МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
Саровский физико-технический институт -
филиал федерального государственного автономного образовательного учреждения
высшего образования «Национальный исследовательский ядерный университет
«МИФИ»
(СарФТИ НИЯУ МИФИ)

ФИЗИКО-ТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КВАЛИФИКАЦИОННОЙ РАБОТЕ

На тему:

Реализация и адаптация нейронной сети распознавания рукописных букв для исполнения на устройствах различного класса производительности

СТУДЕНТ Никитин Егор Андреевич
ФИО подпись, число

НАУЧНЫЙ РУКОВОДИТЕЛЬ старший научный сотрудник,
Дмитриев Николай Александрович
должность, уч.степень, ФИО, подпись, число

РЕЦЕНЗЕНТ начальник научно-исследовательского отдела ИТМФ, к.ф. – м.н.,
Бабанов Алексей Викторович
должность, уч.степень, ФИО, подпись, число

ЗАВ. КАФЕДРОЙ член-корр. РАН,
профессор, д.ф. – м.н. Шагалиев Рашит Мирзагалиевич
должность, уч.степень, ФИО, подпись, число

Саров, 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»

Саровский физико-технический институт -

филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»

(СарФТИ НИЯУ МИФИ)

ФИЗИКО-ТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ

УТВЕРЖДАЮ
Заведующий кафедрой
прикладной математики
_____Р.М.Шагалиев

«_____» _____ 2024 г.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент Никитин Егор Андреевич
фамилия, имя, отчество

Направление 01.03.02 «Прикладная математика и информатика»
наименование направления

Профиль подготовки Высокопроизводительные вычисления и технологии параллельного программирования
наименование профиля

Научный руководитель старший научный сотрудник, Дмитриев Николай Александрович
должность, уч. степень и звание, фамилия, имя, отчество

Саров, 2024

Наименование темы квалификационной работы

Реализация и адаптация нейронной сети распознавания рукописных букв для исполнения на устройствах различного класса производительности

Место выполнения математическое отделение РФЯЦ-ВНИИЭФ

Исходные данные литература, указанная в работе

Содержание

1. Основные сведения о нейронных сетях.
2. Создание и оптимизация нейронной сети для распознавания рукописных букв.
3. Квантизация модели и оценка производительности на разных устройствах.

Допуск к защите

К защите представляется:
пояснительная записка _____ страниц
иллюстрационный материал _____ слайдов

Научный руководитель квалификационной работы
Старший научный сотрудник

_____ Дмитриев Н.А.
(подпись)

Рецензент квалификационной работы
Начальник научно-исследовательского отдела ИТМФ, к.ф.- м.н. _____ Бабанов А.В.
(подпись)

Студент гр. ПМ-40, Никитин Егор Андреевич допущен к защите квалификационной
(индекс группы, фамилия, имя, отчество)
работы по направлению 01.03.02 «Прикладная математика и информатика»

Дата защиты 20.06.2024 г.

Заведующий кафедрой прикладной математики
член-корр. РАН, профессор, д.ф.- м.н. _____ Шагалиев Р.М.
(подпись)

Аннотация

Работа посвящена исследованию в области создания и обучения нейронных сетей для распознавания (классификации) рукописных букв, а также запуска на исполнение (инференс) таких сетей на устройствах с различной архитектурой (и производительностью) центрального процессора.

В первой главе рассматриваются основные принципы работы нейронных сетей, алгоритм обучения (градиентный спуск, обратное распространение ошибки), функции активации и функции стоимости.

Вторая глава посвящена практической реализации нейронной сети, подготовке набора данных (датасета), подбору оптимальных гиперпараметров, выбору функций активации и методам борьбы с замедлением обучения.

В третьей главе представлены результаты тестирования обученной нейронной сети на различных устройствах, а также описан процесс квантизации модели для ускорения инференса и адаптации к устройствам, обладающим слабыми вычислительными ресурсами.

Данная дипломная работа состоит из введения, трех глав, заключения, списка литературы и включает в себя 35 рисунков и 6 таблиц.

Содержание

Введение

На фоне активной цифровизации различных сфер человеческой жизнедеятельности и бурно развивающихся технологий искусственного интеллекта, все большую актуальность приобретают техники анализа изображений. В частности, задача распознавания объектов на изображении востребована и находит прикладное применение в самых разнообразных областях.

В последние годы глубокие нейронные сети стали доминирующим подходом к решению задач распознавания образов, к которым относится и задача распознавания текста. Они демонстрируют высокую точность и способность к обобщению знаний, что делает их привлекательными для практического применения.

Распознавание рукописного текста является одной из важных задач в области искусственного интеллекта в целом, и в задачах распознавания в частности. Она имеет широкий спектр сценариев применения: от оцифровки архивов и автоматизации ввода данных до разработки интеллектуальных интерфейсов и систем помощи людям с ограниченными возможностями.

Эффективность работы нейронных сетей напрямую зависит от множества факторов: качества подготовки данных для обучения, архитектуры сети, выбора функций активации нейронов и функций стоимости, методов инициализации весов и других факторов. Понимание влияния каждого из этих компонентов на процесс обучения и итоговый результат критически важно для создания высокоэффективных моделей.

С развитием аппаратного обеспечения, и, в особенности, портативной электроники, особый интерес исследователей вызывает задача оптимизации моделей нейронных сетей для их инференса на устройствах с низкой производительностью. Основная сложность здесь заключается в соблюдении баланса между сложностью вычислений и точностью получаемого результата.

В настоящее время, при разработке технологий искусственного интеллекта становятся актуальными и вопросы информационной безопасности. Использование готовых моделей нейронных сетей и зарубежных фреймворков создает потенциальные риски утечек данных и несет угрозы инфраструктуре, на которой они запускаются.

Цель и задачи

Цель

Исследование применения нейронных сетей для задачи распознавания рукописных букв русского алфавита, исполнения на устройствах различного класса производительности, а также исследование влияния различных компонентов нейронной сети на процесс обучения и итоговый результат.

Задачи:

- Изучение принципов работы нейронных сетей, получение компетенций и навыков в области создания и обучения нейронных сетей.
- Подготовка наборов данных для обучения и инференса нейронной сети.
- Исследование и подбор оптимальных гиперпараметров нейронной сети (размер входного пакета, количество нейронов, количество слоев сети), функций активации нейронов и функций стоимости (ошибки).
- Выбор наиболее эффективного способа инициализации весов нейронной сети и техники борьбы с переобучением.
- Достижение точности распознавания рукописных букв русского алфавита не менее 90%.
- Понижение вычислительной сложности нейронной сети путем квантизации ее параметров.

1 Теоретическая часть

1.1 Определение нейронной сети. Виды нейронных сетей

Нейронная сеть – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма. В нейронных сетях обычно используются три базовых типа слоев – входной, выходной и скрытый. Скрытым слоем называют любой промежуточный слой, находящийся между входным и выходным слоями. В нейронной сети может содержаться несколько скрытых слоев, в таком случае сеть называют глубокой. Каждый слой содержит нейроны, которые обрабатывают информацию и передают ее дальше.

Одними из первых моделей нейросетей являются персептрон и многослойный персептрон. Персептрон – простейший вид нейронной сети, в основе которого лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов. Рассмотрим модель нейронной сети известную как многослойный персептрон:

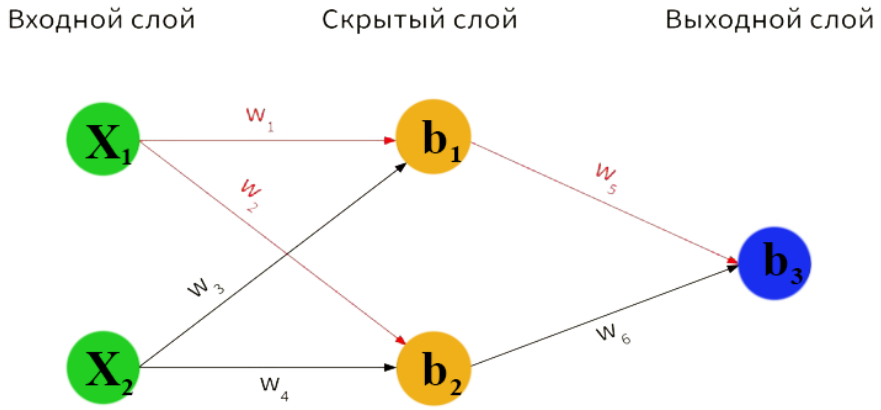


Рисунок 1 – многослойный персептрон, где X – входные данные, w – веса, b – нейроны скрытого и выходного слоя, включая смещения

Для многослойного персептрона компонентами, непосредственно влияющими на обучение, являются:

Веса (weights) – вещественные числа, выражающие важность соответствующих входных чисел для результатов, т.е. они представляют силу связи между нейронами.

Смещения (bias) – это константы, которые добавляются к выходу каждого нейрона. Они позволяют контролировать общий уровень активации нейронов. Чем больше смещение, тем проще активироваться нейрону.

В общем случае, веса и смещения являются параметрами, которые определяют, как нейронная сеть будет реагировать на входные данные.

Для того чтобы нейронная сеть могла точно выполнять поставленную задачу, её нужно обучить. Обучение нейронной сети – это процесс оптимизации значений весов и смещений. В общем случае, нейросети обучаются одним из следующих способов:

- С учителем – выходное пространство решений нейронной сети известно.
- Без учителя – нейронная сеть формирует выходное пространство решений только на основе входных воздействий. Такие сети называют самоорганизующимися.
- С подкреплением – система назначения штрафов и поощрений от среды.

В данной работе будет использовано обучение с учителем. Обучение нейронных сетей с учителем происходит путем подачи на вход выборки из обучающих данных, получения на выходе результата, сравнения результата с эталонным значением и корректировки весов связей между нейронами на каждом слое. Это позволяет сети улучшать свою точность в решении задач.

Существует множество типов нейронных сетей, каждый из которых используется для решения ряда задач. Некоторые из наиболее распространенных типов нейронных сетей включают в себя:

- **Прямое распространение (Feed Forward)** – это наиболее простой тип нейронной сети, который состоит из одного или нескольких скрытых слоев и выходного слоя. Он преимущественно используется для задач классификации и регрессии. Пример нейросетей прямого распространения: персептрон, Radial Basis Network, Deep Feed Forward.
- **Рекуррентные нейронные сети (RNN)** – это тип нейронной сети, который преимущественно используется для обработки последовательностей данных, таких как тексты, звуковые записи и временные ряды. Они могут запоминать предыдущие состояния и использовать их для принятия решений. Пример

RNN: Recurrent Neural Network, Long/Short Term Memory, Gated Recurrent Unit, Echo State Network.

- **Трансформеры (Transformer)** – это тип нейронной сети, предложенный в 2017 году, который использует механизм внимания (attention) вместо рекуррентных связей, характерных для традиционных рекуррентных нейронных сетей (RNN). Механизм внимания позволяет модели фокусироваться на наиболее важных частях входной последовательности при обработке каждого элемента. Одни из применений трансформеров это обработка текста и машинный перевод. Пример: GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), Transformer-XL.
- **Сверточные нейронные сети (CNN)** – это тип нейронной сети, который преимущественно используется для анализа изображений и видео. Они содержат сверточные слои, которые могут обнаруживать различные признаки объектов изображения. Пример: Deep Convolutional Network, Deconvolutional Network, Deep Convolutional Inverse Graphics Network.
- **Автоэнкодеры (Auto Encoder)** – это тип нейронной сети, который используется для изучения скрытых признаков в данных и их восстановления. Они могут использоваться для уменьшения размерности данных и генерации новых данных. Пример автоэнкодеров: Variational AE, Denoising AE, Sparse AE.

Каждый тип нейронной сети имеет свои преимущества и недостатки, и выбор конкретного типа зависит от задачи, которую необходимо решить.

Нейроны в слоях нейронных сетей можно классифицировать следующим образом:

- **Входные слои** – принимают на вход данные и передают их дальше по сети. Они не выполняют никаких сложных операций. Пример входных нейронов: Input Cell, Backfed Input Cell, Noisy Input Cell.
- **Скрытые слои** – выполняют более сложные операции с данными, чем входные слои. Они обрабатывают информацию, полученную от входного слоя, и передают ее дальше по сети. Каждый нейрон скрытого слоя

получает входные данные от всех нейронов предыдущего слоя и вычисляет свой выход на основе заданной функции активации. Задача скрытых слоев заключается в извлечении признаков из данных, которые могут быть использованы для решения задачи. Пример скрытых нейронов: Hidden Cell, Probabilistic Hidden Cell, Spiking Hidden Cell, Capsule Cell. Также можно выделить два важных подтипа скрытых слоев:

- **Рекуррентные слои** – позволяют моделировать последовательности данных, такие как тексты, временные ряды или звуковые сигналы. Они работают путем сохранения информации о предыдущих состояниях сети и использования этой информации для принятия решений на следующем шаге. Это позволяет рекуррентным слоям учитывать контекст и зависимости в данных, что часто является важным фактором для успешного решения задач, связанных с последовательностями. Пример рекуррентных нейронов: Recurrent Cell, Memory Cell, Gated Memory Cell.
- **Сверточные слои** – используются для обработки изображений и других типов данных, которые имеют пространственную структуру. Они работают путем сканирования входных данных с помощью набора фильтров (ядер свёртки), которые выделяют различные признаки изображения. Каждый фильтр проходит через входные данные, создавая карту признаков, которая содержит информацию о том, где на изображении находятся определенные объекты или структуры. После этого происходит операция субдискретизации (пулинг), которая уменьшает размерность карты признаков и улучшает инвариантность к масштабу и переносу объектов на изображении. Пример сверточных нейронов: Convolution, Pooling.

- **Выходные слои** – принимают выходные данные от последнего скрытого слоя и преобразуют их в нужный формат для решения конкретной задачи. К примеру, на выходном слое может использоваться распределение вероятностей, что позволяет получать не только предсказанный класс, но

и степень уверенности модели в своем прогнозе. Пример выходных нейронов: Output Cell, Match Input Output Cell.

1.1.1 Краткая историческая справка

История появления нейронных сетей начинается в 1943 году, когда Уоррен Маккаллок и Уолтер Питтс опубликовали статью "Логический калькулятор, использующий нервные элементы". В этой статье они предложили модель искусственного нейрона, который может получать входные сигналы, обрабатывать их и выдавать выходной сигнал.

В 1950-х годах Фрэнк Розенблатт разработал персептрон – нейронную сеть, которая может обучаться распознаванию образов. Однако персептрон имел ограничения и не мог решать сложные задачи.

В 1960-х годах Марвин Минский и Сеймур Пейперт показали, что однослойный персептрон не может решить простейшую задачу распознавания образов, которая называется проблемой XOR. Это привело к затуханию интереса к нейронным сетям на несколько десятилетий.

В 1980-х годах нейронные сети получили новый импульс развития благодаря работам Джеффри Хинтона, Яна Лекуна и Дэвида Румеляхи. Они разработали многослойные нейронные сети, которые могут обучаться решать сложные задачи, такие как распознавание речи и образов, классификация изображений и прогнозирование временных рядов.

В последние годы наблюдается значительный прогресс в области нейронных сетей, которые изменили нашу жизнь и открыли новые возможности в различных областях, к примеру:

1. Обработка естественного языка – нейросети стали значительно лучше в понимании естественного языка. Это привело к созданию более эффективных систем машинного перевода, анализа текста и генерации текста.
2. Обработка изображений – сверточные нейронные сети стали более точными и эффективными в распознавании объектов на изображениях. Это привело к созданию более точных систем автоматического распознавания лиц, автомобилей и других объектов.

3. Автономная навигация – нейросети стали использоваться для разработки автономных систем навигации, таких как беспилотные автомобили и дроны. Это позволяет им работать более эффективно и безопасно в различных условиях.
4. Медицинская диагностика – нейросети стали использоваться в медицинской диагностике для обнаружения заболеваний и прогнозирования их развития. Это позволяет более точно диагностировать и лечить различные заболевания.
5. Игры – нейросети стали использоваться для создания более интеллектуальных компьютерных игр. Например, AlphaGo – нейросеть, разработанная компанией Google, победила чемпиона мира по игре «Го», что было неожиданным результатом в области искусственного интеллекта.

Эти прорывы являются лишь некоторыми из многих достижений, которые были сделаны в области нейросетей в последние годы. Они открывают новые возможности для применения нейросетей в различных областях.

1.2 Принцип работы нейронной сети

Предположим, у нас есть искусственная нейронная сеть, которую мы хотим обучить решению задачи классификации рукописных букв. Входными данными сети являются пиксели отсканированного изображения рукописной буквы. Требуется определить такие веса и смещения, чтобы классификация букв была максимально точной. Для понимания процесса обучения необходимо рассмотреть влияние небольших изменений весов и смещений сети на ее выходные данные. Желательно, чтобы эти изменения были пропорциональны, что делает обучение возможным. Для этого используют функции активации нейрона. Функция активации – это функция, которая на основании входного сигнала определяет выходной. На начальном этапе будем использовать нейроны с сигмоидой в качестве функции активации (сигмоидный нейрон), для которых небольшие изменения весов и смещений приводят только к небольшим изменениям выходных данных. В дальнейшем будет рассмотрена другая функция активации.

Сигмоидный нейрон имеет входные данные $x_1, x_2, \ldots$, веса для каждого входа, $w_1, w_2, \ldots$ и общее смещение b. Его выходным значением будет $\sigma(w \cdot x + b)$, где σ – это сигмоида. Определяется функция следующим образом:

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}, \tag{1.1}$$

где $z = w \cdot x + b$. Сигмоидная функция имеет вид:

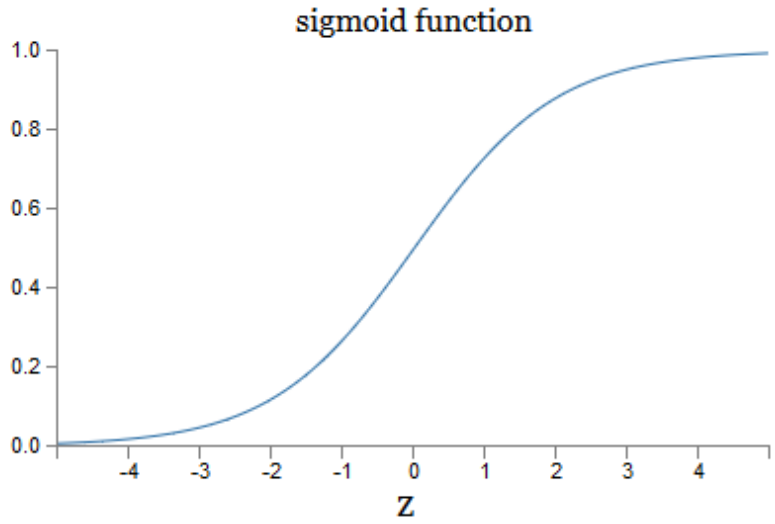


Рисунок 2 – график сигмоиды

В нашем случае выходное значение сигмоидного нейрона с входными данными $x_1, x_2, \ldots$ весами $w_1, w_2, \ldots$ и смещением b будет считаться, как:

$$\frac{1}{1+exp(-\sum_j w_j x_j - b)}. \tag{1.2}$$

Для преобразования выходных значений нейронов в вероятности принадлежности к каждому классу используется функция softmax. Функция softmax – это функция, применяемая на выходном слое многоклассовых классификаторов. Её формула имеет вид:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}, \tag{1.3}$$

где z_i – значение i-го нейрона выходного слоя. Softmax преобразует вектор чисел в вектор вероятностей, сумма которых равна 1. Поэтому значение функции softmax для каждого класса можно интерпретировать как вероятность принадлежности входного объекта к этому классу.

Для того, чтобы выход нейронной сети был относительно предсказуем и приближался к с своему точному значению $y(x)$ для всех обучающих значений x, необходимо подобрать некий алгоритм, позволяющий искать оптимальные значения соответствующих весов и смещений. Для количественной оценки приближения к этой цели определяют некую функцию стоимости, например, квадратичную:

$$C(w,b) = \frac{1}{2n}\sum_{x} || \, y(x) - a \, ||^2, \tag{1.4}$$

где w – набор весов сети, b – набор смещений, n – количество обучающих входных данных, a – вектор выходных данных сети, когда x – входные данные, сумма проходит по всем обучающим входным данным x.

Функция стоимости C является не отрицательной, поскольку все члены суммы неотрицательны. Кроме того, стоимость $C(w,b)$ становится малой тогда, когда $y(x)$ примерно равна выходному вектору a для всех обучающих входных данных x. Можно сделать вывод, что алгоритм сработал хорошо, если сумел найти веса и смещения такие, что $C(w,b) \approx 0$. Получается, цель обучающего алгоритма – минимизация $C(w,b)$ как функции весов и смещений. То есть, нужно найти набор весов и смещений, минимизирующий значение стоимости. Для этого можно воспользоваться алгоритмом под названием градиентный спуск.

1.3 Градиентный спуск

Функцией стоимости $C(v)$ может быть любая вещественная функция от произвольного количества переменных $v = v_1, v_2, \ldots$. В качестве примера, возьмем функцию стоимости от двух переменных $C(v_1, v_2)$, допустим, что её график имеет следующий вид:

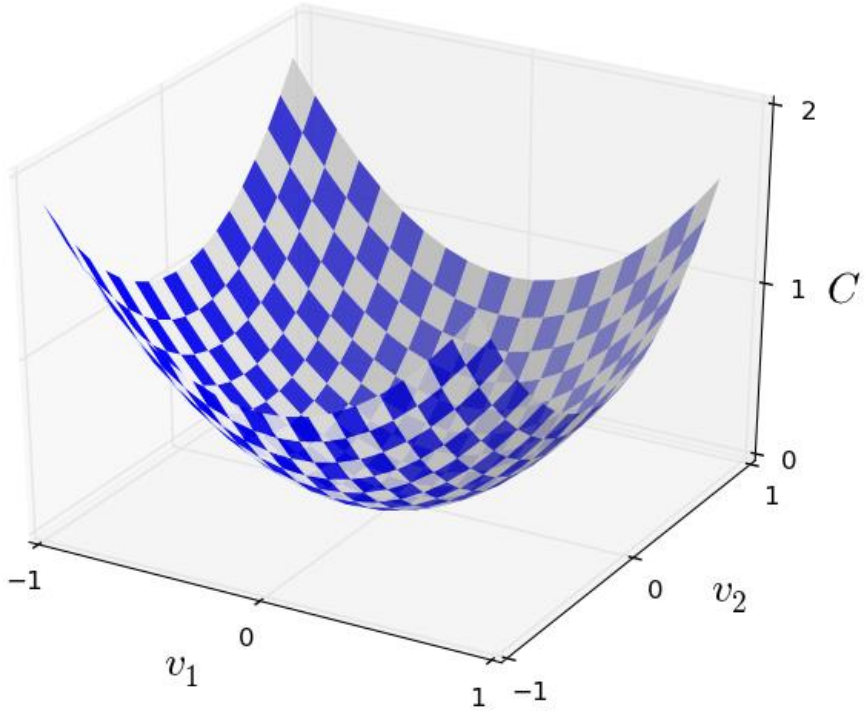


Рисунок 3 – график функции стоимости C

Необходимо найти, где функция стоимости C достигает глобального минимума. Один из способов решения задачи – использовать математический анализ для поиска минимума аналитическим путём. Можно вычислить производные и попытаться использовать их для поиска экстремума. Это сработает, когда C будет функцией от одной или двух переменных. Но при большом количестве переменных задача сильно усложняется. А для нейронных сетей нам необходимо гораздо больше переменных – у крупнейших сетей функции стоимости могут зависеть от миллиардов весов и смещений. Использование анализа для минимизации этих функций, является невозможной задачей.

Существует аналогия, на основе которой можно построить хорошо работающий алгоритм. Функцию стоимости можно представить в виде многомерной поверхности с минимумами и максимумами. Рассмотрим движение шара по поверхности, описываемой функцией стоимости (см. Рисунок 3). Согласно законам физики шар должен скатится на дно. Данную идею можно использовать для поиска глобального минимума функции стоимости. Случайным образом выбирается начальная точка для воображаемого шара, а потом симулируется движение шара, как будто он скатывается к нижней точке поверхности:

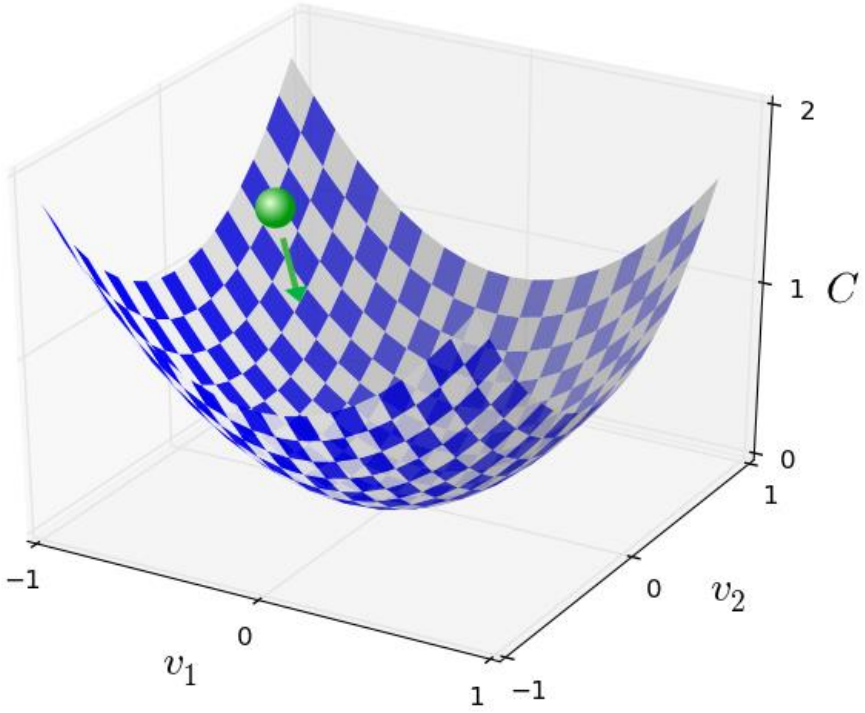


Рисунок 4 – Нахождение глобального минимума функции стоимости C

Далее покажем способ получения формул для алгоритма градиентного спуска.

В первую очередь необходимо задать начальное значение функции стоимости C. Изменение значения функции стоимости на небольшое значение Δv_1 в направлении v_1, и на небольшое значение Δv_2 в направлении v_2, можно записать:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 . \tag{1.5}$$

Найдем способ выбора таких Δv_1 и Δv_2, чтобы ΔC было меньше нуля; то есть, нужно выбирать их так, чтобы значение функции стоимости уменьшалось. Определим Δv как вектор изменений, то есть $\Delta v \equiv \left(\Delta v_1, \Delta v_2\right)^T$, где T – операция транспонирования. Также определим градиент C как вектор частных производных:

$$\nabla C \equiv (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2})^T . \tag{1.6}$$

С учетом (1.6) выражение (1.5) можно переписать, как:

$$\Delta C \approx \nabla C \cdot \Delta v . \tag{1.7}$$

Из этого уравнения видно, что ∇C связывает изменения в v с изменениями C. Выберем:

$$\Delta v = -\eta \nabla C , \tag{1.8}$$

где η – небольшой положительный параметр (будем называть его скоростью обучения).

Тогда по уравнению (1.7) можно понять, что $\Delta C \approx -\eta \nabla C , \cdot \nabla C = -\eta \| \nabla C \|^2$. Поскольку $\| \nabla C \|^2 \geq 0$, это гарантирует, что $\Delta C \leq 0$, то есть функция стоимости C будет всё время уменьшаться, если менять v, как прописано в (1.8). Уравнение (1.8) можно использовать для определения уменьшения функции стоимости в алгоритме градиентного спуска. Поэтому уменьшение функции стоимости на значение Δv можно записать:

$$v \to v' = v - \eta \nabla C . \tag{1.9}$$

Путем многократного применения градиентного спуска, будет понижаться значение функции стоимости C, пока не будет достигнут глобальный минимум.

Для корректной работы алгоритма градиентного спуска необходимо выбирать достаточно малое значение скорости обучения. В противном случае, есть вероятность, что значение функции стоимости будет расти $\Delta C > 0$. Однако, не следует выбирать η слишком маленькой, поскольку тогда изменения Δv будут ничтожно малыми, и алгоритм будет работать слишком медленно. На практике значение η подбирается эмпирически, обеспечивая баланс, при котором значение функции стоимости стремилось бы к нулю, и при этом алгоритм работал не слишком медленно. Для задач классификации градиентный спуск можно считать оптимальной стратегией поиска минимума функции стоимости.

Уравнения (1.7), (1.8), (1.9) также применимы если функция стоимости C будет зависеть от множества переменных $v_1, \ldots, v_m$.

Применяя градиентный спуск к обучению нейронных сетей, с учетом уравнения (1.9), правило обновления весов w и смещений b нейросети можно записать:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}, \tag{1.10}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{1.11}$$

Повторно применяя это правило обновления, значение функции стоимости будет постепенно приближаться к глобальному минимуму, и в конечном итоге с высокой вероятностью достигнет его.

1.4 Проблема градиентного спуска

Из вида уравнения (1.4) можно, заметить, что функция стоимости выглядит, как $C = 1/n \sum_x C_x$, то есть это среднее по стоимости $C_x \equiv \left\| y(x) - a \right\|^2 / 2$ для отдельных обучающих примеров. На практике для вычисления градиента ∇C нам нужно вычислять градиенты ∇C_x отдельно для каждого обучающего входа x, а потом усреднять их, $\nabla C = 1/n \sum_x \nabla C_x$. Минус состоит в том, что, когда количество входных данных будет очень большим, это займёт очень много времени, и такое обучение будет проходить медленно. Одним из решений проблемы является стохастический градиентный спуск.

1.5 Стохастический градиентный спуск

Идея стохастического градиентного спуска состоит в том, чтобы приблизительно вычислить градиент ∇C, вычислив ∇C_x для небольшой случайной выборки обучающих входных данных. Посчитав их среднее, можно быстро получить хорошую оценку истинного градиента ∇C, и это помогает ускорить градиентный спуск, и, следовательно, обучение.

Стохастический градиентный спуск работает через случайную выборку небольшого количества m обучающих входных данных. Назовём эти случайные данные $(X_1, X_2, ..., X_m)$ мини-пакетом. Если размер выборки m будет достаточно большим, то среднее значение ∇C_{X_j} будет достаточно близким к среднему по всем ∇C_x, то есть

$$\frac{\sum_{j=1}^{m}\nabla C_{X_j}}{m} \approx \frac{\sum_{x}\nabla C_x}{n} = \nabla C\,, \tag{1.12}$$

где вторая сумма идёт по всему набору обучающих данных. Поменяв части местами, получим

$$\nabla C \approx \frac{1}{m}\sum_{j=1}^{m}\nabla C_{X_j}\,. \tag{1.13}$$

Из этого следует, что можно оценить общий градиент, вычислив градиенты для случайно выбранного мини-пакета.

С учетом этого можно записать правило обновления весов и смещений:

$$w_k \to w_k' = w_k - \frac{\eta}{m}\sum_{j}\frac{\partial C_{X_j}}{\partial w_k}\ , \tag{1.14}$$

$$b_l \to b_l' = b_l - \frac{\eta}{m}\sum_{j}\frac{\partial C_{X_j}}{\partial b_l}\,, \tag{1.15}$$

где идёт суммирование по всем обучающим примерам X_j в текущем мини-пакете.

В алгоритме стохастического градиентного спуска выбирается случайный мини-пакет входных данных, и нейросеть обучается на нем. Затем выбирается ещё один случайный мини-пакет и обучается на нём. И так далее, до тех пор, пока не закончатся все обучающие данные, это называется окончанием обучающей эпохи. После этого можно начать новую эпоху обучения. Эпоха – это один полный проход нейронной сети через все обучающие примеры из обучающего набора данных.

1.6 Обратное распространение ошибки

Алгоритм обратного распространения ошибки – это метод обучения глубоких нейронных сетей с учителем, который используется для минимизации ошибки прогнозирования. Процесс обратного распространения начинается с вычисления ошибки между прогнозируемым значением и фактическим значением. Затем эта ошибка распространяется обратно через сеть, чтобы определить, какие веса необходимо скорректировать.

Определим матрицу весов w^l (весовая матрица) для каждого слоя l. Элементами весовой матрицы будут веса соединённые со слоем l, то есть элементом в строке j и столбце k будет w^l_{jk}. Таким же образом определяется вектор смещений b^l для каждого слоя l.

Обратное распространение ошибки предполагает подсчёт частных производных $\partial C / \partial w^l_{jk}$ и $\partial C / \partial b^l_j$, которые используются для регулирования весов и смещений. Но для их вычисления сначала вычисляется промежуточное значение δ^l_j, которое называется ошибкой в нейроне j слоя l. Обратное распространение дает процедуру для вычисления ошибки δ^l_j, а потом связывает δ^l_j с $\partial C / \partial w^l_{jk}$ и $\partial C / \partial b^l_j$. Определим ошибку δ^l_j нейрона j в слое l как:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}, \tag{1.16}$$

будем называть выражение $z^l_j = \sum_{k=1}^{n_{(l-1)}} w^l_{jk} a^{l-1}_k + b^l_j$ – взвешенным входом функции активации $j-$го нейрона на слое l, $n_{(l-1)}$ – количество нейронов на предыдущем слое $(l-1)$, w^l_{jk} – вес связи между $k-$м нейроном на слое $(l-1)$ и $j-$м нейроном на слое l, a^{l-1}_k – выходное значение (активация) $k-$го нейрона на слое $(l-1)$, b^l_j – смещение $j-$го нейрона на слое l.

1.6.1 Четыре фундаментальных уравнения

Обратное распространение основано на четырёх фундаментальных уравнениях. Совместно они позволяют вычислить как ошибку δ^l (вектор ошибок слоя l), так и градиент функции стоимости.

Уравнение ошибки выходного слоя, δ^L. Компоненты вектора $\delta^L = \left(\delta^L_1, \delta^L_2, ...\right)$, задаются следующим образом:

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \sigma'(z^L_j). \tag{1.17}$$

Выражение $\partial C/\partial a_j^L$ измеряет, насколько быстро функция стоимости изменяется в зависимости от выходного нейрона на слое L. Если, к примеру, C слабо зависит от конкретного выходного нейрона j, то δ_j^L будет малым. Выражение $\sigma'(z_j^L)$, измеряет, насколько быстро функция активации σ меняется в z_j^L.

Уравнение (1.17) в матричной форме будет иметь вид:

$$\delta^L = \nabla_a C \odot \sigma'(z^L), \tag{1.18}$$

где $\odot$ – произведение Адамара, $\nabla_a C$ – вектор, чьими компонентами будут частные производные $\partial C/\partial a_j^L$.

Выражение ошибки δ^l через ошибку в следующем слое, δ^{l+1}:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \tag{1.19}$$

где $(w^{l+1})^T$ – транспонирование весовой матрицы w^{l+1} для слоя $(l+1)$. Предположим, что ошибка δ^{l+1} для слоя $(l+1)$ известна. Транспонирование весовой матрицы, $(w^{l+1})^T$, можно представить, как перемещение ошибки назад по сети, что даёт некую меру ошибки на выходе слоя l. Затем вычисляется произведение Адамара $\odot\sigma'(z^l)$. Это продвигает ошибку назад через функцию активации в слое l, давая значение ошибки δ^l во взвешенном входе для слоя l.

Комбинируя уравнения (1.19) с (1.18), считается ошибка δ^l для любого слоя сети. Подсчет начинается с использования уравнения (1.18) для вычисления ошибки на выходном слое δ^L. Затем рекурсивно применяется уравнение (1.19), последовательно вычисляется ошибка δ^{L-1} для предпоследнего слоя, затем δ^{L-2} для слоя перед ним, и так далее, продолжая движение в обратном направлении вплоть до первого скрытого слоя сети.

Уравнение для элемента вектора скорости изменения функции стоимости по отношению к любому смещению в сети:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l . \tag{1.20}$$

То есть, ошибка δ_j^l точно равна скорости изменения $\partial C / \partial b^l{}_j$.

Уравнение для элемента вектора скорости изменения функции стоимости по отношению к любому весу в сети:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l , \tag{1.21}$$

где a_k^{l-1} – активация $k-$го нейрона на слое $(l-1)$, а δ_j^l – ошибка $j-$ го нейрона в слое l.

Из уравнения (1.21) можно сделать вывод о том, что когда активация a_k^{l-1} мала, $a_k^{l-1} \approx 0$, то значение $\partial C / \partial w_{jk}^l$ тоже будет малым. В таком случае говорят, что вес не сильно меняется во время градиентного спуска.

Четыре фундаментальных уравнения справедливы для любой функции активации.

1.6.2 Алгоритм обратного распространения ошибки

Уравнения обратного распространения (1.17) – (1.20) позволяют сформулировать следующий порядок подсчёта градиента функции стоимости.

1. Вход x:вычислить соответствующую активацию a^1 для входного слоя.
2. Прямое распространение: для каждого $l = 2,3,\ldots,L$ вычислить $z^l = w^l a^{l-1} + b^l$ и $a^l = \sigma\left(z^l\right)$.
3. Выходная ошибка δ^L : вычислить вектор $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. Обратное распространение ошибки: для каждого $l = L-1, L-2,\ldots,2$ вычислить $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. Выход: для элементов градиента функции стоимости задаются частными производными $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ и $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

Для минимизации ошибки в алгоритме обратного распространения ошибки вычисляется градиент функции стоимости для всех обучающих примеров. То есть все примеры используются для одного обновления параметров модели, что является очень ресурсоёмким. К примеру, на практике, для более эффективной работы алгоритма обратного распространения ошибки его часто комбинируют со стохастическим градиентным спуском, и вычисляют градиент для каждого мини-пакета. В частности, при заданном мини-пакете m обучающих примеров, имеет место следующий алгоритм обратного распространения ошибки:

1. Вход: набор мини-пакетов x.
2. Для каждого мини-пакета x вычислить входной вектор $a^{x,1}$ на первом слое и выполнить следующие шаги:
   - Прямое распространение: для каждого $l = 2,3,\ldots,L$ вычислить $z^{x,l} = w^l a^{x,l-1} + b^l$ и $a^{x,l} = \sigma\left(z^{x,l}\right)$.
   - Выходная ошибка $\delta^{x,L}$: вычислить вектор $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - Обратное распространение ошибки: для каждого $l = L-1, L-2,\ldots,2$ вычислить $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
3. Градиентный спуск: для каждого $l = L, L-1,\ldots,2$ обновить веса со гласно правилу (1.14), и смещения согласно правилу (1.15).

Особенность обратного распространения в том, что она позволяет нам одновременно вычислять все частные производные $\partial C / \partial w_j$, используя только один прямой проход по сети, за которым следует один обратный проход. В некотором приближении, вычислительная стоимость алгоритмов прямого и обратного распространения сопоставима.

1.7 Перекрестная энтропия

Во время обучения нейронных сетей часто возникает проблема замедления обучения, более детально возникновение проблемы рассмотрено в практической части. Одним из решений проблемы для задач классификации будет замена квадратичной функции стоимости (1.4) другой функцией стоимости, известной под названием бинарная перекрёстная энтропия. Функция стоимости с бинарной перекрёстной энтропией для нейрона определяется как:

$$C = -\frac{1}{n}\sum_{x}\left[y \ln a + (1-y)\ln(1-a)\right], \qquad (1.22)$$

где n – общее количество единиц обучающих данных (сумма идёт по всем обучающим данным x), y – соответствующий желаемый выход, a – выход нейросети.

При использовании бинарной перекрестной энтропии параметры y и a принимают значения от 0 до 1.

Перекрёстную энтропию в качестве функции стоимости можно использовать по причине наличия у нее двух свойств:

– Первое свойство заключается в том, что, она всегда больше нуля $C > 0$. Можно увидеть, что все отдельные члены суммы в (1.22) отрицательны, поскольку оба логарифма берутся от чисел в диапазоне от 0 до 1, и перед суммой стоит знак минус.

– Второе свойство заключается в том, что если реальный выход нейросети близок к желаемому выходу для всех обучающих входов x, то перекрестная энтропия будет близка к нулю. Для доказательства этого свойства необходимо предположить, что желаемые выходы y будут равны либо 0, либо 1, это как раз возможно при использовании бинарной перекрестной энтропии.

Представим, что $y = 0$ и $a \approx 0$ для некоего входного x. Если сеть хорошо обучена, то условие $a \approx 0$ будет выполняться. Видим, что первый член выражения (1.22) для стоимости исчезает, поскольку $y = 0$, а второй член будет равен $-ln(1-a) \approx 0$. То же выполняется, когда $y = 1$ и $a \approx 1$. Поэтому вклад стоимости будет небольшим, если реальный выход будет близким к желаемому.

На основании приведенного выше, можно сделать вывод о том, что перекрёстная энтропия положительна, и стремится к нулю, когда нейрон лучше вычисляет желаемый выход y для всех обучающих входов x.

В отличии от квадратичной функции стоимости (1.4), у функции стоимости с перекрёстной энтропией есть преимущество – отсутствие проблемы замедления обучения. Чтобы убедиться в этом, необходимо подсчитать частную производную стоимости с перекрёстной энтропией по весам.

Подставив $a=\sigma(z)$ в (1.22), и дважды применив правило дифференцирования сложной функции получим:

$$\frac{\partial C}{\partial w_j}=-\frac{1}{n}\sum_x\left(\frac{y}{\sigma(z)}-\frac{(1-y)}{1-\sigma(z)}\right)\frac{\partial\sigma}{\partial w_j}=-\frac{1}{n}\sum_x\left(\frac{y}{\sigma(z)}-\frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j. \quad (1.23)$$

Приведя к общему знаменателю и упростив, получим:

$$\frac{\partial C}{\partial w_j}=\frac{1}{n}\sum_x\frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z)-y). \quad (1.24)$$

Используя определение сигмоиды $\sigma(z)=1/\left(1+e^{-z}\right)$, можно показать, что $\sigma'(z)=\sigma(z)\left(1-\sigma(z)\right)$. Подставив это в уравнение (1.24) члены $\sigma'(z)$ и $\sigma(z)\left(1-\sigma(z)\right)$ сократятся, и это приведёт к:

$$\frac{\partial C}{\partial w_j}=\frac{1}{n}\sum_x x_j(\sigma(z)-y). \quad (1.25)$$

Из этого выражения следует, что скорость, с которой обучаются веса, контролируется $\sigma(z)-y$, то есть, ошибкой на выходе. Чем больше ошибка, тем быстрее обучается нейрон.

Аналогично можно вычислить частную производную для смещения, она будет иметь вид:

$$\frac{\partial C}{\partial b}=\frac{1}{n}\sum_x(\sigma(z)-y). \quad (1.26)$$

2 Реализация и повышение точности нейронной сети

В данной части рассматривается создание и обучение нейронной сети, известной как многослойный персептрон, без использования специализированных фреймворков. Тип примененного обучения – с учителем. Для этого был выбран язык программирования Python версии 3.11. В процессе разработки использовались стандартные библиотеки Python, а также библиотека numpy для более быстрой работы с массивами и матрицами. Для построения графиков использовались библиотеки matplotlib и PIL.

Здесь не будет представлена конкретная программная реализация, но будут описаны основные подходы к созданию и обучению нейронной сети. Будут представлены результаты обучения сети, включая постепенное увеличение точности распознавания данных. Это позволит лучше понять процесс обучения нейронной сети и ее способность к адаптации и улучшению результатов.

Для обучения нейронной сети использовался данные из архива CoMNIST (Cyrillic-oriented MNIST). Архив содержит примерно 93000 элементов. Каждый элемент представляет собой изображение рукописной буквы русского алфавита размером 28 на 28 пикселей (суммарно 784 пикселя) черного цвета на белом фоне. Данные, содержащиеся в CoMNIST изначально не структурированы и, таким образом, не подходят для обучения нейронных сетей. В этой связи, потребовалось преобразование указанных данных в вид, удовлетворяющий требованиям обучения и тестирования нейросетей.

2.1 Подготовка датасета

Подготовка датасета для нейросети является одной из ключевых задач в процессе создания и обучения искусственного интеллекта. При обучении нейронная сеть на основе входных данных предсказывает выходные данные, которые сравниваются с правильными ответами. Поэтому обучающие данные играют решающую роль в её конечной производительности и точности. Датасет должен содержать примеры с правильными ответами. Неправильно подготовленные или плохо структурированные данные могут привести к снижению качества предсказаний модели, а в некоторых случаях – к полной её неработоспособности.

2.1.1 Трудности при подготовке данных для обучения

Нейросеть будет обучаться на тех данных, которые ей предоставлены. Если данные содержат ошибки, пропуски или нерелевантную информацию, это может существенно снизить точность модели. Важно убедиться, что данные являются точными, полными и актуальными. Также нейросетям, особенно глубоким, требуется большое количество данных для обучения. Недостаток данных может привести к тому, что модель не сможет выявить важные закономерности и будет плохо обобщаться на новые данные.

Важным аспектом является правильная разметка данных, особенно в задачах классификации или сегментации. Неправильная разметка может ввести модель в заблуждение, что приведёт к ошибочным предсказаниям.

В случае, если данные имеют несбалансированные классы (например, одни классы представлены значительно больше, чем другие), модель может научиться предсказывать преимущественно доминирующий класс, игнорируя менее представленные.

Качественная подготовка данных для обучения позволяет:

- Улучшить точность предсказаний. Хорошо подготовленные данные обеспечивают модели возможность выявлять ключевые детали и зависимости.
- Повысить устойчивость модели. Подготовленные данные помогают обучить модель, которая лучше справляется с новыми и непредсказуемыми данными.

Для подготовки датасета была написана отдельная программа на вход которой подается архив изображений, а на выходе получается два файла с наборами данных для обучения и проверки обучения нейронной сети.

Полученные наборы данных содержат следующее количество примеров:

- Для обучения используется 76000 примеров.
- Для проверки обучения нейронной сети 16500 примеров.

Причем, примеры для обучения и оценки не совпадают. Это помогает убедиться в том, что система может распознавать примеры, которые она не встречала при обучении.

2.1.2 Реализация

Для упорядочивания данных, каждое изображение буквы было помещено в папку с номером, соответствующим типу буквы. Таким образом все изображения буквы А, были помещены в папку с именем 0, изображения буквы Б в папку 1 и т.д.. Количество примеров на одну букву варьируется в пределах от 2000 до 3000. В датасете также присутствовали некорректные изображения от них пришлось избавится ради более точного обучения. Примеры изображений букв:

Рисунок 5 – Буквы: ж, з, л, р, у, щ, а, ю

Каждое изображение преобразуются в массив размером 784 элементов, в котором каждый элемент соответствует цвету пикселя. Для удобства подачи на вход нейронной сети изображение записывается в одномерный массив таким образом, чтобы каждый элемент массива подавался на соответствующий входной нейрон. Так как изображение буквы черное, то мы получаем градацию серого цвета от 0 до 1, где 0 черный цвет 1 белый. Пример массива с буквой Ю, исходное изображение которой представлено на Рисунке 5, представлен на Рисунке 6:

```
1.000 1.000 1.000 1.000 1.000 1.000 0.996 0.996 0.992 0.996 0.992 0.988 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 0.996 0.992 0.988 0.992 0.984 0.992 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 0.992 0.992 0.992 0.996 0.992 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 0.988 0.980 0.890 0.922 0.992 0.996 1.000 0.996 1.000 1.000 0.996 1.000 0.996 0.996 0.996 1.000 0.996 0.996 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 0.996 0.988 0.827 0.412 0.737 0.992 0.992 0.988 0.984 0.984 0.992 0.992 0.988 0.953 0.824 0.863 0.976 0.984 0.976 1.000 0.996 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 0.992 0.643 0.094 0.663 0.996 0.996 0.988 0.980 0.941 0.988 0.988 0.941 0.584 0.149 0.282 0.835 0.988 0.984 0.984 0.992 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 0.996 0.596 0.071 0.769 0.988 0.992 0.988 0.988 0.957 0.863 0.686 0.333 0.075 0.055 0.094 0.651 0.965 0.988 0.984 0.984 0.996
1.000 1.000 1.000 1.000 1.000 1.000 0.996 0.992 0.522 0.243 0.914 0.992 0.976 0.949 0.831 0.514 0.216 0.086 0.035 0.337 0.510 0.082 0.443 0.949 0.976 0.992 0.984 0.996
1.000 1.000 1.000 1.000 1.000 1.000 0.996 0.969 0.424 0.384 0.992 0.973 0.878 0.580 0.200 0.020 0.031 0.020 0.298 0.855 0.925 0.282 0.259 0.878 0.980 0.992 0.988 1.000
1.000 1.000 0.984 0.992 0.996 0.996 0.992 0.910 0.294 0.467 0.953 0.659 0.286 0.161 0.176 0.114 0.020 0.208 0.749 0.980 0.941 0.471 0.086 0.710 0.984 0.984 0.996 1.000
1.000 1.000 0.988 0.992 0.992 0.996 0.988 0.804 0.184 0.471 0.612 0.169 0.180 0.576 0.765 0.392 0.106 0.659 0.957 0.980 0.980 0.710 0.082 0.553 0.961 0.984 1.000 1.000
1.000 1.000 0.996 0.996 0.992 0.988 0.953 0.561 0.082 0.063 0.055 0.204 0.788 0.973 0.875 0.224 0.459 0.949 0.988 0.976 0.980 0.820 0.204 0.435 0.937 0.996 0.992 1.000
1.000 1.000 0.992 0.992 0.996 0.980 0.875 0.263 0.024 0.024 0.345 0.741 0.969 0.969 0.639 0.098 0.725 0.976 0.988 0.992 0.984 0.851 0.239 0.365 0.925 0.996 0.988 1.000
1.000 0.996 0.992 0.992 0.988 0.992 0.725 0.063 0.020 0.212 0.867 0.973 0.980 0.945 0.388 0.184 0.894 0.988 0.984 0.988 0.992 0.827 0.153 0.325 0.925 0.988 0.992 1.000
1.000 1.000 0.996 0.984 0.984 0.961 0.557 0.024 0.137 0.729 0.984 0.984 0.984 0.906 0.275 0.220 0.929 0.996 0.988 0.980 0.992 0.804 0.106 0.365 0.933 0.992 0.996 1.000
1.000 1.000 0.992 0.984 0.980 0.906 0.294 0.016 0.478 0.929 0.992 0.992 0.980 0.886 0.271 0.204 0.945 0.988 0.988 0.984 0.984 0.792 0.098 0.502 0.957 0.992 0.996 1.000
1.000 1.000 0.992 0.980 0.984 0.757 0.122 0.204 0.780 0.969 1.000 0.996 0.992 0.918 0.306 0.196 0.886 0.976 0.984 0.980 0.984 0.659 0.094 0.667 0.980 0.996 0.988 1.000
1.000 0.996 0.992 0.984 0.992 0.576 0.024 0.451 0.973 1.000 0.988 0.984 0.976 0.965 0.455 0.035 0.420 0.867 0.969 0.973 0.827 0.247 0.267 0.835 0.973 0.984 0.988 1.000
1.000 1.000 0.992 0.992 0.980 0.380 0.122 0.702 0.988 0.992 0.988 0.988 0.984 0.984 0.827 0.149 0.075 0.373 0.671 0.667 0.337 0.184 0.690 0.969 0.988 0.980 0.996 1.000
1.000 0.996 0.996 1.000 0.906 0.078 0.200 0.925 0.992 0.992 0.992 0.996 0.996 0.984 0.984 0.671 0.176 0.027 0.039 0.047 0.055 0.498 0.941 0.996 0.992 1.000 0.996 1.000
1.000 0.996 0.988 0.973 0.584 0.039 0.431 0.980 1.000 1.000 1.000 1.000 0.996 0.984 0.980 0.961 0.792 0.549 0.294 0.231 0.522 0.875 0.992 1.000 1.000 1.000 1.000 1.000
1.000 1.000 0.988 0.918 0.302 0.071 0.639 0.996 1.000 1.000 1.000 0.996 0.988 0.988 0.992 0.996 0.976 0.945 0.898 0.867 0.937 0.976 0.996 1.000 1.000 1.000 1.000 1.000
1.000 0.996 0.980 0.902 0.255 0.196 0.843 1.000 1.000 1.000 1.000 0.996 0.992 0.996 0.980 0.988 0.996 0.992 0.992 0.992 0.976 0.988 1.000 1.000 1.000 1.000 1.000 1.000
1.000 0.996 0.992 0.937 0.502 0.514 0.937 1.000 1.000 1.000 1.000 0.996 0.984 0.988 0.992 0.996 0.984 0.980 0.984 0.992 0.996 0.988 0.996 1.000 1.000 1.000 1.000 1.000
1.000 1.000 0.996 0.992 0.910 0.941 0.992 1.000 1.000 1.000 1.000 0.996 0.992 0.996 1.000 0.992 0.992 0.992 0.996 0.992 0.996 0.992 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
```

Рисунок 6 – Буква Ю

Изображения записываются в массив по порядку от А до Я. Сначала все вариации буквы А затем Б и так до Я. Каждому изображению присваивается свой номер буквы от 0 до 32. В итоге получаем массив кортежей, где первый элемент кортежа представляет собой массив, содержащий изображения, а второй элемент соответствующее значение буквы. Данный массив кортежей имеет следующий вид:

$$\underbrace{\left[\overbrace{\left(\overbrace{[...,0.964,0.913,....]}^{784},25\right)}^{},...,\left([...,0.546,0.586,....],6\right),...\right]}_{93000}$$

Рисунок 7– массив кортежей

Далее элементы этого массива (кортежи) перемешиваются и разделяются на набор обучающих данных и на набор данных для проверки обучения нейронной сети. Более наглядно алгоритм изображен на следующей блок схеме:

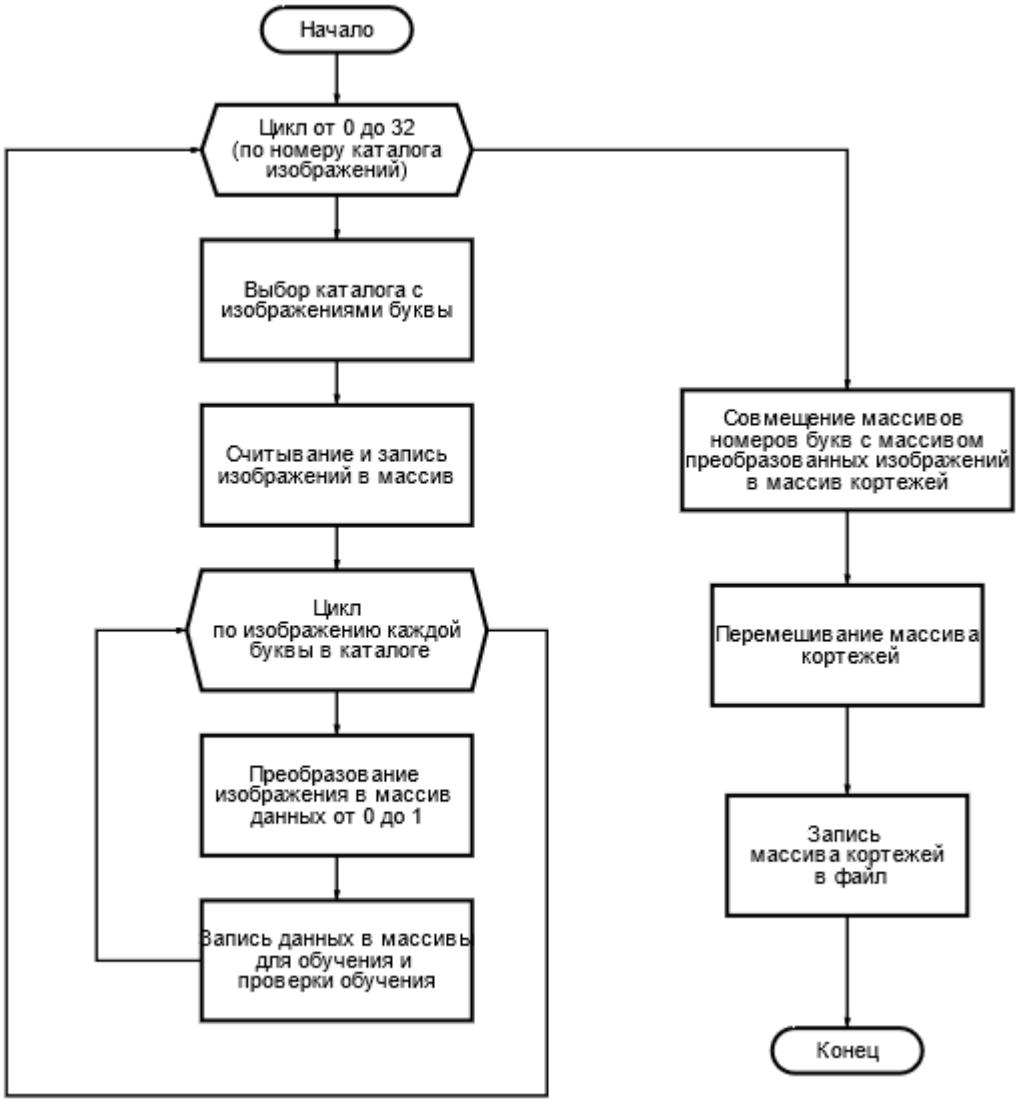


Рисунок 8 – блок схема подготовки датасета

2.2 Обучение нейронной сети

Качество работы нейронной сети напрямую зависит от установленных при её обучении значений гиперпараметров, таких как:

- количество слоев;
- количество нейронов в слое;
- количество эпох обучения;
- скорость обучения η;
- размер мини-пакета m.

Как правило, подбор гиперпараметров занимает значительное время. Это связано с тем, что при изменении любого параметра необходимо повторно проводить обучение нейронной сети с последующим анализом влияния изменений на результат работы. В частности, размер мини-пакета и скорость обучения в нейронных сетях существенно влияют на процесс обучения, в особенности при использовании стохастического градиентного спуска. Выбор оптимальных значений гиперпараметров часто осуществляется эмпирически. В нашем случае эффективность работы нейронной сети обуславливается точностью распознавания рукописных букв. Таким образом ключевой подзадачей обучения рассматриваемой нейросети является задача максимизации точности её работы.

Влияние определенных гиперпараметров на поведение сети:

В работе использована глубокая нейронная сеть, содержащая 4 слоя: входной, выходной слои и два скрытых. Большее количество скрытых слоев в сети, использующей градиентный спуск, будет замедлять скорость обучения.

Количество нейронов в сети влияет на сложность решаемой задачи и способность сети к обобщению.

Эпоха – это один полный проход нейронной сети через все обучающие примеры из обучающего набора данных. Выбор количества эпох – это баланс между качеством обучения и временем, затрачиваемым на него.

Скорость обучения определяет, насколько быстро веса модели обновляются в направлении градиента. Слишком большая скорость обучения может привести к тому,

что модель перестанет обучаться, а слишком маленькая может замедлить процесс обучения.

Размер мини-пакета определяет, сколько образцов данных используется для вычисления градиента на каждом шаге обучения в эпохе. Большие мини-пакеты могут привести к более стабильному обновлению весов, но могут потребовать больше памяти и вычислительных ресурсов. Маленькие мини-пакеты могут привести к более хаотичным обновлениям градиента, но могут ускорить обучение.

Обучающие входные данные обозначим через x. Каждое входное изображение x представим как вектор с $28 \cdot 28 = 784$ элементами. Каждая элемент внутри вектора обозначает яркость одного пикселя изображения. Выходное значение будем обозначать, как $y = y(x)$, где y – вектор. В котором 32 нулевых элемента, и один элемент равный единице. К примеру, для буквы А – первый элемент равняется единице, а остальные нулю.

Для обучения используется набор данных, содержащий 76635 изображений, для оценки точности набор данных, содержащий 16500 изображений. Начальный этап обучения и отладки нейронной сети предполагает использование следующих базовых концепций:

- квадратичной функции стоимости (1.4);
- случайного гауссова распределения от 0 до 1 для инициализации (выбора начальных значений) весов и смещений;
- сигмоидной функции активации (1.1) на скрытых слоях;
- функции softmax (1.3) на выходном слое.

2.3 Структура программы

Для наглядного представления процесса работы нейронной сети по обучению распознавания рукописных букв представлена блок-схема (см. Рисунок 9), демонстрирующая основные этапы работы программы.

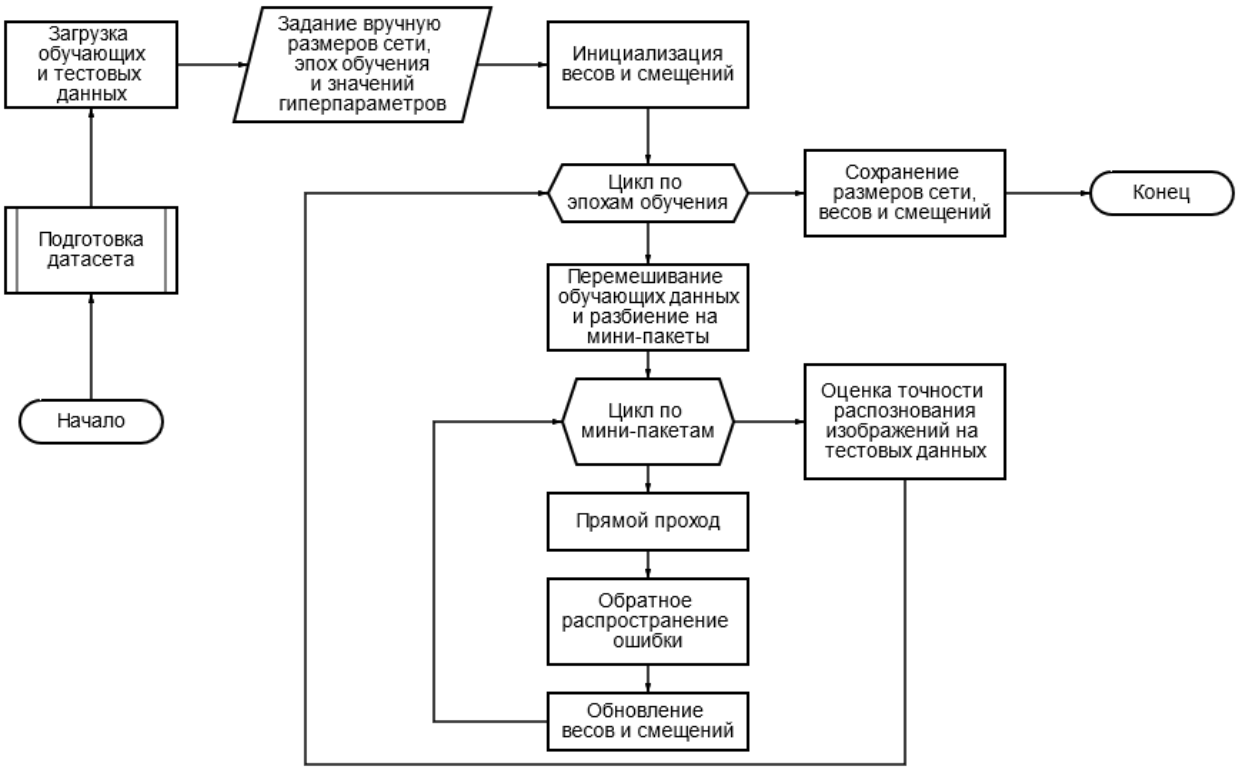


Рисунок 9 – Блок схема программы нейронной сети

Описание программы:

1 Перед началом обучения все веса и смещения сети инициализируются случайными значениями.

2 Цикл по эпохам обучения (стохастический градиентный спуск): обучение происходит итеративно, эпоха за эпохой:

2.1 Перемешивание обучающих данных: в начале каждой эпохи данные обучающей выборки перемешиваются. Это помогает избежать зацикливания алгоритма обучения и улучшает обобщающую способность модели.

2.2 Разбиение на мини-пакеты: обучающая выборка разбивается на небольшие порции – мини-пакеты. это позволяет ускорить обучение и сделать его более стабильным.

2.3 Цикл по мини-пакетам: для каждого мини-пакета выполняются следующие шаги:

2.3.1 Прямой проход: данные мини-пакета подаются на вход сети, и для каждого нейрона на каждом слое вычисляется выходной сигнал. В результате на выходе сети формируется вектор вероятностей для каждой буквы алфавита.

2.3.2 Обратное распространение ошибки: вычисляется ошибка между наиболее вероятной буквой и фактическим значением для данного мини-пакета. Ошибка распространяется обратно по сети, и для каждого параметра (веса и смещения) вычисляется градиент, усреднённый по мини-пакету, указывающий, в каком направлении нужно изменить этот параметр, чтобы уменьшить ошибку.

2.3.3 Обновление весов и смещений: с помощью алгоритма стохастического градиентного спуска веса и смещения сети корректируются в направлении, уменьшающем ошибку.

2.4 Оценка на тестовой выборке: после каждой эпохи сеть оценивается на тестовой выборке, чтобы контролировать прогресс обучения и предотвратить переобучение.

3 Сохранение модели: после завершения обучения модель (количество слоев и нейронов сети, веса и смещения) сохраняется в файл, чтобы ее можно было использовать в дальнейшем без повторного обучения.

2.4 Первоначальный выбор значений гиперпараметров

Для нахождения оптимальных значений гиперпараметров был использован следующий алгоритм, основанный на анализе результатов и поэтапном уточнении:

1. Установка начальных значений:
   - Выбираются исходные значения гиперпараметров, основываясь на общих рекомендациях для аналогичных задач и интуиции.
2. Обучение и оценка модели:
   - Нейронная сеть обучается с текущими значениями гиперпараметров.
   - После обучения производится оценка точности модели на тестовом наборе данных.

3. Анализ результатов:
    - На основе тестовых данных строятся графики зависимости точности от количества пройденных эпох.
    - Также оценивается влияние изменения каждого гиперпараметра на точность в предыдущих экспериментах.
4. Корректировка гиперпараметров:
    - На основе анализа выбирается гиперпараметр, изменение которого, предположительно, могло сильнее всего повлиять на точность.
    - Значение выбранного гиперпараметра изменяется, а остальные гиперпараметры временно фиксируются.
5. Повторение шагов 2-4:
    - Шаги 2-4 последовательно повторяются до тех пор, пока не будет достигнута желаемая точность модели или дальнейшие изменения гиперпараметров не будут приводить к существенному улучшению.

В качестве отправной точки были выбраны значения, основанные на опыте и рекомендациях для схожих задач. Изначально было выбрано следующее количество нейронов для скрытых слоев: второй слой – 128 нейронов, третий слой – 64 нейрона. Первый и четвертый слой являются входным и выходным соответственно. Размер мини-пакета был выбран в 30 изображений.

Подбор гиперпараметров начался с изменения скорости обучения и отслеживания вида графика точности распознавания для тестовых данных. Графики обучения нейросети представлены ниже:

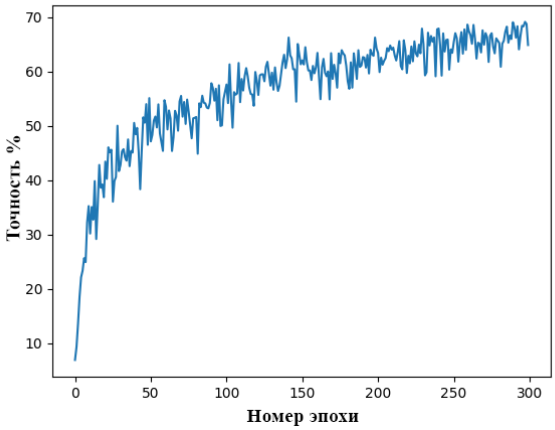


Рисунок 10 – $\eta = 3$

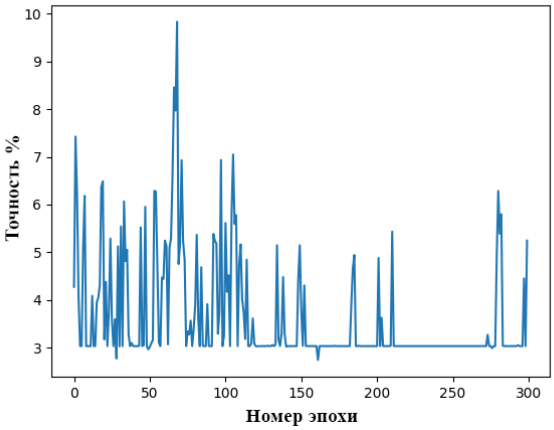


Рисунок 11 – $\eta = 4$

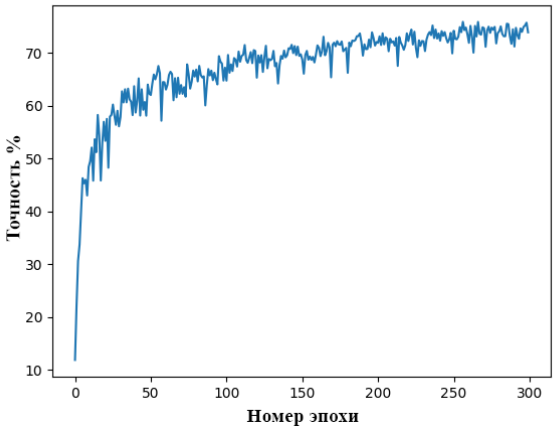


Рисунок 12 – $\eta = 2$

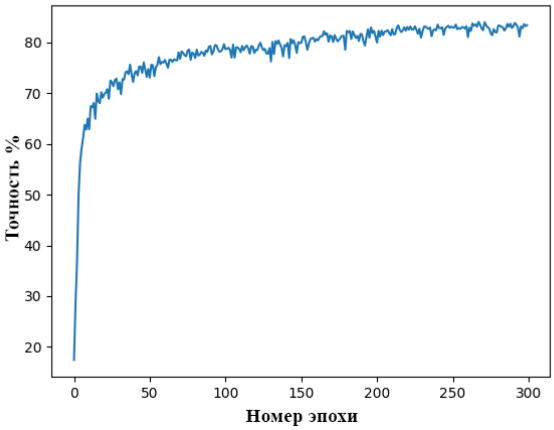


Рисунок 13 – $\eta = 1$

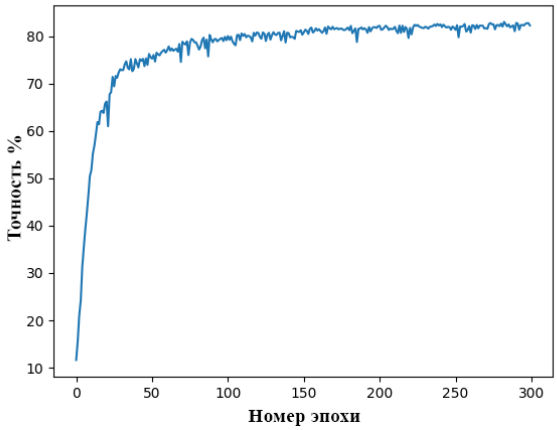


Рисунок 14 – $\eta = 0.8$

Из Рисунков 10 – 14 можно сделать некоторые выводы о том, как скорость обучения η влияет на точность распознавания изображений. Чем больше η тем меньше точность, и обучение более хаотично. Из Рисунка 11 можно сделать вывод, что

при $\eta = 4$ обучение не приносит положительных результатов, скорее всего наблюдается осцилляция функции стоимости C вблизи минимума. В свою очередь можно заметить, что при уменьшении скорости обучения η, обучение проходит более плавно, это видно по уменьшению размера скачков на графиках. При этом точность распознавания увеличивается с, почти, 70% – Рисунок 10 до, примерно, 83% – Рисунок 14. Однако все время уменьшать скорость обучения η нельзя, так при $\eta < 0.8$ обучение практически не происходит.

Из Рисунков 10, 12, 13, 14 можно заметить, что с течением эпох возникает проблема замедления обучения.

2.5 Замедление обучения

Чтобы понять источник проблемы, вспомним, что нейрон обучается через изменение веса и смещения со скоростью, определяемой частными производными функции стоимости, $\partial C / \partial w$ и $\partial C / \partial b$. Сказать, что обучение идёт медленно, это всё равно, что сказать, что эти частные производные малы. Для понимания почему они малы нужно вычислить частные производные.

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z), \tag{2.1}$$

$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z) = a\sigma'(z), \tag{2.2}$$

где $a = \sigma(z)$, $z = wx + b$. Чтобы понять поведение этих выражений, можно исследовать член $\sigma'(z)$ справа. Сигмоида имеет форму:

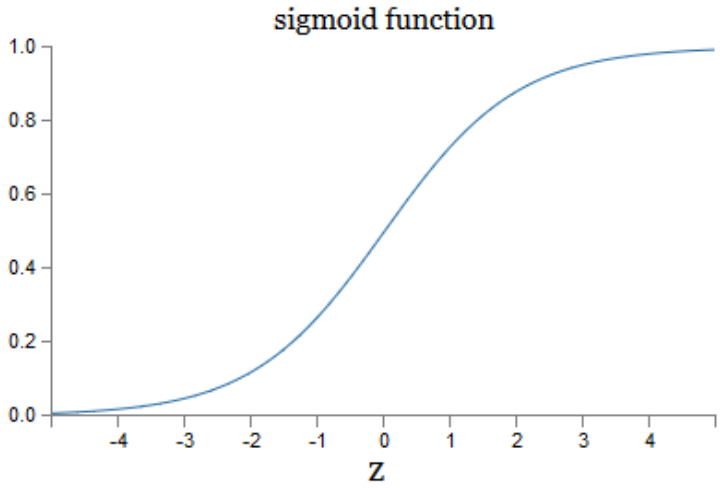


Рисунок 15 – график сигмоиды $\sigma(z)$

Из графика (см. Рисунок 15) видно, что, когда значение $\sigma(z)$ близко к 0 или 1, кривая становится плоской и, следовательно, значение $\sigma'(z)$ становится малым. По этой причине в уравнениях (2.1) и (2.2) значения $\partial C / \partial w$ и $\partial C / \partial b$ становятся также малыми. Отсюда и возникает замедление обучения.

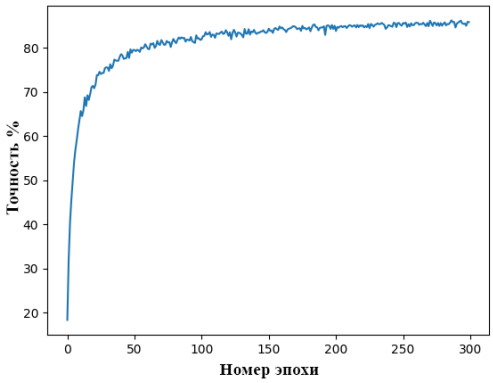
2.6 Использование перекрестной энтропии

С этого момента для более эффективного, и, в частности, быстрого обучения нейросети будет использоваться функция стоимости с перекрестной энтропией (1.22), рассмотренная в теоретической части.

При использовании функции стоимости с перекрестной энтропией, член $\sigma'(z)$ в (2.1) сокращается, что позволяет не волноваться о его малом значении. Это сокращение является свойством, гарантируемым функцией стоимости с перекрёстной энтропией. Аналогичное справедливо и для уравнения (2.2), при использовании перекрестной энтропии его можно будет записать в виде (1.26) и избавиться от члена вызывающего замедление обучения.

Далее в программе уравнения (2.1), (2.2) будут заменены уравнения на (1.25) и (1.26) соответственно. Размер мини-пакета примем равным 15.

Продолжая подбирать скорость обучения η были получены следующие результаты работы нейросети:



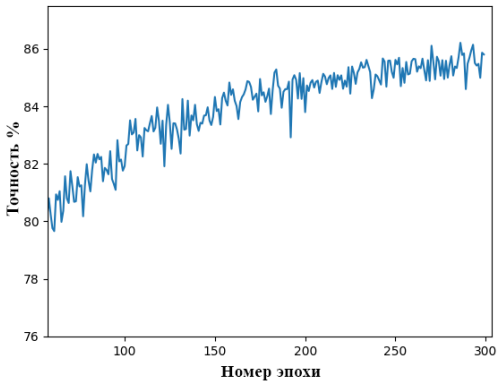
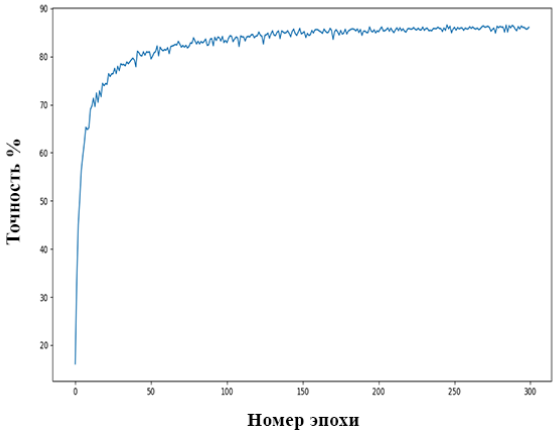


Рисунок 16 – $\eta = 0.05$



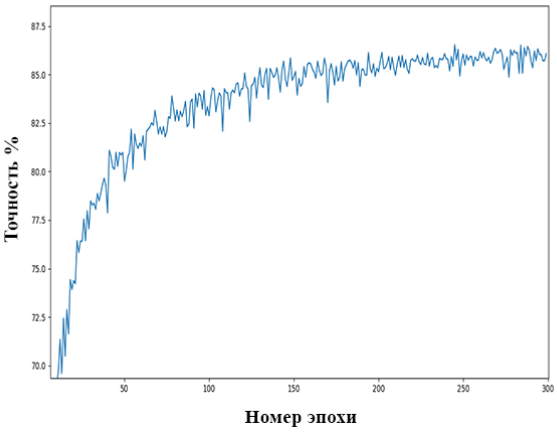
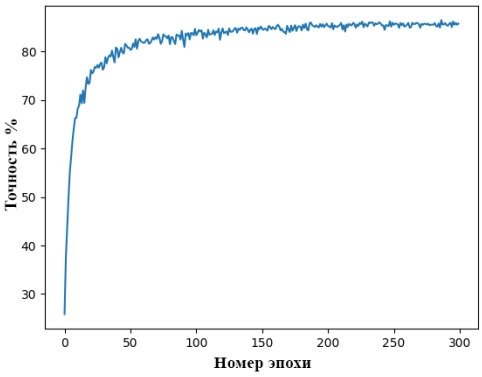


Рисунок 17 – $\eta = 0.075$



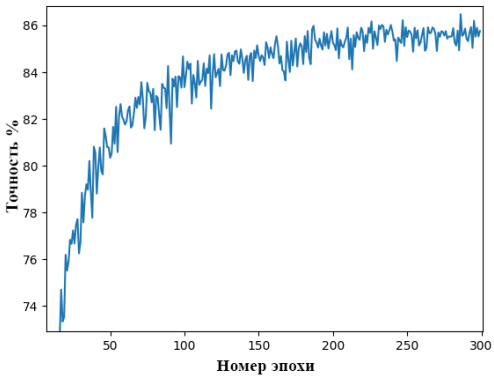


Рисунок 18 – $\eta = 0.09$

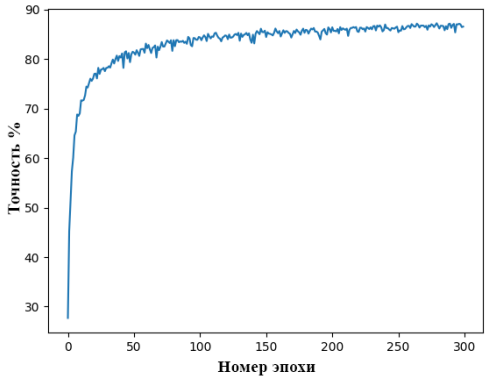


Рисунок 19 – $\eta = 0.1$

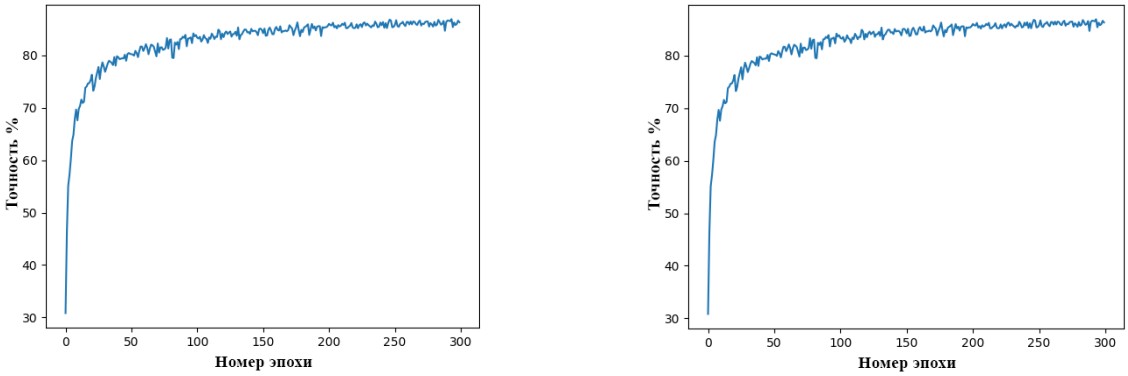


Рисунок 20 – $\eta = 0.12$

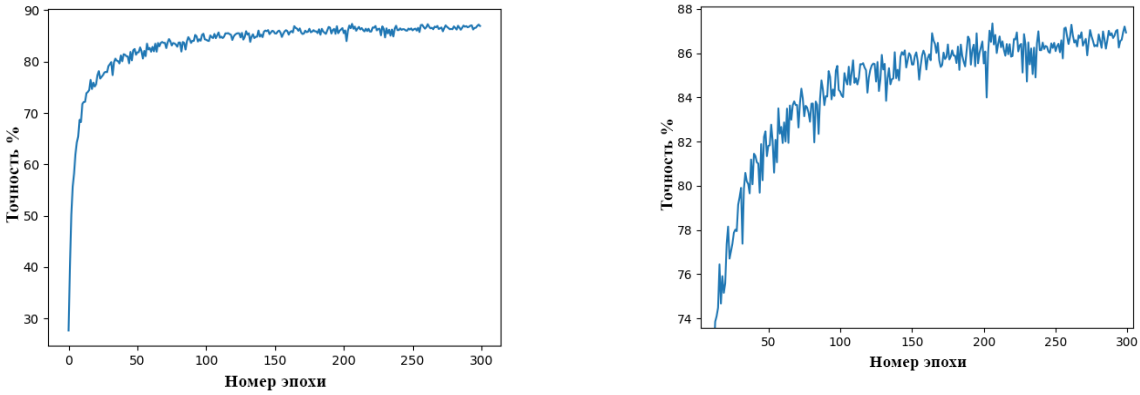


Рисунок 21 – $\eta = 0.11$

Рассмотрим 3 примера: Рисунок 16, 17, 19. Исходя из опыта с квадратичной функции стоимости (1.4), начальная скорость обучения была выбрана небольшой, она составила $\eta = 0.1$ (Рисунок 19), потом $\eta = 0.05$ (Рисунок 16), что в два раза меньше, и затем $\eta = 0.075$ (Рисунок 17). Для этих скоростей обучения итоговые точности составили: 86.2% для $\eta = 0.05$, 86.54% для $\eta = 0.075$ и 87% для $\eta = 0.1$. Можно сделать вывод, что уменьшение скорости менее чем $\eta = 0.1$ не повысит точности нейросети. В

таком случае можно выбрать стратегию постепенного увеличения скорости обучения. Однако данный вывод верен только для текущих гиперпараметров, при другой конфигурации параметров может оказаться, что более эффективная скорость будет менее чем $\eta = 0.1$.

Далее были выбраны скорости $\eta = 0.11$ и $\eta = 0.12$. При $\eta = 0.12$ точность была меньше чем при $\eta = 0.1$ следовательно, нужно выбирать скорость меньше чем $\eta = 0.12$. Поэтому была выбрана скорость $\eta = 0.11$, при которой была достигнута новая максимальная точность в 87.4%.

По виду полученных графиков (Рисунки 16 – 21) можно сделать вывод, что при использовании функции стоимости с перекрестной энтропией (1.22), на начальных этапах обучение проходит быстрее чем при использовании квадратичной функции стоимости. Также заметен рост точности, а при $\eta = 0.11$ достигается максимальная точность в 87.4%, что составляет 14413 правильных ответов из 16500 тестовых изображений.

2.7 Подбор мини-пакета

Дальнейшее изменение скорости обучения, к примеру, на $\eta = 0.115$ или $\eta = 0.105$ скорее всего дало бы небольшой прирост точности. Поэтому далее будет представлен подбор размера мини-пакета m.

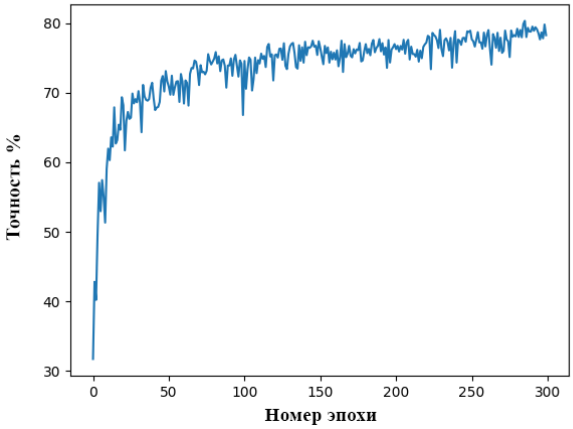


Рисунок 22 – $m = 5$

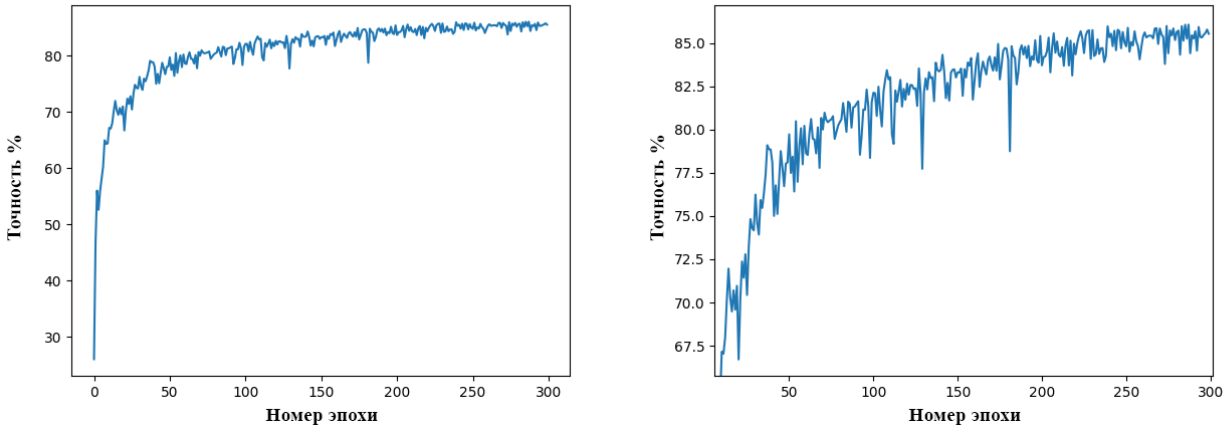


Рисунок 23 – $m = 10$

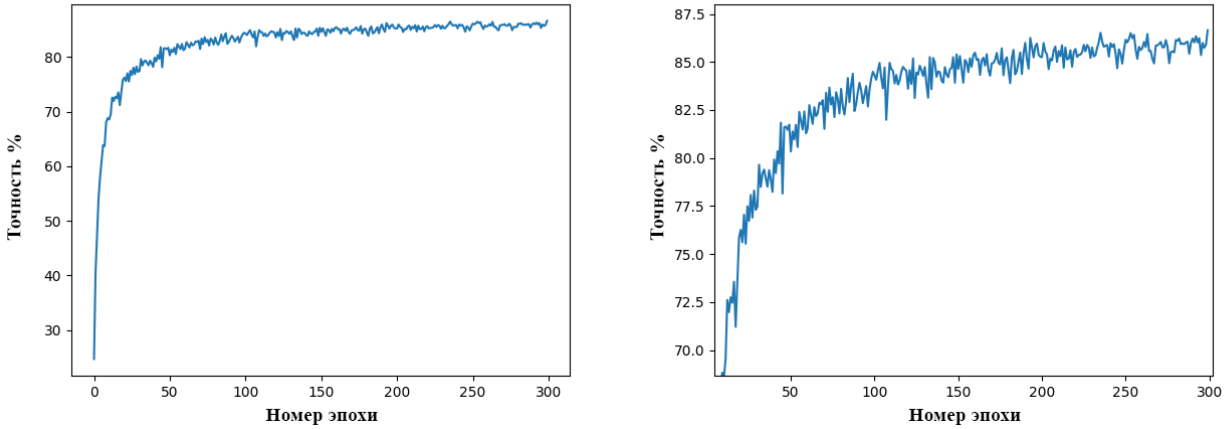


Рисунок 24 – $m = 20$

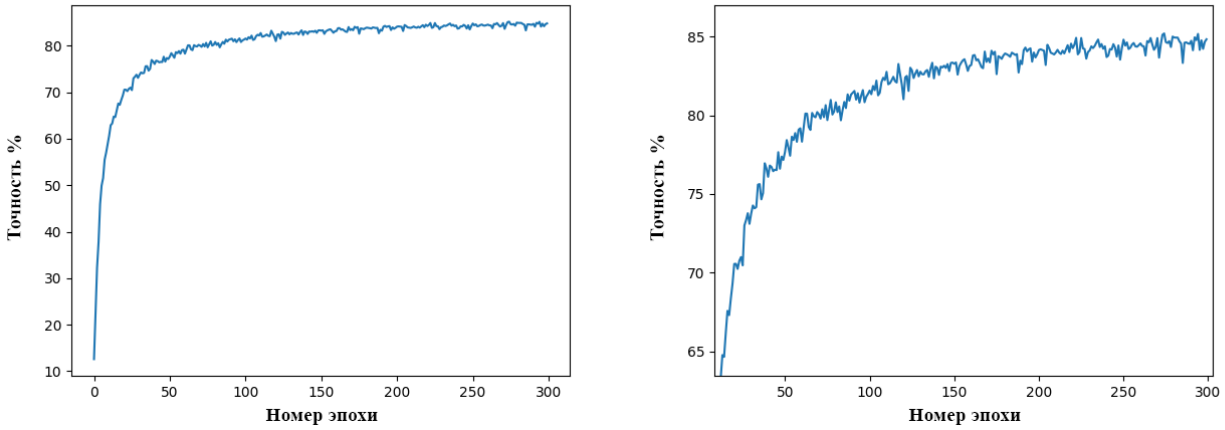


Рисунок 25 – $m = 50$

На основании полученных данных составлена Таблица 1.

Таблица 1 – Точность нейросети в зависимости от скорости обучения η и мини-пакета m.

η	m	Точность %
0.11	5	80.4
0.11	10	86.0
0.11	15	87.4
0.11	20	86.6
0.11	50	85.2

По полученным результатам можно сделать вывод, что изначальный размер мини-пакета $m=15$ был подобран удачно, так как при нем достигается максимальная точность. Также можно заметить, что при увеличении размера мини-пакета, график точности становится более гладким. Это связано с тем, что обучение существенно замедляется, так как «шаги» становятся меньше и требуется больше времени для нахождения глобального минимума.

2.8 Изменение количества нейронов и слоев нейросети

Количество нейронов и слоев в нейросети может существенно влиять на её точность. В целом, более глубокие нейронные сети имеют больший потенциал для обучения сложных зависимостей в данных и, следовательно, могут достигать более высокой точности.

Большее количество слоев в нейросети также может увеличить её способность к извлечению более сложных и абстрактных признаков из данных, что может улучшить точность. Однако, более глубокие сети могут быть более трудными для обучения и требовать большего количества данных для успешного обучения.

До этого у сети было два скрытых слоя, количество нейронов в них составляло 128 в первом и 64 во втором. Теперь рассмотрим модели с следующим количеством нейронов: 128 и 96 (Рисунок 26), 192 и 128 (Рисунок 27), 256 и 128 (Рисунок 28). Также рассмотрим модель с тремя скрытыми слоями (Рисунок 29), у которой 256, 128, 64 нейрона в слоях.

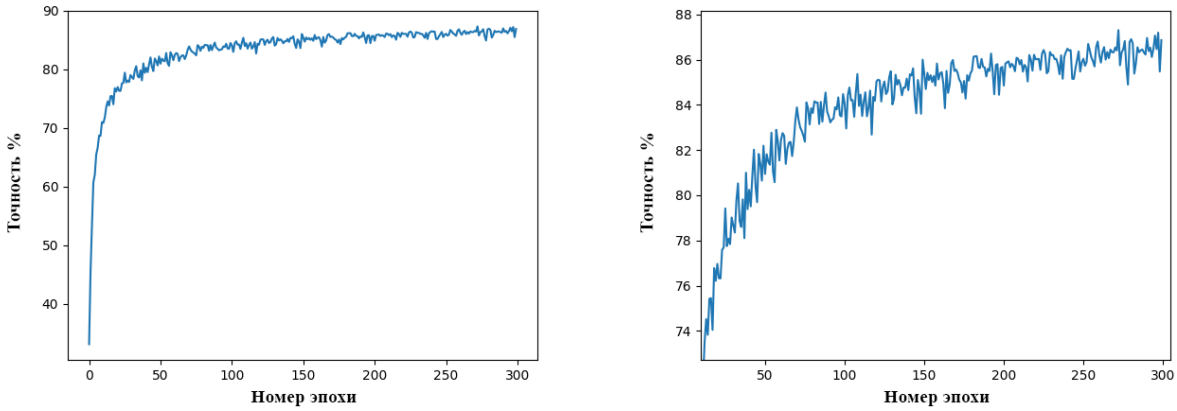


Рисунок 26 – 128, 96 нейронов

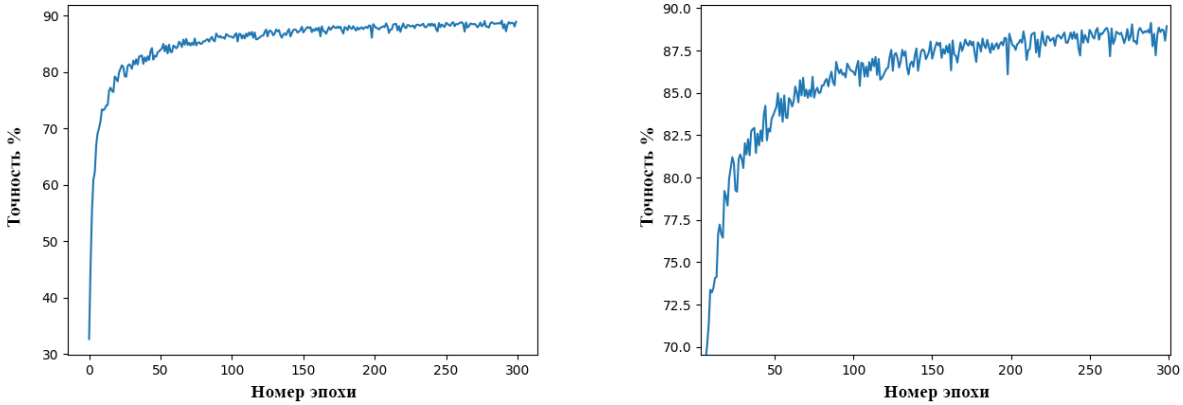


Рисунок 27 – 192, 128 нейронов

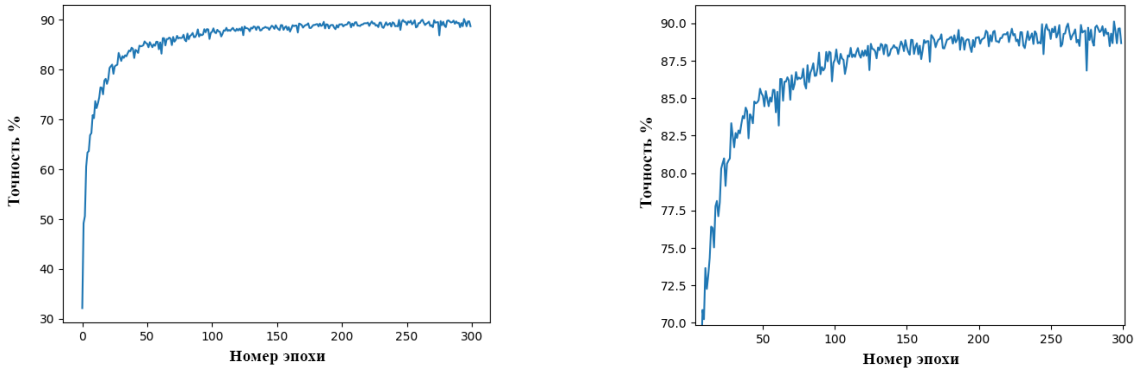


Рисунок 28 – 256, 128 нейронов

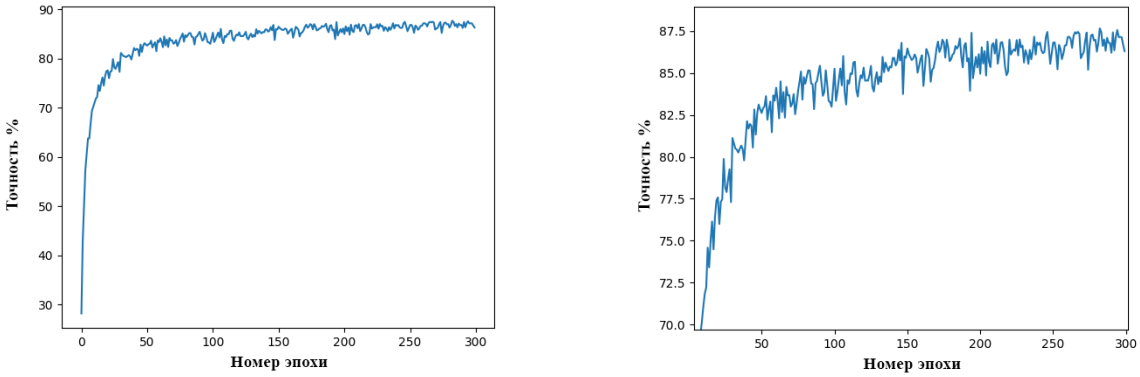


Рисунок 29 – 256, 128, 64 нейрона

Анализ результатов (Рисунок 26 - 28) показывает, что увеличение количества нейронов в скрытых слоях приводит к повышению точности обучения. Максимальная точность (90.1%) была достигнута для модели с 256 и 128 нейронами (Рисунок 28).

Как отмечалось ранее, увеличение количества слоев не всегда приводит к повышению точности. В частности, для модели с тремя скрытыми слоями (Рисунок 29) наблюдалось снижение точности до 87.67% по сравнению с аналогичной моделью с двумя скрытыми слоями (Рисунок 28).

Был проведен ещё один тест с 512 и 256 нейронами, в результате точность была немного выше, чем 90.1%, однако, обучение такой модели потребовал очень много времени, в два раза дольше чем для модели, представленной на Рисунке 27. Поэтому в дальнейшем будет рассматриваться модель, содержащая только 256 и 128 нейрона.

2.9 Регуляризация

По виду графиков (Рисунок 10 – 14, 16 – 29) можно сделать вывод, что нейронная сеть замедляется в обучении, и практически перестает обучаться с, примерно, 200 эпохи. Скорее всего значение функции стоимости застревает в локальном минимуме.

Для борьбы с застреванием в локальном минимуме применяются различные методы регуляризации, которые позволяют улучшить обобщающую способность модели при фиксированном размере сети и объеме обучающих данных. Рассмотрим метод регуляризации L2, также известный как ослабление весов. Её идея состоит в том, чтобы добавить к функции стоимости дополнительный член под названием член регуляризации. Бинарная перекрёстная энтропия с регуляризацией будет выглядеть:

$$C = -\frac{1}{n}\sum_{xj}\left[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)\right] + \frac{\lambda}{2n}\sum_w w^2 . \qquad (2.3)$$

Первый член – обычное выражение для перекрёстной энтропии. Второй член – сумма квадратов всех весов сети. Он масштабируется множителем $\lambda / 2n$, где $\lambda > 0-$ это параметр регуляризации, а n – размер обучающего набора.

В общем случае можно записать регуляризированную функцию стоимости следующим образом:

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2 , \qquad (2.4)$$

где C_0 – оригинальная функция стоимости без регуляризации.

Интуитивно ясно, что смысл регуляризации склонить сеть к предпочтению более малых весов, при прочих равных. Крупные веса будут возможны, только если они значительно улучшают первую часть функции стоимости. То есть, регуляризация – это способ выбора компромисса между нахождением малых весов и минимизацией изначальной функции стоимости. Важно, что эти два элемента компромисса зависят от значения λ: когда λ мала, мы предпочитаем минимизировать оригинальную функцию стоимости, а когда λ велика, то предпочитаем малые веса.

Применим обучающий алгоритм со стохастическим градиентным спуском к регуляризованной нейронной сети. Для этого необходимо понять, как подсчитывать частные производные, $\partial C / \partial w$ и $\partial C / \partial b$ для всех весов и смещений в сети. После взятия частных производных в уравнении (2.4) получим:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w, \qquad (2.5)$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b} . \qquad (2.6)$$

Члены $\partial C_0 / \partial w$ и $\partial C_0 / \partial b$ можно вычислить через обратное распространение, как описано в предыдущей главе. Можно заметить, что подсчитать градиент регуляризованной функции стоимости несложно, для этого нужно, использовать обратное распространение, а потом добавить $\lambda / n \cdot w$ к частной производной всех

весовых членов. Частные производные по смещениям не меняются, поэтому правило обучения градиентным спуском для смещений не отличается от обычного:

$$b \to b - \eta \frac{\partial C_0}{\partial b}. \tag{2.7}$$

Правило обучения для весов записывается:

$$w \to w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w = \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}. \tag{2.8}$$

Всё то же самое, что и в обычном правиле градиентного спуска, только сначала масштабируется вес w на множитель $1 - \eta\lambda / n$. Это масштабирование иногда называют ослаблением весов, поскольку оно уменьшает веса. Может показаться, что веса неудержимо стремятся к нулю. Однако это не так, поскольку другой член может привести к увеличению весов, если это приводит к уменьшению нерегуляризованной функции стоимости.

Регуляризованное правило обучения для стохастического градиентного спуска для весов, записывается так:

$$w \to \left(1 - \frac{\eta\lambda}{n}\right) w - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial w}, \tag{2.9}$$

где сумма идёт по обучающим примерам x в мини-пакете, а C_x – нерегуляризованная стоимость для каждого обучающего примера. То же самое, что и обычное правило стохастического градиентного спуска, за исключением члена $1 - \eta\lambda / n$ – фактора ослабления веса. Для смещений правило обновления записывается:

$$b \to b - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial b}. \tag{2.10}$$

С этого момента в программе будет использоваться регуляризованное правило обучения для стохастического градиентного спуска. Для этого выражения (1.14), (1.15) были заменены на (2.9) и (2.10) соответственно.

Для примера, были выбраны значения $\lambda = 8$ (Рисунок 30) и $\lambda = 7.6$ (Рисунок 31). Данные значения были выбраны исходя из количества обучающих данных.

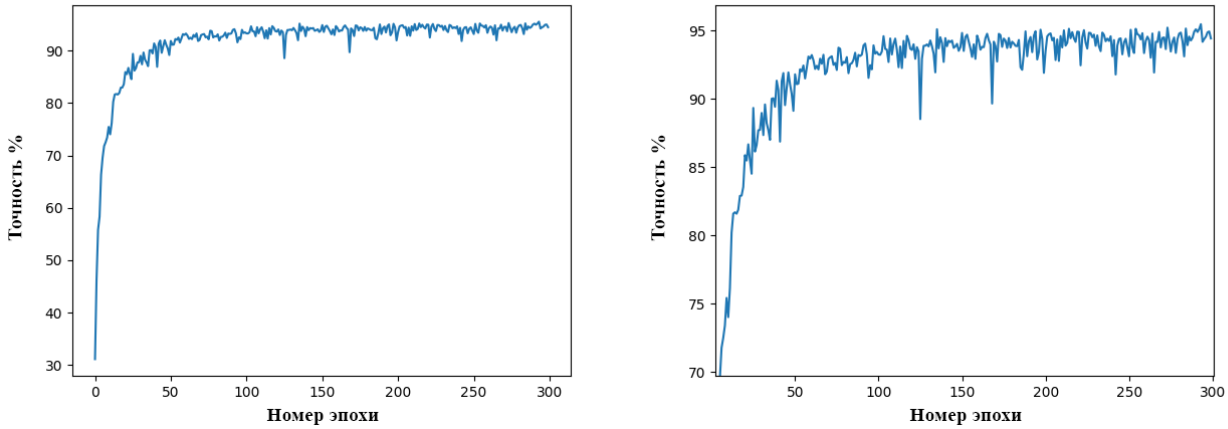


Рисунок 30 – $\lambda = 8$

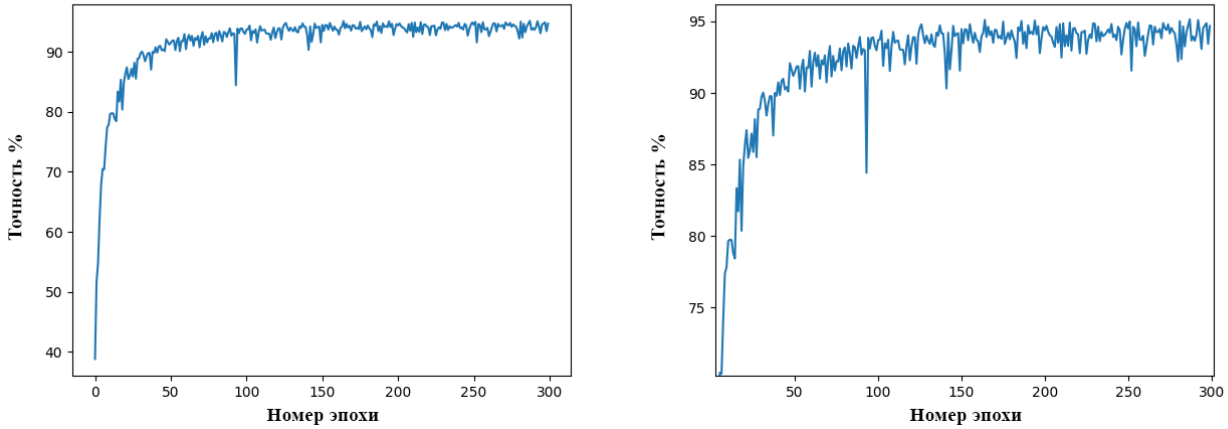


Рисунок 31 – $\lambda = 7.6$

Анализируя результаты, можно сделать вывод, что точность значительно возросла, а пиковая точность классификации достигла 95.47%, по сравнению с предыдущим пиком 90.1%, достигнутым в случае без регуляризации.

Эвристически, когда у функции стоимости нет регуляризации, размер весов в сети, скорее всего, будет расти. Со временем это может привести к очень большим значениям весов. А из-за этого вектор весов может застрять, показывая примерно в одном и том же направлении, поскольку изменения из-за градиентного спуска делают лишь ничтожно малые изменения в направлении при больших значениях весов.

2.10 Улучшение инициализации весов

В предыдущих экспериментах инициализация весов и смещений осуществлялась с помощью независимого гауссовского распределения с нулевым математическим ожиданием и единичной дисперсией. Однако, такой подход является

достаточно произвольным, что обуславливает необходимость поиска более эффективного метода инициализации.

Рассмотрим нейронную сеть с большим количеством входных нейронов (например, 1000) и проанализируем инициализацию весов, соединяющих входной слой с первым скрытым слоем, с помощью стандартного нормального распределения. На Рисунке 32 показаны веса, соединяющие входные нейроны с первым нейроном скрытого слоя:

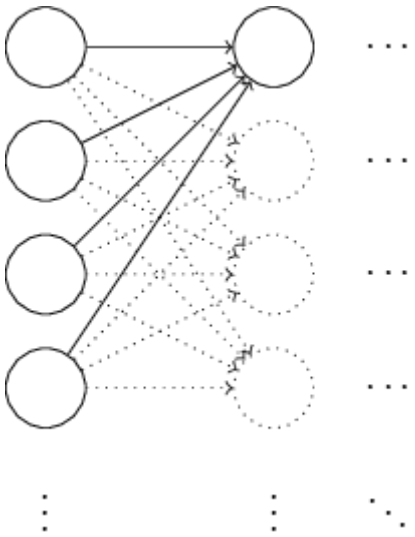


Рисунок 32 – веса, соединяющие входные нейроны с первым нейроном в скрытом слое

Предположим, что сеть будет обучаться с входом x, в котором половина входных нейронов включены, то есть, имеют значение 1, а половина – выключены, то есть, имеют значение 0. Рассмотрим взвешенную сумму $z = \sum_j w_j x_j + b$ входов для скрытого нейрона. 500 членов суммы исчезают, поскольку соответствующие x_j равны 0. Поэтому z – это сумма 501 нормализованных гауссовых случайных переменных, 500 весов и 1 дополнительное смещение. Поэтому и само значение z имеет гауссово распределение с математическим ожиданием 0 и среднеквадратичным отклонением $\sqrt{501} \approx 22,4$. То есть, у z довольно широкое гауссово распределение, без острых пиков:

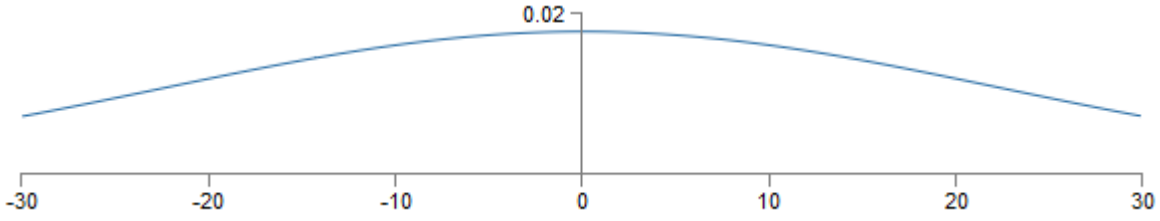


Рисунок 33 – распределение z

В частности, по Рисунку 33 видно, что $|z|$, скорее всего, будет довольно крупным, то есть, $z \gg 1$ или $z \gg -1$. В таком случае выход скрытых нейронов $\sigma(z)$ будет очень близок к 1 или 0. Это значит, что скрытый нейрон насытится. И когда это произойдёт, небольшие изменения весов будут давать малые изменения в активации скрытого нейрона. Эти малые изменения, в свою очередь, практически не затронут остальные нейроны в сети, и поэтому будут малые изменения в функции стоимости. В итоге эти веса будут обучаться очень медленно, при использовании алгоритма градиентного спуска.

Существует способ выбрать лучшие варианты инициализации для весов и смещений, чтобы не было подобного насыщения, и с помощью которого можно избежать замедления обучения. Предположим, что есть нейрон с количеством входящих весов n_{in}. Тогда эти веса нужно инициализировать случайными гауссовыми распределениями с математическим ожиданием 0 и среднеквадратичным отклонением $1/\sqrt{n_{in}}$. То есть, происходит сжатие гессианов, и уменьшение вероятности насыщения нейрона. Затем выберем гауссово распределение для смещений с математическим ожиданием 0 и среднеквадратичным отклонением 1. Сделав такой выбор, получим, что $z = \sum_j w_j x_j + b$ будет случайной переменной с гауссовым распределением с математическим ожиданием 0, однако с гораздо более выраженным пиком, чем раньше. Допустим, что 500 входов равны 0, и 500 равны 1. Тогда можно показать, что z имеет гауссово распределение с математическим ожиданием 0 и среднеквадратичным отклонением $\sqrt{(3/2)} = 1,22\ldots$. Получим график с гораздо более острым пиком:

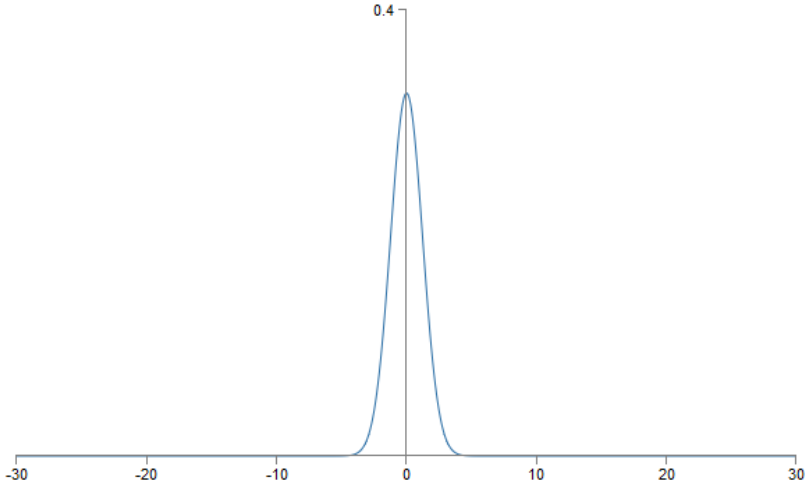


Рисунок 34 – распределение z

Такой нейрон насытится с гораздо меньшей вероятностью, и, соответственно, с меньшей вероятностью столкнётся с замедлением обучения.

В дальнейших экспериментах веса будут инициализироваться описанным выше способом, а смещения – из стандартного нормального распределения. Влияние метода инициализации смещений на результаты обучения незначительно, поскольку основной проблемой является насыщение нейронов из-за больших значений взвешенной суммы входов.

В Таблице 2 представлены результаты обучения, при подборе оптимальных гиперпараметров, таких как: размер мини-пакета m, скорость обучения η и параметр регуляризации λ. Результаты получены с учетом всех улучшений, представленных выше.

Таблица 2 – результат подбора гиперпараметров m, η, λ.

m	η	λ	Точность %
8	0.01	1.5	95.740
10	0.01	1.5	95.820
15	0.02	1.5	95.900
15	0.02	3.0	95.890
15	0.02	3.5	95.935
15	0.02	4.0	95.870
15	0.02	5.0	95.900
15	0.02	8.0	95.650
15	0.02	11.4	95.300

По сравнению с предыдущими результатами точность выросла на 5–6%, что является существенным приростом. Улучшенная инициализация весов в совокупности с подбором параметра регуляризации λ, позволили достигнуть максимальной точности классификации в 95.935%, что составляет 15829 верных распознаваний из 16500 тестовых данных. Можно сделать вывод, что относительное большое значение параметра регуляризации λ приводит к снижению точности. Самые эффективные значения λ находятся в промежутке от 1.5 до 5.

2.11 Функция активации ReLU

Функции активации в нейронных сетях используются для введения нелинейности в выходные значения нейронов. Они помогают нейронной сети обучаться и извлекать сложные зависимости между входными данными и выходными предсказаниями. До этого в нейронной сети использовался сигмоидный нейрон, а точнее, сигмодная функция (1.1). Однако, она может вызывать проблему затухания градиента при обучении глубоких нейронных сетей.

Чтобы избежать этого можно использовать функцию ReLU (Rectified Linear Unit Activation), которая преобразует отрицательные значения в 0, а положительные значения оставляет без изменений. Эта функция хорошо подходит для распознавания текста, так как она помогает избежать проблемы затухания градиента и способствует быстрой сходимости при обучении. Она имеет вид:

$$f(x) = \max(0, x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.11}$$

Производная функции активации ReLU, необходимая для вычисления ошибки, имеет следующий вид:

$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.12}$$

Результаты обучения нейронной сети с функцией активации ReLU представлены в Таблице 3.

Таблица 3 – результаты подбора гиперпараметров m, η и λ, при использовании функции активации ReLU.

η	λ	Точность %
0.02	3.50	95.820
0.01	5.00	95.900
0.02	1.00	95.000
0.01	0.00	95.400
0.01	1.00	95.380
0.02	0.00	94.820
0.01	2.00	95.666
0.01	4.00	95.575
0.01	8.00	96.200
0.01	16.00	96.340
0.01	12.00	96.260
0.01	32.00	96.100
0.01	24.00	96.260
0.01	20.00	96.545
0.01	18.00	96.224
0.01	22.00	96.164
0.01	19.00	95.940

По полученным данным можно сделать вывод, что по сравнению с сигмоидной функцией активации точность обучения возросла примерно на 0.6%. Максимальная точность была достигнута при следующих значениях гиперпараметров: $m=15$, $\eta=0.01$, $\lambda=20$. Она составила 96.545%. График обучения с данными гиперпараметрами выглядит следующим образом:

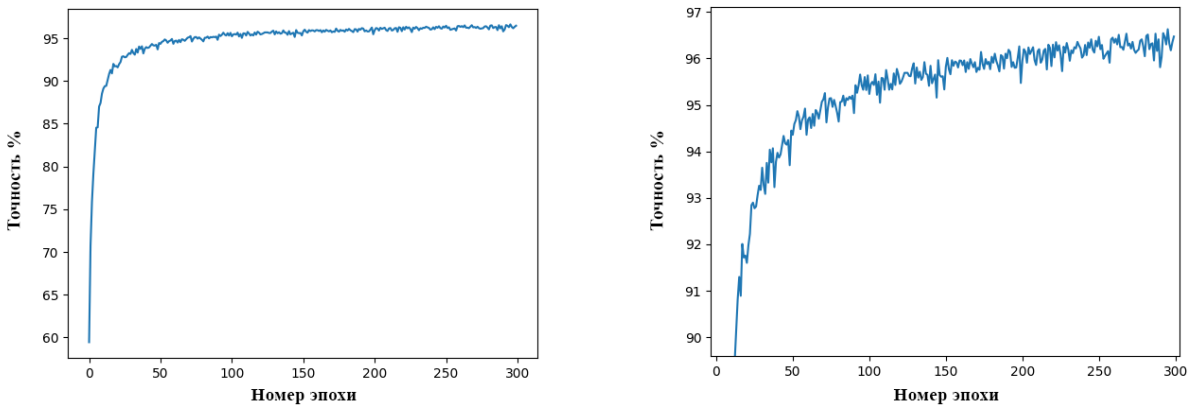


Рисунок 35 – $m=15$, $\eta=0.01$, $\lambda=20$

Использование ReLU позволило увеличить точность распознавания примерно на 0.6% по сравнению с сигмоидной функцией активации. По графику Рисунка 35 можно заметить значительное ускорение обучения на начальных этапах, что связано с отсутствием затухания градиента.

3 Тестирование нейронной сети

3.1 Квантизация

Квантизация модели нейронной сети – это метод оптимизации, который позволяет уменьшить размер модели и ускорить ее выполнение, особенно на устройствах, имеющих низкую производительность, таких как мобильные устройства или встраиваемые системы (микрокомпьютеры). Квантизацию используют по следующим взаимосвязанным причинам:

- Уменьшение размера модели: квантованные модели занимают меньше места в памяти, что особенно важно для мобильных устройств и встраиваемых систем, обладающих слабыми вычислительными ресурсами.
- Ускорение инференса: вычисления с использованием типов данных меньшей разрядности (например, float16 вместо float32) выполняются быстрее, что ускоряет процесс инференса.
- Снижение энергопотребления: более быстрые вычисления и меньший объем используемой памяти приводят к снижению энергопотребления, что также важно для мобильных и встраиваемых устройств

Как правило, квантизация нейронной сети включает в себя следующие шаги:

- Обучение модели: модель нейронной сети обучается как обычно, используя полную точность (например, float32).
- Выбор метода квантизации: существуют различные методы квантизации, такие как:
    - Посттренировочная квантизация: применяется к уже обученной модели. Этот метод проще в реализации, но может привести к большей потере точности.
    - Квантовое обучение: модель обучается с использованием квантованных весов и активаций. Этот метод сложнее, но может обеспечить лучшую точность после квантизации.

- Преобразование модели: веса, смещения и, возможно, активации нейронов преобразуются из типа данных с высокой точностью в тип данных с низкой точностью.
- Тестирование и оптимизация: Квантованная модель тестируется и оптимизируется для достижения баланса между точностью и производительностью.

До этого момента использовалась модель нейронной сети, использующая тип данных float32 для представления весов и смещений. Это стандартный подход, обеспечивающий высокую точность вычислений во время обучения и инференса – процесса использования обученной модели для прогнозирования на новых входных данных.

В данной работе для ускорения инференса нейронной сети классификации рукописных букв была применена, самая простая посттренировочная квантизация. Веса и смещения модели были преобразованы из типа данных float32 в тип данных float16.

Для оценки влияния квантизации на производительность модели было проведено сравнение времени инференса для исходной модели, обозначим её FP32, и квантованной модели, обозначим её FP16. Тестирование проводилось на 10000 изображениях на следующих устройствах:

- Ноутбук Lenovo Ideapad;
- Ноутбук Apple MacBook Pro;
- Микрокомпьютер Raspberry Pi 4;
- Персональный компьютер №1 (далее ПК1);
- Персональный компьютер №2 (далее ПК2);

Основные характеристики тестируемых устройств представлены в Таблице 4. Важно отметить, что для всех запусков использовался стандартный компилятор CPython без привлечения каких-либо дополнительных библиотек или фреймворков для ускорения вычислений. Это позволяет обеспечить воспроизводимость результатов на любой платформе с установленным интерпретатором Python.

Таблица 4 – характеристики тестируемых устройств.

	Lenovo Ideapad	Raspberry Pi 4	MacBook Pro	ПК1	ПК2
	Центральный процессор				
Модель	Intel Core i3-5005U	Broadcom BCM2711	Apple M1	AMD Ryzen 7 PRO 3700	Intel Core i7-10700KF
Архитектура	x86	ARMv8-A	ARMv8.5-A	x84	x86
Разрядность, бит	64	64	64	64	64
Микроархитектура	Broadwell-U	Cortex-A72	Firestorm / Icestorm	Zen2	Comet Lake
Количество ядер, шт.	2	4	8	8	8
Тактовая частота, ГГц	2.0	1.5	3.2	3.6	3.8
Объем L3-кэш, Мб	3	1	16	32	16
	Оперативная память				
Тип	DDR3	LPDDR4	LPDDR4X	DDR 4	DDR 4
Объём, Гб	8	2	16	64	64
Частота, МГц	1995	3200	4266	2666	3200
	Дисковая память				
Тип	HDD	microSD	SSD (M2)	SSD (M2)	SSD (M2)
Объём, Гб	1024	16	1024	500	500
Скорость чтения, Мб/с	300	100	3000	550	550
Скорость записи, Мб/с	300	100	2700	520	520
	Операционная система				
Семейство	Windows	Linux	MacOS	Windows	Windows
Версия	10 Pro	Debian Bullseye 5.15.32-v8+	14.5	10 Pro	10 Pro

Таблица 5 – Результат времени выполнения инференса модели FP32, для устройств различного класса производительности.

Устройство	Время (сек.)	Точность %
Lenovo Ideapad	2.7978	96.23
Raspberry Pi 4	9.6059	96.23
MacBook Pro	0.6824	96.23
ПК1	1.8161	96.23
ПК2	1.4610	96.23

Таблица 6 – Результат времени выполнения инференса модели FP16, для устройств различного класса производительности.

Устройство	Время сек.	Точность %
Lenovo Ideapad	2,6111	96.19
Raspberry Pi 4	8.7375	96.19
MacBook Pro	0.6757	96.19
ПК1	1.7865	96.19
ПК2	1.4570	96.19

Анализ результатов тестирования (Таблица 5 и Таблица 6) позволяет сделать вывод, что квантизация модели привела к ускорению инференса на всех тестируемых устройствах, хотя степень ускорения варьируется в зависимости от платформы. Наиболее заметное ускорение наблюдалось на микрокомпьютере Raspberry Pi 4 (9%) и ноутбуке Lenovo Ideapad (6.6%). На более производительных устройствах (MacBook Pro, ПК1, ПК2) эффект ускорения был менее выраженным (менее 1%).

Квантизация модели привела к небольшому снижению точности классификации на 0.04%. Такое снижение является несущественным для большинства практических приложений и компенсируется выигрышем в производительности.

Результаты демонстрируют зависимость эффективности квантизации от аппаратной платформы. На устройствах, обладающих слабыми вычислительными ресурсами (например, Raspberry Pi 4), квантизация даёт более значительное ускорение.

Таким образом, квантизация модели является эффективным методом оптимизации, позволяющим ускорить инференс без существенной потери точности. Выбор между исходной и квантованной моделью зависит от конкретного приложения и требований к производительности и точности.

Заключение

В работе были исследованы основные виды нейросетей, их принципы работы, алгоритм обучения с учителем, некоторые эффективные методы борьбы с проблемой замедлением обучения, такие как, функция стоимости с бинарной перекрестной энтропией и регуляризация.

Была создана многослойная нейронная сеть для классификации рукописных букв русского алфавита. В процессе разработки не использовались специализированные фреймворки, что позволило глубоко понять принципы работы нейронных сетей и алгоритмы их обучения.

Сеть была обучена на датасете, который был сформирован на основе неподготовленных данных из архива CoMNIST, что позволило понять аспекты подготовки и создания датасета и особенностей работы с ним. Для обучения нейронной сети использовался тип обучения с учителем.

В ходе исследования точность классификации моделью тренировочных данных была повышена с 70% до 96.5%. Для этого потребовалось выполнить подбор оптимальных гиперпараметров сети, количество скрытых слоев и нейронов на них, исследовать функцию стоимости с бинарной перекрёстной энтропией, регуляризацию L2, функцию активации ReLU, функцию softmax и улучшить способ инициализации весов.

Исследование показало, что самая простая посттренировочная квантизация позволяет добиться ускорения инференса на всех тестируемых устройствах без существенной потери точности, а наибольшее ускорение наблюдалось на устройствах, обладающих слабыми вычислительными ресурсами. Из этого следует, что квантизация является эффективным методом оптимизации нейронных сетей для задачи классификации рукописных букв, позволяющим адаптировать модель для работы на устройствах с различными характеристиками.

Список литературы

1. Аппаратное обеспечение для ИИ // Альманах Искусственный Интеллект. Аналитический сборник. 2021. №9. URL: https://ai.gov.ru/en/knowledgebase/razrabotka-i-issledovaniya-v-oblasti-ii/2021_alymanah_ii_9_apparatnoe_obespechenie_ii_mfti/
2. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с анг. А. А. Слинкина. — 2-е изд., испр. — М.: ДМК Пресс, 2018. — 652 с.
3. Мюллер А. Введение в машинное обучение с помощью Python. — М.: ДМК Пресс, 2019. — 472 с.
4. Николенко С. И. , Кадурин А. А., Архангельская Е. О. Глубокое обучение. — СПб.: Питер, 2018. — 480 с.
5. Рашид, Тарик. Создаём нейронную сеть.: Пер. с англ. — СПб.: ООО "Диалектика", 2019. — 272 с.
6. Хайкин, Саймон Нейронные сети полный курс, 2-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2006. — 1104 с.
7. Шолле Франсуа Ш78 Глубокое обучение на Python. 2-е межд. издание. — СПб.: Питер, 2023. — 576 с.
8. Ciresan D.C., Meier U., Schmidhuber J. Multi-column deep neural networks for image classification. – 2012. – arXiv:1202.2745.
9. Sharma, S., Sharma, S. and Athaiya, A., 2017. Activation functions in neural networks. Towards Data Sci, 6(12), pp.310-316.
10. Ахметзянов К.Р., Тур А.И., Кокоулин А.Н., Южаков А.А. ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ НЕЙРОННОЙ СЕТИ // Вестник ПНИПУ. Электротехника, информационные технологии, системы управления. 2020. №36. URL: https://cyberleninka.ru/article/n/optimizatsiya-vychisleniy-neyronnoy-seti
11. Фаустова К.И. Нейронные сети: применение сегодня и перспективы развития // Территория науки. 2017. №4. URL: https://cyberleninka.ru/article/n/neyronnye-seti-primenenie-segodnya-i-perspektivy-razvitiya
12. Michael Nielsen, neural network and deep learning URL: http://neuralnetworksanddeeplearning.com/index.html