МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет компьютерных наук

Кафедра цифровых технологий

Классификация с помощью нейронных сетей и квантовых алгоритмов

ВКР Бакалаврская работа

02.03.01 Математика и компьютерные науки

Распределенные системы и искусственный интеллект

Допущено к защите в ГЭК __.__.20__

Зав. кафедрой _____ *С. Д. Кургалин, д. ф.-м. н., профессор*
подпись

Обучающийся _____ *И. В. Петров, 4 курс, д/о*
подпись

Руководитель _____ *А. Ф. Клинских, д. ф.-м. н, профессор*
подпись *расшифровка подписи, ученая степень, звание, должность*

Воронеж 2021

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет компьютерных наук

Кафедра цифровых технологий

УТВЕРЖДАЮ
заведующий кафедрой

_____ _____
подпись, расшифровка подписи
__.__.20__

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
ОБУЧАЮЩЕГОСЯ ПЕТРОВА ИЛЬИ ВЛАДИМИРОВИЧА
фамилия, имя, отчество

1. Тема работы «Классификация с помощью нейронных сетей и квантовых алгоритмов», утверждена решением ученого совета факультета компьютерных наук от __ .__.20__

2. Направление подготовки / специальность 02.03.01 Математика и компьютерные науки

3. Срок сдачи законченной работы 11.06.2021

4. Календарный план: (строится в соответствии со структурой ВКР)

№	Структура ВКР	Сроки выполнения	Примечание
1	Введение	01.10.2020-03.10.2020	
2	Нейронные сети	03.10.2020-01.02.2021	
3	Квантовые вычисления	01.02.2021-20.04.2021	
4	Программная реализация классического и квантового подхода для задачи классификации	20.04.2021-05.06.2021	
5	Заключение	05.06.2021-06.06.2021	
6	Список использованных источников	06.06.2021-08.06.2021	

Обучающийся _____ _____
Подпись *расшифровка подписи*

Руководитель _____ _____
Подпись *расшифровка подписи*

РЕФЕРАТ

Реферат Бакалаврская работа с. 48, рис.13, табл.1

МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, КВАНТОВЫЕ АЛГОРИТМЫ, КЛАССИФИКАЦИЯ

Объектом исследования является изучение классических и квантовых алгоритмов для классификации

Цель работы – решение задачи классификации при помощи квантовых алгоритмов и нейронных сетей, а именно определения класса точек по их координатам

В результате исследования были реализованы методы классификации с помощью нейронных сетей и квантовых алгоритмов, а также было произведено сравнение двух алгоритмов.

СОДЕРЖАНИЕ

Введение

Одной из самых развивающихся областей в сфере IT являются нейронные сети. Несмотря на то, что первые попытки разработать математическую модель работы человеческого мозга датируются серединой 20-ого века, нейросети до сих пор являются актуальным решением многих серьезных задач. Без них сейчас невозможно представить работу с распознаванием текста, голоса и изображений, а также анализ данных различного типа. Но есть отрасль, которая уже сейчас готова составить конкуренцию нейронной сети – квантовые алгоритмы. В данной работе я подробно рассмотрю и сравню эти два вида обучения на примере классификации данных.

1 Нейронные сети

1.1 Искусственные нейронные сети и строение нейрона

Исследования в области искусственных нейронных сетей пережили несколько периодов активизации.

Первый период был в 1943 году, когда нейрофизиолог Уоррен Мак-Каллох и математик Уолтер Питтс написали статью о работе нейронов. Для описания работы нейронов мозга, они смоделировали простую НС, с использованием электрической цепи [1]. Их основная идея заключалась в том, что любая связь типа "вход-выход" может быть реализована искусственной (формальной) НС.

Вторым этапом в развитии НС стало изобретение перцептрона в 1957 году Фрэнком Розенблаттом. В 1958 г. Фрэнк Розенблатт продемонстрировал компьютерную модель электронного устройства, названную им перцептроном, а в 1960 г. – первый действующий нейрокомпьютер «Марк-1», который моделировал совместную работу человеческого глаза и мозга[2]. Основным назначением машины было распознавание. Концом этого этапа стала публикация Марвина Ли Минского и Сеймура Пайперта в 1969 году, в которой они указали на важный класс задач, которые однослойный персептрон решать не может. Минский и Пайперт дискредитировали исследования НС и финансирование в области искусственного интеллекта. Но несмотря на демонстрацию Минским и Пайпертом ограничений перцептронов, исследования нейронной сети все же продолжались.

В 1982 году Джон Хопфилд – физик с мировым именем, заинтересовавшись нейронными сетями, написал две особо читаемые статьи о НС и провел многочисленные лекции по всему миру, чем убедил сотни высококвалифицированных ученых, математиков и технологов присоединиться к формирующемуся полю НС. Хопфилд показал, что высокосвязная сеть нейронов с обратными связями может быть описана как динамическая система, обладающая "энергией". При ассоциативном вызове сеть, стартующая в произвольном (случайном) состоянии, сходится к

конечному устойчивому состоянию с наименьшей энергией. Новый подход к описанию НС с обратными связями оказался очень плодотворным.

Подобный прорыв произошел и в связи с многослойными сетями без обратных связей. Для обучения таких сетей был разработан алгоритм обратного распространения ошибки.

С середины 80-х годов теория нейронных сетей получила «технологический импульс», вызванный появлением новых доступных и высокопроизводительных персональных компьютеров.

Самых значительных достижений в данном вопросе достигла американская компания IBM (англ. Internationa Business Machines). Первые результаты исследований были продемонстрированы 14 ноября 2009 года. Компания представила на суд общественности успешно смоделированный мозг кошки. Правда, следует отметить, что тогда его работа была в 643 раза медленнее реального времени.

Следующей вехой в развитии ИНС можно считать 18 августа 2011 года, когда IBM создали передовой на тот момент нейронный процессор, который содержал 256 нейронов и 262144 синапсов [3].

Искусственные нейронные сети являются относительно грубыми электронными моделями, основанными на нервной структуре головного мозга. Фундаментальным обрабатывающим элементом нейронной сети является нейрон. Нейрон (нервная клетка) — это специальная биологическая клетка, которая обрабатывает информацию. По данным оценки, в мозге существует огромное количество нейронов, каждый из которых обладает примерно 10^{11} многочисленными взаимосвязями.

На рисунке 1.1 представлена схема биологического нейрона.

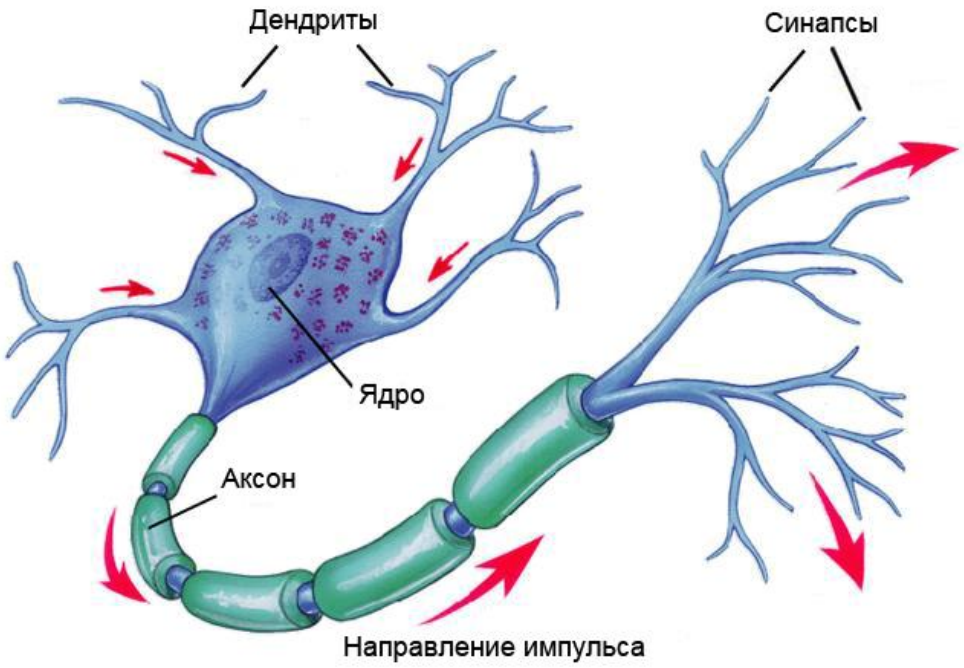

Рисунок 1.1 – Биологический нейрон

Как показано на приведенной выше схеме, типичный нейрон состоит из следующих четырех частей, с помощью которых мы можем объяснить его работу:

- Дендриты — древоподобные ветвей, отвечающий за получение информации от других нейронов, к которым подключен данный. В другом смысле, мы можем сказать, что это «уши» нейрона;
- Ядро — это тело клетки нейрона, отвечающее за обработку информации, полученной от дендритов;
- Аксон — некий «кабель», посредством которого нейроны посылают информацию;
- Синапсы — это соединения между аксоном и дендритами других нейронов.

Последние экспериментальные данные предоставили дополнительные доказательства того, что нейроны структурно более сложно устроены, чем упрощенно описано выше. Они значительно более сложны, чем существующие искусственные нейроны, которые встроены в сегодняшние искусственные нейронные сети. По мере того, как биология обеспечивает

лучшее понимание нейронов, и по мере развития технологий, разработчики сетей могут продолжать совершенствовать свои системы, опираясь на понимание человеческого мозга.

Но в настоящее время грандиозное воссоздание мозга – не является целью искусственных нейронных сетей. Наоборот, нейросетевые исследователи ищут способы применения возможностей природы, для которых люди могут спроектировать решения проблем, которые не были решены с помощью традиционных вычислений.

Для этого нужна основная единица нейронных сетей – искусственные нейроны, моделирующие четыре основные функции биологических нейронов. Рисунок 1.2 показывает фундаментальное представление искусственного нейрона – перцептрона [4-9].

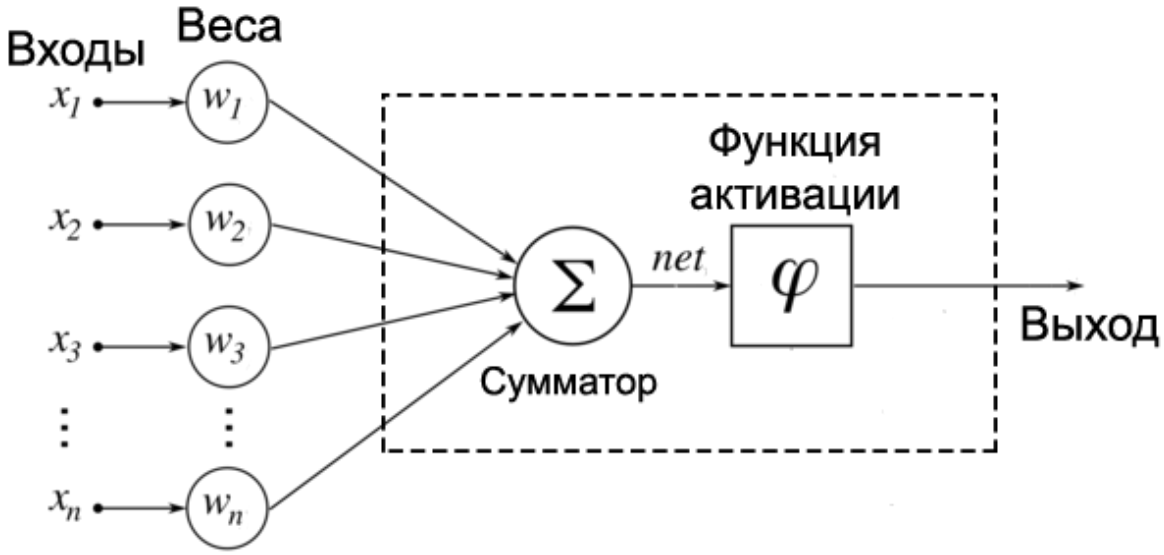

Рисунок 1.2 – Искусственный нейрон

Соотнесем известное нам понятие о биологическом нейроне с понятиями, характерными для персептрона. Данные приведены в следующей таблице:

Таблица 1.1 – Сравнительная характеристика понятий БНС и ИНС

Биологическая нейронная сеть	Искусственная нейронная сеть
Ядро	Узел
Дендриты	Входные данные
Синапсы	Веса
Аксон	Выходные данные

1.2 Сети прямого распространения

Линейные модели для регрессии и классификации имеют вид:

$$y(\mathrm{x}, \mathrm{w}) = f\left(\sum_{j=1}^{M} w_j \, \varphi_j(\mathrm{x})\right) \tag{1.1}$$

Где $f(\cdot)$ нелинейная функция активации в случае классификации и является тождественной в случае регрессии. Наша цель расширить эту модель путем представления базисных функций $\varphi_j(\mathrm{x})$, которые будут зависеть от параметров, и потом разрешить этим параметрам изменяться одновременно с коэффициентами $\{w_j\}$ во время испытаний. Существуют, конечно же, много путей построить параметрические нелинейные базисные функции. Нейронные сети используют базисные функции, которые выглядят так же как (1.1), и поэтому каждая базисная функция и есть нелинейная функция от линейной комбинации входных данных, где коэффициенты в линейной комбинации являются адаптивными параметрами.

Это приводит нас к основной нейросетевой модели, которая может быть описана как набор функциональных преобразований. Сначала мы строим M линейных комбинаций входных переменных $x_1, \ldots, x_D$ в виде

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{1.2}$$

где $j = 1, \dots, M$ и верхний индекс (1) означает, что соответствующие параметры на первом «слое» сети. Мы должны обозначить $w_{ji}^{(1)}$ как «*веса*» и параметры $w_{j0}^{(1)}$ как *«погрешности*». Величины a_j известны как «*активации»*. Каждая из них потом преобразуется с помощью дифференцируемой, нелинейной «функции активации» $h(\cdot)$, чтобы получить

$$z_j = h(a_j) \tag{1.3}$$

Эти величины соответствуют выходным результатам базисных функций в (1.1), в контексте нейронных сетей называются «скрытые юниты». Нелинейные функции $h(\cdot)$, как правило, являются сигмоидными (или «tanh») функциями. Исходя из (1.1), эти величины снова линейно комбинируются, чтобы дать *выходные единицы активации.*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_i + w_{k0}^{(2)} \tag{1.4}$$

где $k = 1, \dots, K$ и K – общее количество выходных результатов. Преобразования соответствуют второму слою сети, и снова $w_{k0}^{(2)}$ - параметры погрешности. Наконец, выходные единицы активации преобразуются, используя соответствующую функцию активации, чтобы дать набор сетевых выходных данных y_k. Выбор функции активации определяется характером данных и предполагаемым распределением искомых переменных и следуют тем же аспектам, что и для линейных моделей. Таким образом, для стандартных регрессионных проблем функция активации имеет тот же вид, так что $y_k = a_k$.

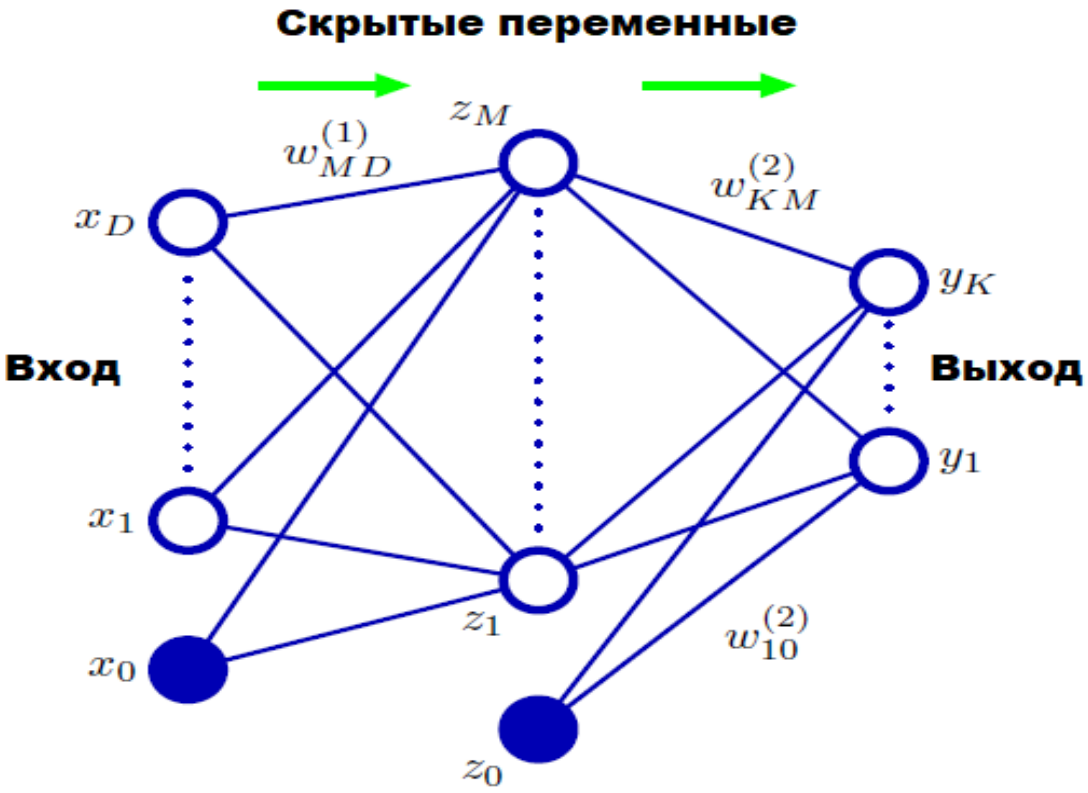

Рисунок 1.3 – Двухуровневая нейронная сеть

Точно также для многочисленных бинарных проблем классификации каждый выходной юнит активации преобразуется, используя логистическую сигмоидную функцию так, что

$$y_k = \sigma(a_k) \tag{1.5}$$

где

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{1.6}$$

Мы можем скомбинировать эти различные этапы, чтобы получить общую сетевую функцию, которая примет следующий вид:

$$y_k(\mathrm{x},\mathrm{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)}\, h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \qquad (1.7)$$

где набор всех весов и параметров погрешности сгруппирован вместе в вектор w. Таким образом, модель нейросети просто нелинейная функция, которая состоит из набора входных переменных $\{x_i\}$ и набором выходных переменных $\{y_k\}$ регулируемая вектором **w**, вектором корректируемых параметров.

Эта функция может быть представлена в форме сетевой диаграммы, представленной на рисунке 1.3. Процесс оценки (1.7) может быть рассмотрен как *прямое распространение* информации через сеть.

Параметры погрешности в (1.2) могут быть включены в набор весовых параметров путем определения дополнительной входной переменной x_0, для которой мы закрепим значение $x_0 = 1$, таким образом, (1.2) примет вид:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad (1.8)$$

Мы можем похожим образом превратить второуровневые погрешности во второуровневые веса так, что окончательная сетевая функция примет вид:

$$y_k(\mathrm{x},\mathrm{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \qquad (1.9)$$

Как было замечено из рисунка 1.3, нейросетевая модель содержит две этапа, каждый из которых напоминает «персептрон» (математическая или компьютерная модель восприятия информации мозгом) и по этой причине нейронная сеть также известна как *многослойный персептрон (MLP).* Ключевое различие в сравнение с персептроном, впрочем, это то, что нейронная сеть использует непрерывные сигмоидальные нелинейности в скрытых данных, в то время как персептрон использует кусочно-постоянные нелинейности. Это означает, что нейронная сетевая функция

дифференцируема в отношении к сетевым параметрам, и это свойство будет играть главную роль в сетевом обучении.

Архитектура сетей, показанная на картинке 1.3, чаще всего используется на практике. Однако она легко обобщается в пример, который включает в себе дополнительные уровни обработки, каждый из которых состоит из взвешенной линейной комбинации вида (1.4) с последующим поэлементным преобразованием с использованием нелинейной функции активации. Отметим, что бывает некоторая путаница в литературе относительно терминологии для расчета числа уровней в подобных сетях. Таким образом, сеть на рисунке 1.3 может быть описана как 3-ех уровневая сеть. Рекомендуется терминология, в которой рисунок 1.3 называется двухуровневой сетью [10], потому что число уровней адаптивных весов важно для определения сетевых свойств.

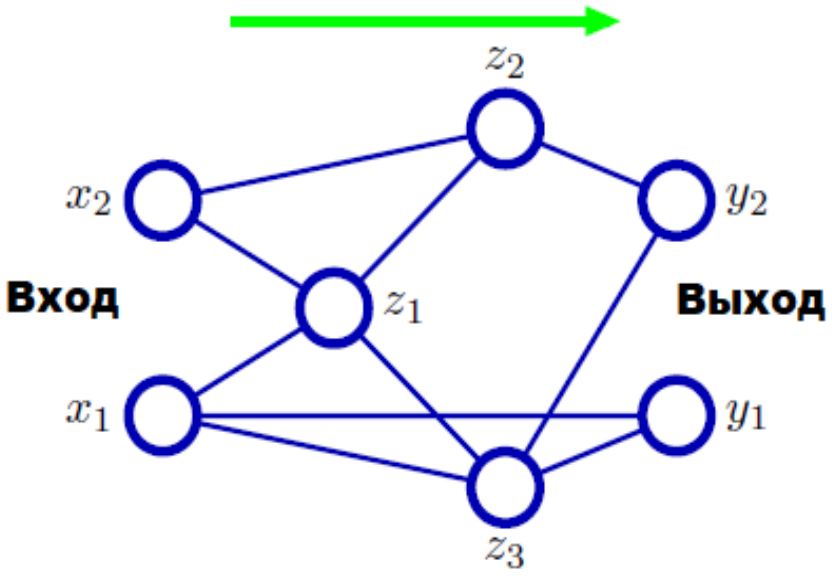

Рисунок 1.4 – Нейронная сеть с топологией

Из-за того, что есть прямое соответствие между сетевой диаграммой и ее математической функции, мы можем разработать более общие сетевые

преобразования путем рассмотрения более сложных сетевых диаграмм. Однако данные должны быть ограничены архитектурой прямого распространения («feed forward»), другими словами ограничены архитектурой, у которой нет замкнутых направленных циклов. Это продемонстрировано на простом примере в рисунке 1.4. Каждый (выходной или выходной) юнит в такой сети вычисляется следующей функцией:

$$z_k = h\left(\sum_j w_{kj} z_j\right) \tag{1.10}$$

Возможность двуслойной сети моделировать разнообразный диапазон функций проиллюстрирована на рисунке 1.5. Эта картинка также демонстрирует, как отдельные скрытые юниты работают совместно, чтобы аппроксимировать окончательную функцию.

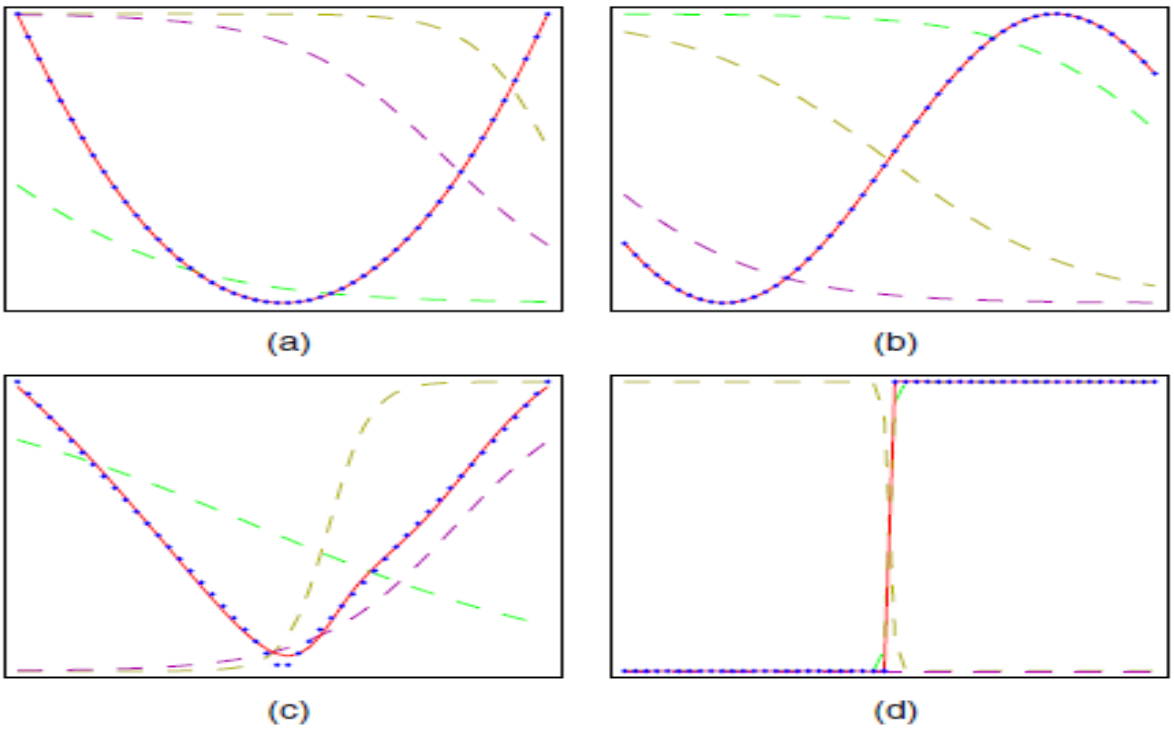

Рисунок 1.5 – Иллюстрация способности многослойного персептрона аппроксимировать четыре различные функции

1.3 Сетевое обучение

Мы рассмотрели нейронные сети как общий класс параметрических нелинейных функций от вектора **x** входных переменных и вектора **y** выходных переменных. Простой подход к проблеме определения сетевых параметров — это минимизировать сумму квадратов функции ошибки. Учитывая тренировочный набор, содержащий набор входных векторов $\{x_n\}$, где $n = 1, \dots, N$ вместе с содержащим набором целевых векторов $\{t_n\}$, мы должны минимизировать функцию ошибки

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N} \|y(\mathrm{x}_n, \mathrm{w}) - t_n\|^2 \qquad (1.11)$$

Тем не менее, мы можем предоставить гораздо более общий взгляд на обучение сети, сначала дав вероятностную интерпретацию выходам сети. Это также даст нам более четкое понимание при выборе функции ошибок.

Начнем с обсуждения проблем регрессии, первым шагом рассмотрим целевую переменную *t*, которая может принимать любое действительное значение. Мы предполагаем, что *t* имеет гауссово распределение с зависимым средним x, который получается на выходе нейронной сети, так что

$$p(t|\mathrm{x}, \mathrm{w}) = \mathcal{N}(t|y(\mathrm{x}, \mathrm{w}), \beta^{-1}) \qquad (1.12)$$

где β - точность (обратная дисперсия) Гауссовского шума. Конечно, это несколько ограничительное предположение. Для условного распределения (1.12), достаточно взять тождественную функцию активации, потому что такая сеть может аппроксимировать любую непрерывную функцию от x до *y*. Учитывая набор данных из N независимых, одинаково распределенных наблюдений $\mathbf{X} = \{\mathrm{x}_1, \dots, \mathrm{x}_N\}$ вместе с соответствующими целевыми значениями $\mathbf{t} = \{\mathrm{t}_1, \dots, \mathrm{t}_N\}$, мы можем построить соответствующую функцию правдоподобия

$$p(\mathbf{t}|\mathbf{X}, \mathrm{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathrm{x}_n, \mathrm{w}, \beta) \tag{1.13}$$

Взяв отрицательный логарифм, получим функцию ошибок

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathrm{x}_n, \mathrm{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{1.14}$$

которую можно использовать для изучения параметров w и β. Здесь мы рассматриваем подход максимального правдоподобия. Стоит обратить внимание на то, что обычно рассматривается минимизация функции ошибок, а не максимизация (*log*) вероятности, и поэтому здесь мы будем следовать этому соглашению. Рассмотрим сначала решение w. Максимизация функции правдоподобия эквивалентна минимизации функция ошибки суммы квадратов, которая определяется как

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathrm{x}_n, \mathrm{w}) - t_n\}^2 \tag{1.15}$$

где мы отбросили аддитивные и мультипликативные константы. Значение w, найденное путем минимизации $E(\mathrm{w})$, мы будем обозначать w_{ML}, потому что он соответствует максимально вероятному решению. На практике нелинейность сетевой функции $y(\mathrm{x}_n, \mathrm{w})$ приводит к невыпуклости ошибки $E(\mathrm{w})$, и поэтому на практике могут быть найдены локальные максимумы вероятности, соответствующие локальным минимумам функции ошибок. Найдя w_{ML}, значение β можно найти путем минимизации отрицательный логарифм от вероятности, чтобы получить

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\mathrm{x}_n, \mathrm{w}_{ML}) - t_n\}^2 \tag{1.16}$$

Обратите внимание, что это можно оценить после завершения итеративной оптимизации, необходимой для поиска w_{ML}. Если у нас есть несколько целевых переменных, и мы предполагаем, что они являются условно независимыми от x

и w с точностью шума β, то условное распределение целевых значений задается следующим образом:

$$p(\mathrm{t}|\mathrm{x},\mathrm{w}) = \mathcal{N}(\mathrm{t}|y(\mathrm{x},\mathrm{w}),\beta^{-1}\mathbf{I}) \tag{1.17}$$

Следуя тем же аргументам, что и для одной целевой переменной, мы видим, что максимальные веса правдоподобия определяются путем минимизации функции ошибок суммы квадратов.

Тогда точность шума определяется выражением

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}\|y(\mathrm{x}_n,\mathrm{w}_{ML}) - \mathrm{t}_n\|^2 \tag{1.18}$$

где K - количество целевых переменных. От предположения о независимости можно отказаться за счет немного более сложной задачи оптимизации. В случае регрессии мы можем рассматривать сеть как имеющую выходную функцию активации, которая является тождественной, так что $y_k = a_k$. Соответствующая функция ошибок суммы квадратов обладает свойством

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{1.19}$$

которое мы будем использовать при дальнейшем обсуждении обратного распространения ошибок. Теперь рассмотрим случай бинарной классификации, в которой у нас есть одна целевая переменная t такая, что $t = 1$ обозначает класс C_1, а $t = 0$ обозначает класс C_2. Мы рассматриваем сеть с одним выходом, функция активации которой является логистическая сигмоида

$$y = \sigma(a) \equiv \frac{1}{1+\exp(-a)} \tag{1.20}$$

так что $0 \leq y(\mathrm{x},\mathrm{w}) \leq 1$. Мы можем интерпретировать $y(\mathrm{x},\mathrm{w})$ как условную вероятность $p(C_1|x)$, где $p(C_2|x)$ задается как $1 - y(\mathrm{x},\mathrm{w})$. Условное распределение целевых данных при заданных входных данных в таком случае является распределением Бернулли следующей формы

$$p(t|\mathrm{x},\mathrm{w}) = y(\mathrm{x},\mathrm{w})^t\{1 - y(\mathrm{x},\mathrm{w})\}^{1-t} \tag{1.21}$$

Если мы рассмотрим обучающий набор независимых наблюдений, то функция ошибок, которая задается отрицательным логарифмическим правдоподобием, будет *кросс-энтропийной* функцией ошибок вида

$$E(\mathrm{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} \tag{1.22}$$

где y_n эквивалентно $y(\mathrm{x}_n, \mathrm{w})$. Обратите внимание, что нет аналога точности шума β, поскольку предполагается, что целевые значения правильно помечены. Однако модель легко расширяется, чтобы учесть ошибки маркировки. Было обнаружено, что использование кросс-энтропийной функции ошибки вместо суммы квадратов для задачи классификации приводит к более быстрому обучению, а также к улучшенному обобщению.

Если у нас есть K отдельных бинарных классификаций, то мы можем использовать сеть, имеющую K выходов, каждый из которых имеет логистическую функцию активации. С каждым выходом связана метка двоичного класса $t_k \in \{0, 1\}$, где $k = 1, \ldots, K$. Если мы предположим, что метки классов независимы при заданном входном векторе, то условное распределение целей будет

$$p(\mathrm{t}|\mathrm{x},\mathrm{w}) = \prod_{k=1}^{K} y_k(\mathrm{x},\mathrm{w})^{t_k}\,[1 - y_k(\mathrm{x},\mathrm{w})]^{1-t_k} \tag{1.23}$$

Взяв отрицательный логарифм соответствующей функции правдоподобия, получаем следующую функцию ошибок

$$E(\mathrm{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk} \ln y_{nk} + (1 - t_{nk})\ln(1 - y_{nk})\} \tag{1.24}$$

где y_{nk} эквивалентно $y_k(\mathrm{x}_n, \mathrm{w})$. Опять же, производная функции ошибок относительно активации для конкретного выходного результата принимает форму (1.19), как и в случае регрессии.

Предположим, что мы используем стандартную двухуровневую сеть, изображенную на рисунке 1.3. Мы видим, что весовые параметры на первом уровне сети распределяются между различными выходами, тогда как в линейной модели каждая проблема классификации решается независимым образом. Первый уровень сети можно рассматривать как выполнение нелинейного извлечения признаков, а совместное использование функций между различными выходными данными может позволить сэкономить на вычислениях, а также может привести к улучшенному обобщению.

Наконец, мы рассматриваем стандартную задачу мультиклассовой классификации, в которой каждый вход относится к одному из K взаимоисключающих классов. Бинарные целевые переменные $t_k \in \{0, 1\}$ имеют схему кодирования «1 из K», указывающую класс, а выходные данные сети интерпретируются как $y_k(\mathrm{x}, \mathrm{w}) = p(t_k = 1 \mid \mathrm{x})$, что приводит к следующей функции ошибки

$$E(\mathrm{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathrm{x}_n, \mathrm{w}) \tag{1.25}$$

мы видим, что функция активации блока вывода задается функцией «softmax»

$$y_k(\mathrm{x}, \mathrm{w}) = \frac{\exp(a_k(\mathrm{x}, \mathrm{w}))}{\sum_j \exp(a_j(\mathrm{x}, \mathrm{w}))} \tag{1.26}$$

которая удовлетворяет условию $0 \leq y_k \leq 1$ и $\sum_k y_k = 1$. Обратите внимание, что $y_k(\mathrm{x}, \mathrm{w})$ не изменяется, если константа добавляется ко всем $a_k(\mathrm{x}, \mathrm{w})$, в результате чего функция ошибок остается постоянной для некоторых направлений в весовом пространстве. Это вырождение снимается, если к функции ошибок добавить регуляризацию.

Стоит повторить, что производная функции ошибок относительно активации для конкретного выходного устройства принимает форму (1.19).

Таким образом, существует выбор, как функции активации выходного устройства, так и функции ошибки в соответствии с типом решаемой

проблемы. Для регрессии мы используем линейные выходы и ошибку «суммы квадратов», для (нескольких независимых) двоичных классификаций мы используем логистические сигмоидные выходы и кросс-энтропийную функцию ошибки, а для мультиклассовой классификации мы используем выходы softmax с соответствующей мультиклассовой кросс-энтропийной функции ошибки. Для задач классификации, включающих два класса, мы можем использовать один логистический сигмоидный выход, или в качестве альтернативы мы можем воспользоваться сетью с двумя выходами, имеющую функцию активации «softmax» на выходе.

1.3.1 Оптимизация параметров

Следующим заданием будет нахождение весового вектора **w**, который минимизирует выбранную функцию $E(\mathrm{w})$. Первое, что мы должны отметить это, если мы сделаем маленький шаг в нашем весовом пространстве от w до $\mathrm{w} + \delta\mathrm{w}$, тогда изменение в функции ошибки будет следующим $\delta E \simeq \delta\mathrm{w}^T \nabla E(\mathrm{w})$, где вектор $\nabla E(\mathrm{w})$ направлен в сторону наибольшей скорости увеличения функции ошибки. Из-за того, что $E(\mathrm{w})$ гладкая непрерывная функция от w, ее наименьшая величина появится в такой точке весового пространства, что градиент от функции ошибки исчезает. То есть:

$$\nabla E(\mathrm{w}) = 0 \tag{1.27}$$

Наша цель найти вектор w, при котором $E(\mathrm{w})$ примет наименьшее значение. Наименьшее значение функции ошибки для любого вектора веса называется «глобальным минимумом». Любые другие минимумы, соответствующие более внушительным значениям функции ошибки называют «локальными минимумами». Для успешного применения нейронных сетей, возможно, и не нужно искать глобальный минимум (и вообще неизвестно, будет ли найден глобальный минимум), но может

быть такое, что придется сравнить несколько локальных минимумов для того, чтобы найти достаточно хорошее решение.

Большинство методов включает в себя выбор некоторых начальных значений $\mathrm{w}^{(0)}$ для весового вектора и затем их перемещение по следующему принципу:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} + \Delta\mathrm{w}^{(\tau)} \tag{1.28}$$

где τ – итерационный шаг.

1.3.2 Оптимизация методом градиентного спуска

Самым простым подходом к использованию информации о градиенте является выбор метод обновления веса (1.28), который включает в себя совершение маленького шага в направлении отрицательного градиента. То есть:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} + \eta\nabla E(\mathrm{w}^{(\tau)}) \tag{1.29}$$

Где параметр $\eta > 0$ известен как «*коэффициент скорости обучения*».

На каждом шагу весовой вектор перемещается в направлении наибольшей скорости уменьшения функции ошибки. Этот алгоритм известен как *«градиентный спуск».*

Также существуют более эффективные методы для оптимизации, например *«сопряженные градиенты»* и *«метод quasi-Newton»*. В отличие от градиентного спуска, эти алгоритмы имеют свойство, при котором функция ошибки всегда уменьшается на каждой итерации до тех пор, пока весовой вектор не достигнет глобального минимума.

Для получения достойного минимального значения стоит «прогнать» алгоритм, основанный на градиенте, несколько раз, при этом каждый раз используя разные начальные точки, а после сравнить результаты.

Существует, однако, интерактивная версия градиентного спуска, которая, как оказалось, более полезная для использования на практике с большим набором данных. Функции ошибок, основанные на максимальной вероятности для набора независимых наблюдений, составляют сумму выражений для каждой точки данных:

$$E(\mathrm{w}) = \sum_{n=1}^{N} E_n(\mathrm{w}) \tag{1.30}$$

Интерактивный градиентный спуск, также известный как «последовательный градиентный спуск» вносит изменения в формулу весового вектора. Получается следующее:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} + \eta \nabla E_n(\mathrm{w}^{(\tau)}) \tag{1.31}$$

1.4 Обратное распространение ошибки

В этом разделе мы рассмотрим эффективный способ для оценки градиента функции ошибки E(w) для нейронной сети прямого распространения. Мы увидим, что это может быть достигнуто с использованием локальной схемы передачи сообщений, в которой информация отправляется попеременно вперед и назад по сети, это и называется *«обратным распространением ошибки» (backpropogation или backprop).*

Теперь мы выведем алгоритм обратного распространения для общей сети, имеющей произвольный топологию прямой связи (feed-forward), произвольные дифференцируемые нелинейные функции активации, и широкий класс функций ошибок. Полученные формулы затем будут проиллюстрированы с помощью простой многоуровневой сетевой структуры, имеющей один слой сигмоидальных скрытых единиц вместе с суммой квадратов функции ошибки.

Многие функции ошибок, чаще всего использующиеся на практике, выглядят следующим образом:

$$E(\mathrm{w}) = \sum_{n=1}^{N} E_n(\mathrm{w}) \tag{1.32}$$

Представим простую линейную модель, в которой выходные данные y_k– линейная комбинация входных данных x_i таким образом, что:

$$y_k = \sum_i w_{ki} x_i \tag{1.33}$$

Также вдобавок к линейной модели обозначим функцию ошибки, которая для нашего примера с входной системой из n, примет вид:

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{1.34}$$

Где $y_{nk} = y_k(\mathrm{x}_n, \mathrm{w})$. Градиент этой функции ошибки по отношению к весу w_{ji} выглядит следующим образом:

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{1.35}$$

Это может быть интерпретировано как «локальное» вычисление, включающее результат «ошибочного сигнала» (error signal) $y_{nj} - t_{nj}$.

В общей сети прямого распространения каждая единица вычисляет взвешенную сумму входных данных:

$$a_j = \sum_i w_{ji} z_i \tag{1.36}$$

Где z_i - активация или входная единица, которая связывается с j, и w_{ji}– веса, образованные при помощи этой связи.

Сумма в (1.36), при использовании нелинейной функции активации $h(\cdot)$, дает активацию z_j и принимает вид:

$$z_j = h(a_j) \tag{1.37}$$

Отметим, что одна или несколько переменных z_j в сумме (1.36) могут быть входными данными, также единица j в (1.37) может быть выходом.

Для каждого примера в тренировочном наборе, мы должны предположить, что мы выбрали соответствующий входной вектор для нашей сети и высчитали активации всех скрытых и выходных единиц в сети благодаря последовательному применению выражения в (1.36) и в (1.37). Этот процесс обычно называют *прямым распространением*, потому что это может считаться, как прямой поток информации, осуществляемый через сеть.

Теперь рассмотрим вычисление производной E_n по весу w_{ji}. Мы должны заметить, что E_n зависит от веса w_{ji} через суммированную по j входную единицу a_j. Сформировав правило для частичных производных, получим следующее выражение:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{1.38}$$

Представим полезную замену:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{1.39}$$

Где δ - обычно называется «ошибкой» по причинам, которые мы рассмотрим далее. Используя (1.36), мы можем записать следующее:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{1.40}$$

Подставляя (1.39) и (1.40) в (1.38), мы получим:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{1.41}$$

Таким образом, чтобы вычислить производные, нам нужно только рассчитать значение δ_j для каждой скрытой и выходной единицы в сети, а затем применить (1.41).

Получим следующее выражение:

$$\delta_k = y_k - t_k \tag{1.42}$$

Чтобы вычислить δ для скрытых единиц, мы снова используем введенное нами правило для вычисления производных:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{1.43}$$

Где сумма пробегает по всем значениям *k*, которым *j* отправляет «связь». Устройство значений и весов проиллюстрировано на рисунке 1.6.

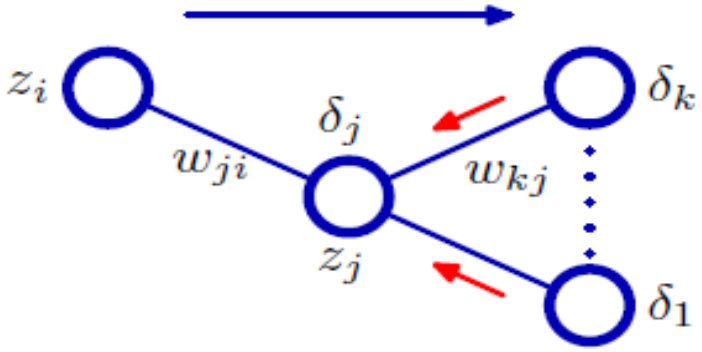

Рисунок 1.6 – Иллюстрация вычисления δ_j для скрытой единицы j путем обратного распространения значений δ от тех блоков k, к которым блок j отправляет соединения.

Отметим, что переменные, помеченные как *k-ые*, могут включать другие скрытые юниты и (или) выходные данные. Если мы теперь заменим обозначение δ, данное в (1.39), значением (1.43), а также воспользуемся

(1.36) и (1.37), мы получим следующую формулу *«обратного распространения».*

$$\delta_j = h'(a_j) \sum_k w_{kj}\, \delta_k \tag{1.44}$$

1.4.1 Использование обратного распространения

Вышеупомянутый вывод процедуры обратного распространения ошибки позволил получить общие формы для функции ошибок, функций активации и топологии сети. Чтобы проиллюстрировать применение этого алгоритма, рассмотрим конкретный пример. Данный пример как из-за его простоты, так и из-за его практической важности, потому что многие модели нейронных сетей используют этот тип сети. В частности, мы рассмотрим двухуровневую сеть, показанную на рисунке 1.3, вместе с ошибкой суммы квадратов, в которой выходные блоки имеют линейные функции активации, так что $y_k = a_k$, в то время как скрытые блоки имеют логистические функции активации, заданные следующим образом

$$h(a) \equiv \tanh(a) \tag{1.45}$$

где

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{1.46}$$

Полезной особенностью этой функции является то, что ее производная может быть выражена в особенно простой форме:

$$h'(a) = 1 - h(a)^2 \tag{1.47}$$

Мы также рассматриваем стандартную функцию ошибок суммы квадратов, так что для шаблона *n* ошибка определяется выражением

$$E_n = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2 \tag{1.48}$$

где y_k - активация блока вывода k, а t_k - соответствующая целевая переменная для конкретного входного шаблона x_n.

Поочередно для каждого шаблона в обучающем наборе мы сначала выполняем прямое распространение, используя

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{1.49}$$

$$z_j = \tanh(a_j) \tag{1.50}$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{1.51}$$

Затем мы вычисляем δ для каждой выходной единицы, используя

$$\delta_k = y_k - t_k \tag{1.52}$$

Затем мы распространяем их в обратном направлении, чтобы получить δ для скрытых единиц, используя

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k \tag{1.53}$$

Наконец, производные по весам первого и второго уровней имеют вид

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_i \tag{1.54}$$

2 Квантовые вычисления

2.1 Проблема квантовых алгоритмов

Классические алгоритмы подразумевают конечную последовательность инструкций, или шаг за шагом выполняемые процедуры. Аналогично, квантовые алгоритмы также подразумевают пошаговое выполнение процедур, выполняемых на квантовом компьютере. Все классические алгоритмы выполнимы также и на квантовом компьютере, однако термин квантовый алгоритм, как правило, используется для алгоритмов, учитывающих особенности квантовых вычислений (суперпозицию, запутанность [11] и т. д.). Все алгоритмы, реализуемые на квантовом компьютере, могут быть реализованы и на классическом компьютере, а все проблемы, неразрешимые при помощи классических компьютеров остаются неразрешимыми и с помощью квантовых компьютеров. Но все же интерес к квантовым компьютерам оправдан тем, что некоторые алгоритмы могут быть выполнены на них гораздо быстрее, чем на классических компьютерах.

2.2 Квантовые компьютеры

Квантовые методы выполнения вычислительных операций, а также передачи и обработки информации, уже начинают воплощаться в реально функционирующих экспериментальных устройствах, что стимулирует усилия по реализации квантовых компьютеров - этого нового направления в вычислительной технике [12-17].

Количество публикаций по квантовой теории информации и квантовым вычислениям приобрело в последнее время лавинообразный характер, появились и экспериментальные работы.

Принципиальная схема работы любого квантового компьютера может быть представлена следующим образом (см. рис.2.1).

Рис. 2.1. Схематическая структура квантового компьютера

Основной его частью является квантовый регистр - совокупность некоторого числа L кубитов. До ввода информации в компьютер все кубиты регистра должны быть приведены в основные базисные (булевы) состояния, то есть в состояние $\{|0_1>, |0_2>, \dots, |0_L>\}$. Эта операция называется подготовкой начального состояния или инициализацией. Далее каждый кубит подвергается селективному воздействию, например, с помощью импульсов внешнего электромагнитного поля, управляемых классическим компьютером, которое переведет основные базисные состояния определенных кубитов в неосновное состояния $a|0> + b|1>$. При этом состояние всего регистра перейдет в суперпозицию базисных состояний вида $|n_c> = |n_1 n_2 \dots n_L>$, где $n = 0$ или 1 , задающую бинарное представление числа $n = \sum_{i=1}^{L} n_i 2^i$.

При вводе информации в квантовый компьютер состояние входного регистра, с помощью соответствующих импульсных воздействий преобразуется в соответствующую когерентную суперпозицию базисных ортогональных состояний $|\psi(0)> = \sum_{n=0}^{2^L} c_n |n_c>$. В таком виде информация далее подвергается воздействию квантового процессора, выполняющего последовательность квантовых логических операций, определяемую унитарным преобразованием $\hat{U}(t)$, действующим на состояние всего регистра. К некоторому моменту времени t в результате преобразований исходное

квантовое состояние становится новой суперпозицией вида $|\psi(t)>=\sum_{n,m}^{2^L-1} c_n \widehat{U}_{nm}|n>.$, которая и определяет результат преобразования информации на выходе компьютера.

Совокупность всех возможных операций на входе данного компьютера, формирующих исходные состояния, а также осуществляющих унитарные локальные преобразования, соответствующие алгоритму вычисления, способы подавления потери когерентности - так называемой декогерентизации (decoherence) квантовых состояний и исправления случайных ошибок, играют здесь ту же роль, что и "программное обеспечение" (software) в классическом компьютере.

Теперь обратимся к аппаратной части квантового компьютера. При выборе конкретной схемы любого квантового компьютера необходимо решить три вопроса: во-первых, выбрать физическую систему, представляющую требуемую систему кубитов, во-вторых, определить физический механизм, определяющий взаимодействие между кубитами, необходимое для выполнения двухкубитовых операций, в-третьих, определить способы селективного управления кубитами и измерения их состояния на выходе. Все это вместе взятое аналогично "аппаратному обеспечению" (hardware) классического компьютера.

Считается, что для реализации полномасштабного квантового компьютера, превосходящего по производительности любой классический компьютер, на каких бы физических принципах он не работал, следует обеспечить выполнение следующих пяти основных требований:

1. Физическая система, представляющая полномасштабный квантовый компьютер, должна содержать достаточно большое число $L > 10^3$ хорошо различаемых кубитов для выполнения соответствующих квантовых операций.

2. Необходимо обеспечить условия для приготовления входного регистра в исходном основном базисном состоянии $\{|0_1>, |0_2>, \dots, |0_L>\}$, то

есть возможность процесса инициализации.

3. Необходимо обеспечить максимальное подавление эффектов декогерентизации квантовых состояний, обусловленное взаимодействием системы кубитов с окружающей средой, что приводит к разрушению суперпозиций квантовых состояний и может сделать невозможной выполнение квантовых алгоритмов. Время декогерентизации должно по крайней мере в 10^4 раз превышать время выполнения основных квантовых операций (времени такта). Для этого система кубитов должна быть достаточно слабо связана с окружением.

4. Необходимо обеспечить за время такта выполнение требуемой совокупности квантовых логических операций, определяющей унитарное преобразование $\widehat{U}(t)$. Эта совокупность должна содержать определенный набор только двухкубитовых операций, типа контролируемого НЕ (аналог исключающего ИЛИ в классических компьютерах), осуществляющих операции поворота вектора состояния двух взаимодействующих кубитов в четырехмерном гильбертовом пространстве, и однокубитовых операций, осуществляющих поворот вектора состояния кубита в двухмерном гильбертовом пространстве, таких как операции НЕ, Адамара и некоторые другие.

5. Необходимо обеспечить с достаточно высокой надежностью измерение состояния квантовой системы на выходе. Проблема измерения конечного квантового состояния является одной из основных проблем квантовых вычислений. Это связано ещё и с тем, что измерения квантового состояния системы изменяют её.

2.3 Квантовый алгоритм классификации

Для решения задачи классификации с помощью квантовых алгоритмах зададим вектора $\overrightarrow{r_j}$, с помощью которых будет происходить обучение.

$$\overrightarrow{r_j} = \left\{\left(\overrightarrow{r_j}\right)_s, s = 0,1,\dots,N-1\right\}, j = 1,2,\dots,M \quad (2.1)$$

$$\overrightarrow{r_1} = \{0,1\} \quad (2.2)$$

$$\overrightarrow{r_2} = \{1,0\} \quad (2.3)$$

Где N – количество параметров, иными словами размерность нашего пространства,

M – число данных для обучения.

В данной задаче рассматриваем двумерное пространство. Поэтому получим вектора следующего вида:

$\overrightarrow{r_1} = \{0,1\}$, который отвечает за первый класс;

$\overrightarrow{r_2} = \{1,0\}$,который отвечает за второй класс

В квантовом виде эти вектора можно представить соответствующим образом:

$$|r_j> = \sum_{s=0}^{N-1} \left(\overrightarrow{r_j}\right)_s |s> \quad (2.4)$$

$$|r_1> = |0> \quad (2.5)$$

$$|r_2> = |1> \quad (2.6)$$

Следующим шагом происходит формирование вектора, который будет классифицироваться.

Классический вид:

$$\vec{r} = \{(\vec{r})_s, s = 0,1,\dots,N-1\} \quad (2.7)$$

Квантовый вид:

$$|r> = \sum_{s=0}^{N-1} (\vec{r})_s |s> \quad (2.8)$$

Вектор состояния (2.8) имеет следующее условие:

$$<r|r> = 1 \quad (2.9)$$

Далее следует использовать функцию *Karush-Kuhn-Tucker*, а именно задача заключается в минимизации параметра L по параметру α. Таким образом, нам нужно найти параметры α, β, которые обеспечивают минимум L.

$$L(\vec{\alpha}) = \sum_{j=1}^{M} y_j \alpha_j - \frac{1}{2} \sum_{j,j'}^{M} \alpha_j \, (\overrightarrow{r_j}, \overrightarrow{r_{j'}}) \alpha_{j'} \tag{2.10}$$

$$y_j = \pm 1 \tag{2.11}$$

$$\sum_{j} \alpha_j = 0 \tag{2.12}$$

$$y_j a_j \geq 0 \tag{2.13}$$

$$\beta = 1 - \alpha |\overrightarrow{r_1}| + \alpha (\overrightarrow{r_1}, \overrightarrow{r_2}) \tag{2.14}$$

Получаем следующее:

$$L(\alpha) = 2\alpha - \alpha^2 \tag{2.15}$$

$$\alpha = 1, \text{т. к.} \left(\frac{\partial L}{\partial \alpha} \right) \tag{2.16}$$

$$\alpha_1 = \alpha, \alpha_2 = -\alpha, \beta = 0 \tag{2.17}$$

Следующая наша цель – получение вектора, по которому обучается модель:

$$\begin{gathered} |U> = \frac{1}{\sqrt{N}} [\beta |00> \otimes |0> + \alpha |1> |0> + (-\alpha) |2> |1>] = \\ = \frac{|10> - |21>}{\sqrt{2}} = \frac{|010> - |101>}{\sqrt{2}} \end{gathered} \tag{2.18}$$

Конечный вид вектора (2.8), который требуется классифицировать, выглядит следующим образом:

$$|r> = \frac{1}{\sqrt{1 + 2r^2}} [|000> + (|01> + |10>) \otimes (x_1 |0> + x2 |1>)] \tag{2.19}$$

Далее формируем вектор состояния ψ, инициализирующий вектор, который будет подан на квантовую схему:

$$|\psi> = \frac{|U> \otimes |0> + |r> \otimes |1>}{\sqrt{2}} \tag{2.20}$$

Следовательно:

$$|\psi> \ = [0,0,\frac{1}{2},0,0,-\frac{1}{2},\ 0,0,\frac{1}{\sqrt{2a}},0,\frac{x_1}{\sqrt{2a}},\frac{x_2}{\sqrt{2a}},\frac{x_1}{\sqrt{2a}},\frac{x_2}{\sqrt{2a}},0,0] \quad (2.21)$$

где

$$a = \sqrt{1+2r^2} \quad (2.22)$$

$$r^2 = {x_1}^2 + {x_2}^2 \quad (2.23)$$

Где x_1, x_2 – параметры, которые отвечают за координаты точки.

После инициализации вектора состояния на нашей схеме мы должны получить следующее значение:

$$P(1) = \frac{1 - Re < U|r >}{2} \quad (2.24)$$

Это результат 1 кубита при значении равном единице.

3 Программная реализация классического и квантового подхода для задачи классификации

3.1 Классический алгоритм

Перейдем теперь к написанию программы, которая позволить нам использовать два метода классификации: классический (с помощью нейронных сетей) и квантовый (с помощью квантовых алгоритмов).

Программа выполнена на языке программирования *Python* версии *3.8.5* в среде разработки *Anaconda*.

Первым шагом совершим импорт библиотек, которые помогут нам сгенерировать данные, а также работать с нейронными сетями [18, 19]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
import random
```

Мы поставим задачу классификации следующим образом. Введем понятие главной диагонали, которая будет делить наши данные на два класса. Первый класс будет иметь синий цвет, точки этого класса будут располагаться выше главной диагонали. Второй класс мы обозначим красным цветом, точки второго класса будут располагаться ниже главной диагонали. Задачей для нейронных сетей будет являться обучение на тренировочной выборке и предсказание класса на точках, принадлежащих тестовой выборке.

Возьмем интервал равный $[-40, 40]$, на этом интервале сгенерируем 80 точек с помощью функции *random*:

```
m, b = 1, 0

lower, upper = -40, 40
num_points = 80
x1 = [random.randrange(start=-40, stop=40) for i in range(num_points)]
x2 = [random.randrange(start=-40, stop=40) for i in range(num_points)]
```

```
y1 = [random.randrange(start=lower, stop=m*x+b) for x in x1]
y2 = [random.randrange(start=m*x+b, stop=upper) for x in x2]



plt.plot(np.arange(-40,40), m*np.arange(-40,40)+b)
plt.scatter(x1, y1, c='red')
plt.scatter(x2, y2, c='blue')
plt.show()
```

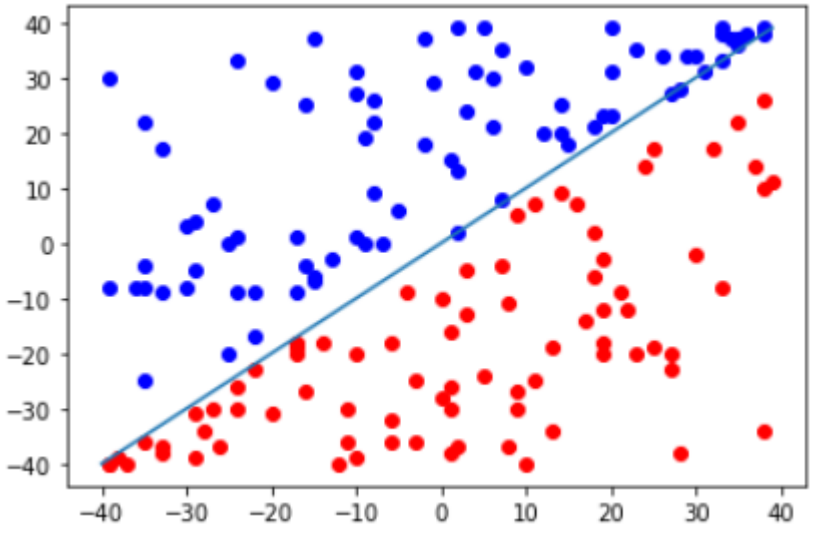

Рисунок 3.1 – Сгенерированные точки и главная диагональ, которая делит их на классы

Далее происходит подготовление наших данных для использования в нейронной сети. В виде входных данных будут использоваться координаты точки. В виде выходных данных будет использоваться их класс (красный или синий). Пометим классы с помощью чисел -1 и 1 для первого и второго класса соответственно:

```
x1, x2, y1, y2 = np.array(x1).reshape(-1,1), np.array(x2).reshape(-
1,1), np.array(y1).reshape(-1,1), np.array(y2).reshape(-1,1)


x_upper = np.concatenate((x2, y2), axis=1)
x_lower = np.concatenate((x1, y1), axis=1)

X = np.concatenate((x_upper, x_lower), axis=0)
res1 = np.array([-1]*len(x1))
res2 = np.array([1]*len(x2))
```

```
y = np.concatenate((res1, res2), axis=0)
```

Теперь зададим параметры нейронной сети. Они были определены эмпирическим путем. Также введем функцию, которая позволит нам определить класс относительно чисел, которые мы этому классу присвоили. Обучаем нейронную сеть на 70% наших данных, оставляя 30% для проверки правильности ее работы.

```
clf = MLPClassifier(activation = 'logistic',
max_iter=1000,hidden_layer_sizes = (5,))

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30)

clf.fit(X_train, y_train)

def get_color(y,zn):
    colors = []

    for i in range(len(y)):
        if y[i] == zn:
            colors.append('red')
        else:
            colors.append('blue')

    return(colors)

colors = get_color(y_train, 1)
```

Визуализируем деление данных на тренировочную и тестовую выборку:

```
plt.scatter(X_train[:, 0], X_train[:, 1], c = colors)
plt.scatter(X_test[:, 0], X_test[:, 1], c ='k', marker = "x")
plt.plot(np.arange(-40,40), m*np.arange(-40,40)+b)
plt.show()
```

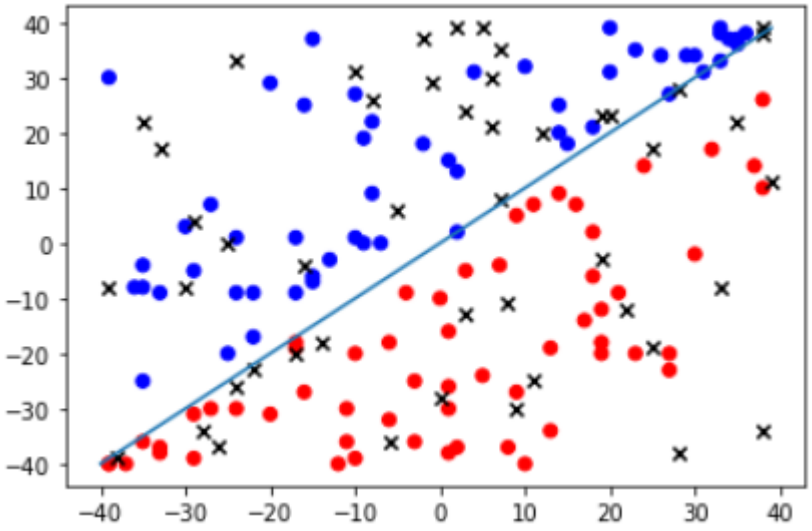

Рисунок 3.2 – Тренировочная и тестовая выборки

На рисунке 3.2 мы можем увидеть результат деления. Красные и синие точки будут являться тренировочной выборкой, черные крестики – точки, которые нам нужно будет предсказать, чтобы проверить, насколько хорошо наша нейронная сеть справляется с задачей классификации.

Переходим к этапу предсказания с помощью нашей обученной модели. Визуализируем результат:

```
y_pred = clf.predict(X_test)

testColors = get_color(y_pred, 1)

plt.scatter(X_train[:, 0], X_train[:, 1], c = colors)
plt.scatter(X_test[:, 0], X_test[:, 1], c = testColors, marker = "x")
plt.plot(np.arange(-40,40), m*np.arange(-40,40)+b)
plt.show()
```

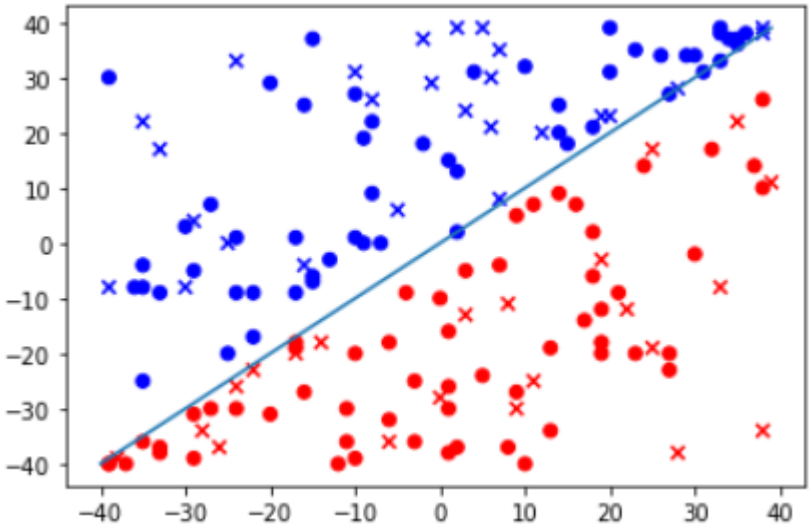

Рисунок 3.3 – Результат предсказания нейронной сети

Из рисунка 3.3 видно, что нейронная сеть идеально справилась с поставленной задачей. Но также следует использовать специальную метрику для оценки точности классификации. Такой метрикой выберем *F-меру*.

F-мера является хорошим кандидатом на формальную метрику оценки качества классификатора. Она сводит к одному числу две других основополагающих метрики: точность и полноту. Имея "F-меру" гораздо проще ответить на вопрос: "поменялся алгоритм в лучшую сторону или нет?"

$$F = \frac{2 \times precision \times recall}{precision + recall} \tag{3.1}$$

Высчитаем F по формуле (3.1):

```
print(f1_score(y_test, y_pred))
```

$F = 1$, это еще раз подтверждает, что наша модель выполнила классификацию без ошибок.

3.2 Квантовый алгоритм

Теперь переходим к квантовой части задачи классификации. Импортируем необходимые библиотеки и функции, а также указываем бэкенд, на котором будет работать наш квантовый алгоритм:

```
import qiskit
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import Aer
from qiskit import execute
backend_sim = Aer.get_backend('qasm_simulator')
```

Для начала задачей будет построение квантовой схемы, которая будет иметь следующий вид:

```
x1 = 20
x2 = 20
r = x1 * x1 + x2 * x2
a = np.sqrt(1 + 2*r)
psi = [0, 0, 0.5, 0, 0, -0.5, 0, 0, 1/(np.sqrt(2)*a), 0,
x1/(np.sqrt(2)*a), x2/(np.sqrt(2)*a), x1/(np.sqrt(2)*a),
x2/(np.sqrt(2)*a), 0, 0]
qc = QuantumCircuit(4)
qc.initialize(psi, [0,1,2,3])
qc.h(3)
qc.measure_all()

qc.draw('mpl')
```

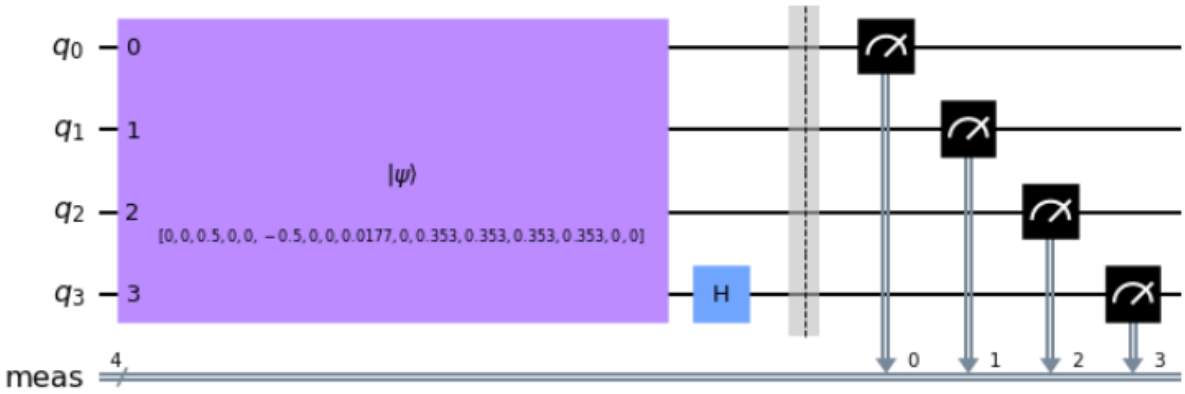

Рисунок 3.4 – Квантовая схема для т.(20,20)

На рисунке 3.4 видна нужная для построения квантовая схема, на которую мы инициализировали данные для случайной точки. Теперь мы будем

использовать функцию, которая позволит нам инициализировать все тестовые точки в данную схему:

```
def quant_state(x1, x2, backend_sim = backend_sim):
    r = x1 * x1 + x2 * x2
    a = np.sqrt(1 + 2*r)
    psi = [0, 0, 0.5, 0, 0, -0.5, 0, 0,          1/(np.sqrt(2)*a), 0,
x1/(np.sqrt(2)*a), x2/(np.sqrt(2)*a), x1/(np.sqrt(2)*a),
x2/(np.sqrt(2)*a), 0, 0]
    qc = QuantumCircuit(4)
    qc.initialize(psi, [0,1,2,3])
    qc.h(3)
    qc.measure_all()
    job_sim = execute(qc, backend_sim, shots=1000)
    result_sim = job_sim.result()
    counts = result_sim.get_counts(qc)

    quantumState_1 = 0
    for i in counts.keys():
        list_i = list(i)
        if list_i[0] == '1':
            quantumState_1 += counts[i]

    return quantumState_1
```

В данной функции, помимо инициализации, фиксируется ее результат. А именно значение формулы (2.24) - результат первого кубита при значении равном единице. Относительно этого значения мы будем определять класс для нашей тестовой выборки. Если значение (кол-во измерений) больше или равно 500, то мы будем присваивать точке первый класс, если меньше то второй.

```
quant_res = []


for i,j in zip(X_test[:, 0], X_test[:, 1]):
    qs = quant_state(i, j, backend_sim = backend_sim)
    if qs >= 500:
        quant_res.append(-1)
    else:
        quant_res.append(1)


quantColors = get_color(quant_res, 1)
```

Проделав данную классификацию, мы визуализируем результат:

```
plt.scatter(X_train[:, 0], X_train[:, 1], c = colors)
plt.scatter(X_test[:, 0], X_test[:, 1], c = quantColors, marker = "x")
plt.plot(np.arange(-40,40), m*np.arange(-40,40)+b)
plt.show()
```

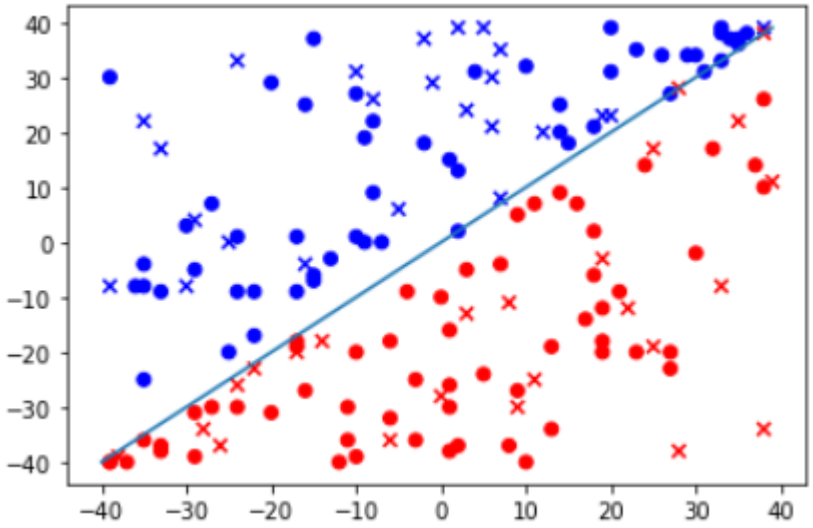

Рисунок 3.5 – Результат классификации с помощью квантового алгоритма

Также высчитывается *F-меру* для данного результата:

```
print(f1_score(y_test, quant_res))
```

$F = 0.95$, это означает, что квантовый алгоритм успешно справился с задачей классификации.

Чтобы закрепить результат, мы проверим наш алгоритм с помощью облачной среды IBM Quantum Experience [20]. Это позволит свободно подключиться к квантовому процессору IBM с помощью своего обычного компьютера.

Произведем подключение к квантовому процессору

```
from qiskit import IBMQ
provider = IBMQ.load_account()
backend = provider.get_backend('ibmq_qasm_simulator')
```

Проделаем тот же самый алгоритм, используя бэкенд IBM.

Получаем следующий результат:

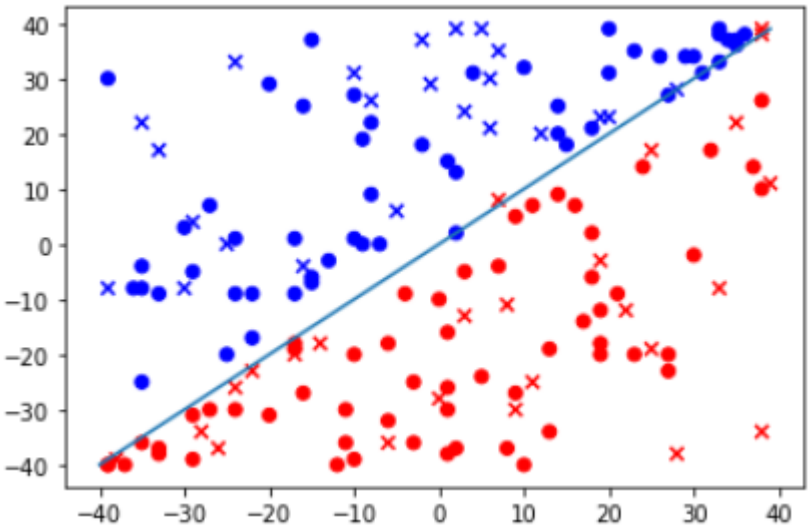

Рисунок 3.6 – Результат классификации на бэкенде IBM

$$F = 0.93$$

Значение F на квантовом процессоре практически равно предыдущему нашему значению на обычном симуляторе.

Несмотря на то, что нейронная сеть дала безупречный результат, квантовый алгоритм может смело конкурировать с классическим алгоритмом и предоставлять результат близкий к идеальному.

Заключение

В ходе дипломной работы были решены следующие задачи:

1. Реализована нейронная сеть, классифицирующая данные
2. Реализован квантовый алгоритм классификации данных.
3. Произведено сравнение двух алгоритмов (классического и квантового).

Список использованных источников

1. Хайкин С. Нейронные сети [Текст] : полный курс : [перевод с английского] / С. Хайкин. - Изд. 2-е, испр. - Москва ; Санкт-Петербург : Диалектика, 2019. - 1103 с.
2. Розенблатт Ф. Принципы нейродинамики: Перцептроны и теория механизмов мозга. / Ф. Розенблатт. – М.: Мир, 1965. –480 с.
3. Москалев Н. С. Виды архитектур нейронных сетей // Молодой ученый. — 2016. — №29. — С. 30-34. — URL https://moluch.ru/archive/133/37121/ (дата обращения: 11.06.2021).
4. Барский А. Б. Нейроинформатика. Однослойные логические нейронные сети : учебное пособие / А. Б. Барский ; Московский гос. ун-т путей сообщ. (МИИТ), Каф. "Вычислительные системы и сети". - Москва : МИИТ, 2008. - 208 с.
5. Никифоров И. К. Нейросетевые технологии. Искусственные нейронные сети : учебное пособие / И. К. Никифоров ; М-во образования и науки Российской Федерации, Федеральное агентство по образованию, Федеральное гос. образовательное учреждение высш. проф. образования "Чувашский гос. ун-т им. И. Н. Ульянова". - Чебоксары : Изд-во Чувашского ун-та, 2008. - 263 с.
6. Рассел С. Искусственный интеллект [Текст] : современный подход : [перевод с английского] / С. Рассел, П. Норвиг. - 2-е изд. - Москва : Диалектика ; Санкт-Петербург : Диалектика, 2019. - 1407 с.
7. Гудфеллоу Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль ; [пер. с англ. А. А. Слинкина]. - 2-е цв. изд., испр. - Москва : ДМК Пресс, 2018. - 651 с.
8. Микелуччи У. Прикладное глубокое обучение : подход к пониманию глубоких нейронных сетей на основе метода кейсов / У. Микелуччи ; перевод с английского Андрея Логунова. - Санкт-Петербург : БХВ-Петербург, 2020. - 368 с.

9. Крон Д. Глубокое обучение в картинках : визуальный гид по искусственному интеллекту : 16+ / Д. Крон, Г. Бейлевельд и А. Бассенс ; [перевел с английского А. Киселев]. - Санкт-Петербург [и др.] : Питер, 2020. - 399 с.
10. Bishop C. M. Pattern Recognition and Machine Learning / C. M. Bishop – Springer Science+Business Media, LLC, 2006. – 729 с.
11. Запрягаев С.А. Введение в квантовые информационные системы / С.А. Запрягаев. – Воронеж: ВГУ, 2015. – 219 с.
12. Ожигов Ю. И. Квантовый компьютер / Ю. И. Ожигов ; Московский государственный университет имени М. В. Ломоносова, Факультет вычислительной математики и кибернетики. - Москва : МАКС Пресс, 2020. - 171 с
13. Williams, Colin P. Explorations in quantum computing / Colin P. Williams, Scott H. Clearwater. - New York : Springer : Telos, Cop. 1998. - XX, 307 с.
14. Стин Э. Квантовые вычисления / Э. Стин; Пер. с англ. И.Д. Пасынкова. - М. ; Ижевск : Науч.-изд. центр "Регуляр. и хаот. динамика", 2000. - 111 с.
15. Кайе Ф. Введение в квантовые вычисления [Текст] / Ф. Кайе, Р. Лафламм, М. Моска ; пер. с англ. Т. С. Никитиной ; под науч. ред. А. В. Анохина. - Москва ; Ин-т компьютерных исслед. ; Ижевск : R & C dynamics, 2009. - 346 с.
16. Сысоев С. С. Введение в квантовые вычисления. Квантовые алгоритмы : учебное пособие / С. С. Сысоев ; Санкт-Петербургский государственный университет. - Санкт-Петербург : Изд-во Санкт-Петербургского ун-та, 2019. - 143 с.
17. Имре Ш. Квантовые вычисления и связь : инженерный подход / Ш. Имре, Ф. Балаж ; пер. с англ. А. А. Калачёва [и др.] ; под ред. В. В. Самарцева. - Москва : Физматлит, 2008. - 319 с.
18. Мюллер А. Введение в машинное обучение с помощью Python [Текст] : руководство для специалистов по работе с данными : [полноцветное издание] / А. Мюллер, С. Гвидо ; [перевод с английского и редакция А. В. Груздева]. - Москва [и др.] : Диалектика, 2017. - 472, [1] с.

19. Дейтел П. Python. Искусственный интеллект, большие данные и облачные вычисления / [П. Дейтел, Х. Дейтел ; перевел с английского Е. Матвеев]. - Санкт-Петербург [и др.] : Питер, 2020. - 861 с.
20. Силва В. Разработка с использованием квантовых компьютеров : программирование квантовых машин в облаке: Python, Qiskit, Quantum Assembly language и IBM QExperience / В. Силва ; [перевел с английского К. Синица]. - Санкт-Петербург [и др.] : Питер, 2020. - 351 с.