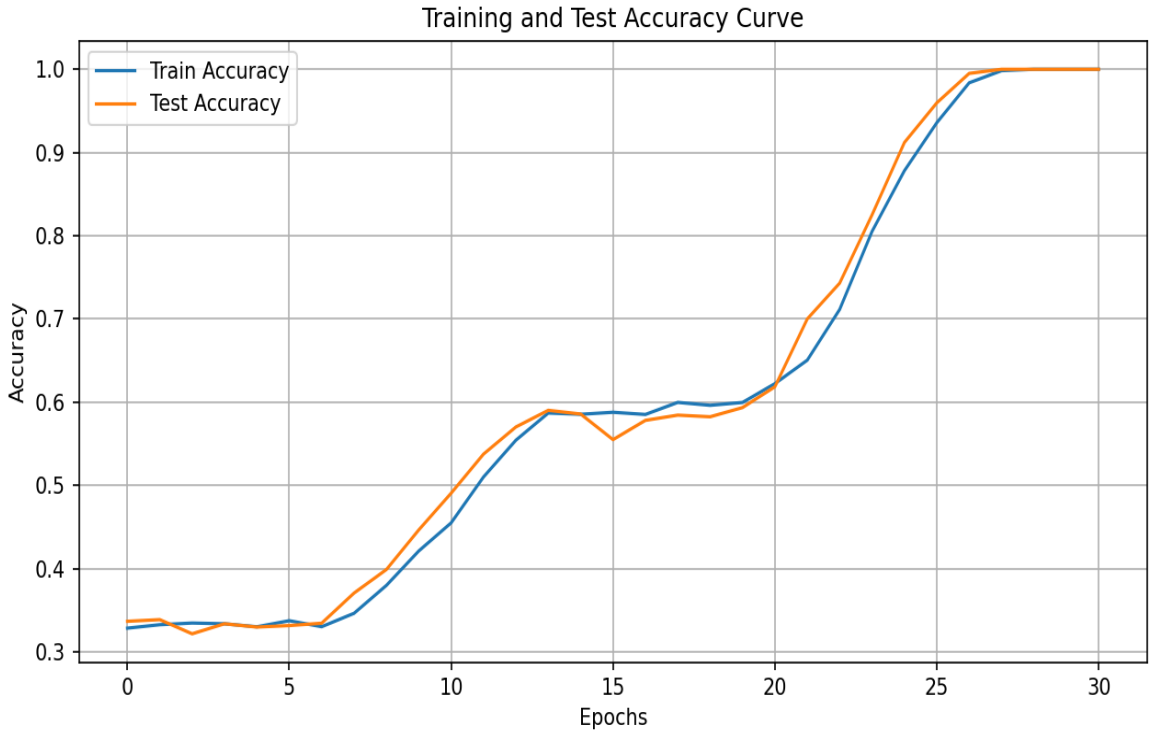

Рис. 40: График точности на обучающей и тестовой выборке в зависимости от эпохи для $n = 3, L = 5$

B. Программная реализация

```
for j in range(15000):
    ax1.axis('off')
    x = np.arange(n, m, h)
    y = np.zeros((int((l - k) / h), len(x)))

    y[:, 0] = np.arange(-k, -l, -h)

    for i in range(1, len(x)):
        y[:, i] = random.sample(range(-l + 1, 0), l - 1)

    y_new = np.zeros((len(x), int((l - k) / h)))
    for i in range(len(x)):
        y_new[i] = y[:, i]
    ax1.plot( *args: x, y_new, linewidth = 7, marker='o', markersize = 16, color='black')
    fig.tight_layout(pad=0)
    fig.savefig( fname: f'Dataset_n4_l4/{j}_{abs(y[:, len(x)-1])}.png', dpi=16)
    ax1.clear()
```

Рис. 41: Генерация датасета для неглубокой нейронной сети

```python
class TestDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.transform = transform

        self.image_paths = []
        self.labels = []

        for img_name in os.listdir(data_dir):
            self.image_paths.append(os.path.join(data_dir, img_name))
            l_bracket = self.image_paths[-1].find('[')
            r_bracket = self.image_paths[-1].find(']')
            label_str = self.image_paths[-1][l_bracket + 1: r_bracket]
            label_str = label_str.replace( __old: '.',  __new: '')
            targets = list(map(int, label_str.split()))
            targets = [x - 1 for x in targets]
            targets = torch.tensor(targets, dtype=torch.int64)
            # targets = F.one_hot(targets, 3)
            targets = targets.float()
            self.labels.append(targets)
```

Рис. 42: Преобразование изображения в вектор перестановки

```python
class ImageFeatureExtractor(nn.Module):
    def __init__(self, input_channels=1, image_size=image_size, n=n):
        self.n = n
        self.image_size = image_size
        super(ImageFeatureExtractor, self).__init__()
        self.convs = nn.Sequential(
            nn.Conv2d( in_channels: 1,  out_channels: 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d( in_channels: 32,  out_channels: 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc1 = nn.Linear(64*16*16,  out_features: 10000)
        self.fc2 = nn.Linear( in_features: 10000,  out_features: 800)
        self.fc3 = nn.Linear( in_features: 800, self.n)
    ...
    def forward(self, x):
        x = self.convs(x)
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = self.softmax_n(x, self.n)
        return x
```

Рис. 43: Архитектура неглубокой нейронной сети

```python
for epoch in range(epochs):
    model.train()
    cur_train_loss = 0
    cur_test_loss = 0
    acc_train = 0
    acc_test = 0
    for batch_idx, (data, targets) in enumerate(train_loader):
        data, targets = data.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(data)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        cur_train_loss += loss.item()
        predicted_vectors, pred_ind = torch.sort(outputs, descending=True)
        true_vectors, true_ind = torch.sort(targets, descending=True)
        accuracy = (pred_ind == true_ind).float().mean()
        acc_train += accuracy.item()
    avg_train_loss = cur_train_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    accuracy_train = acc_train / len(train_loader)
    train_acc.append(accuracy_train)
```

Рис. 44: Обучение неглубокой нейронной сети

```python
def generate_permutation_dataset(perms, n, L):
    perms = [torch.tensor(p) for p in perms]
    dataset = []
    for _ in range(20000):
        inputs = torch.stack([perms[np.random.randint(len(perms))] for _ in range(L)])
        targets = inputs[0]
        for i in range(1, L):
            targets = inputs[i][targets]
        dataset.append((inputs, targets))
    return dataset
```

Рис. 45: Генерация датасета для глубокой нейросети

```python
class MatrixMultNetwork(nn.Module):
    def __init__(self, n):
        self.n = n
        self.L = L
        super(MatrixMultNetwork, self).__init__()
        self.fc1 = nn.Linear(2 * self.n, 3*self.n**2)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(3*self.n**2, self.n)

    def forward(self, x):
        x = x.view(-1, 2*self.n)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.softmax(x)
        return x
```

Рис. 46: Архитектура глубокой нейросети при $L = 2$

```python
class DeepMatrixNetwork(nn.Module):
    def __init__(self, n, L):
        super(DeepMatrixNetwork, self).__init__()
        self.n = n
        self.L = L
        self.fc = nn.ModuleList()
        for i in range(self.L-1):
            self.fc.append(nn.Linear(2*self.n, 2*n**4))
            self.fc.append(nn.Linear(2*self.n**4, self.n))

    def forward(self, x):
        X, Y = [], []
        X.append(x[:, :2, :].view(-1, 2*self.n))
        Y.append(torch.relu(self.fc[0](X[0])))
        Y[0] = self.fc[1](Y[0])
        if self.L == 2:
            return Y[0]
        for i in range(1, self.L-1):
            X.append(torch.cat( tensors: (x[:, i + 1, :], Y[- 1]), dim=1).view( *shape: -1, 2 * self.n))
            Y.append(torch.relu(self.fc[2 * i](X[-1])))
            Y[i] = self.fc[2 * i + 1](Y[-1])
        return Y[-1]
```

Рис. 47: Архитектура глубокой нейросети при $L \geq 2$