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

Факультет компьютерных наук

Кафедра цифровых технологий

Вероятностный подход к обработке последовательных данных

ВКР Бакалаврская работа

02.03.01 Математика и компьютерные науки

Распределенные системы и искусственный интеллект

Допущено к защите в ГЭК __.__.2022

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

Обучающийся _____ *А. М. Гузенко, 4 курс, д/о*
подпись

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

Воронеж 2022

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

Факультет компьютерных наук

Кафедра цифровых технологий

УТВЕРЖДАЮ
заведующий кафедрой
_____ _____
подпись, расшифровка подписи
__.__.2022

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
ОБУЧАЮЩЕГОСЯ ГУЗЕНКО АЛЕКСЕЯ МИХАЙЛОВИЧА
фамилия, имя, отчество

1. Тема работы «Вероятностный подход к обработке последовательных данных», утверждена решением ученого совета факультета компьютерных наук от __ .__.2022

2. Направление подготовки / специальность 02.03.01 Математика и компьютерные науки

3. Срок сдачи законченной работы __.__.2022

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

№	Структура ВКР	Сроки выполнения	Примечание
1	Введение	16.10.2021-2.11.2021	
2	Литературный обзор	2.11.2021–5.12.2021	
3	Последовательные данные	5.12.2021–30.01.2022	
4	Нейронные сети	30.01.2022-25.03.2022	
5	Вероятностный подход в машинном обучении	25.03.2022-19.05.2022	
6	Заключение	19.05.2022-30.05.2022	
7	Список использованных источников	30.05.2022-02.05.2022	

Обучающийся _____ Гузенко А.М.
Подпись *расшифровка подписи*

Руководитель _____ Клинских А.Ф.
Подпись *расшифровка подписи*

РЕФЕРАТ

Реферат Бакалаврская работа с. 55, 1 таблица, 17 рисунков, 20 источников

МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, БАЙЕСОВСКИЙ МЕТОД, КЛАССИФИКАЦИЯ

Объектом исследования являются классический и байесовский подходы в решении задач машинного обучения, в частности, классификации в рамках проблемы обработки и анализа временных рядов.

Цель работы: сравнение двух подходов к построению нейронных сетей, классического подхода и байесовского подхода.

Степень внедрения: готовы два программных продукта для проведения сравнения эффективности двух подходов для решения задач классификации с использованием нейронных сетей.

В результате работы была выявлена эффективность работы байесовской нейронной сети на сгенерированных данных. Вероятностный подход в машинном обучении показал свою эффективность в решении задачи классификации.

Содержание

Введение

Применение анализа временных рядов и методов машинного обучения играют огромную роль в жизни человека и делает большой шаг в технологическом прогрессе. Машинное обучение применяется во многих сферах, от сетевых магазинов до промышленных производств. Где-то они носят рекомендательный характер для человека, а в каких-то уже могут заменить человека полностью. В пример можно привести систему автоматического торможения от Volvo, учитывая дистанцию и скорость, при помощи передней камеры, в режиме реального времени рассчитывая тормозной путь, система может принять решение об экстренном торможении, с реакцией, которая не сравнится с человеческой.

Анализ временных рядов и машинное обучение получили свое быстрое развитие в основном по двум причинам: увеличение количества информации для анализа и развитие вычислительных мощностей, но всему есть предел. Если для информации одна из основных проблем это правильный сбор и последующее интерпретирование информации, то для машинного обучения актуален вопрос эффективности алгоритмов. Для решения данной проблемы используются как другие вычислительные модели, например, квантовые вычисления, так и поиск более эффективных алгоритмов для обучения модели. Один из таких алгоритмов является вероятностный подход, или же Байесовский подход, названный в честь Томаса Байеса, который разработал одну из важнейших теорем в теории вероятности, теорему Байеса о зависимых вероятностях.

В последнее время вероятностных подход нашел отклик и стремительно развивается, созданы несколько библиотек для создания моделей, анализа данных. Решаются проблемы эффективности и точности моделей, созданием улучшенных методов для работы с вероятностными распределениями.

Главные задачи в данной работе:

1. Взять временной ряд, в котором требуется решить задачу классификации по одному или нескольким переменным.
2. Реализовать две модели для решения задач классификации.
3. Сравнить обычный подход в решении задачи классификации и вероятностный подход в эффективности на разных выборках.
4. Сравнить скорости обучения моделей на равном объеме данных.
5. Сделать выводы об областях применения данного подхода, его преимуществах и недостатках.

Литературный обзор

В первую очередь стоит отметить, что рассматриваемая область быстроразвивающаяся, и, соответственно, источники для исследования в данной области должны быть как можно более актуальными, но не стоит забывать о качестве этих источников, так как некоторые поспешные выводы могут быть ошибочны, но данный тезис в большей мере касается научных статей, результатов недавно проведенных исследований. Основополагающим для данной работы я выбрал три литературных источника, остальные же либо в какой-то мере дополняют их, либо более углубленно раскрывают некоторые темы.

Для анализа временных рядов за основу была взята книга «Практический анализ временных рядов: Прогнозирование со статистикой и машинное обучение» Эйлин Нильсен 2019 года. Эйлин кандидат прикладной физики Колумбийского университета, занималась исследованиями влияния развития технологий на право и медицину. В данный момент она занимается разработкой нейронной сети для прогнозирования в сфере финансов. В этой книге предлагается практическое знакомство с временными рядами, их анализом, обработкой и последующим применением.

Для Байесовского подхода за основу были взяты книги «Байесовские модели. Байесовская статистика на языке Python» Аллен Б. Дауни 2013 года и «Байесовский анализ на Python: введение в статистическое моделирование и вероятностное программирование с использованием PyMC3 и ArviZ» Освальдо Мартин 2018 года.

Аллен Б. Дауни профессор компьютерных наук в инженерном колледже штата Массачусетс, выпустил множество книг по обработке данных и программированию в целом. Книга по Байесовской статистике является руководством к применению языка программирования Python для данной сферы, без углубления в математические основы.

Освальдо Мартин ученый-исследователь агентства National Scientific and Technical Research Council (CONICET) в Аргентине, работающий в области структурной биоинформатики, является одним из основных разработчиков библиотек PyMC3 и ArviZ, преподаватель курса по Байесовскому анализу данных. Книга представляет собой введение в Байесовский анализ, излагается методический подход моделирования вероятностных моделей, применение теоремы Байеса для вывода логических следствий из используемых моделей и данных.

Дополнительно использовались следующие источники.

Для понимания математических процессов, то есть теории вероятности и математической статистики, идеально подойдут работы двух советских математиков, а именно «Курс теории вероятностей» Бориса Владимировича Гнеденко и «Теория вероятностей и математическая статистика» Владимира Ефимовича Гмурмана. Данные две книги дают систематические знания по теории вероятностей и математической статистике, предоставляют разбор реальных примеров и задач для самостоятельного решения.

Для понимания общего устройства и работы нейросетей подойдут некоторые разделы следующих книг «Глубокое обучение с точки зрения практика» Джош Паттерсон, «Pattern Recognition and Machine Learning» Christopher Michael Bishop, «Building Machine Learning and Deep Learning Models on Google Cloud Platform» Ekaba Bisong, «Глубокое обучение» Гудфеллоу Ян, «Прикладное глубокое обучение: подход к пониманию глубоких нейронных сетей на основе метода кейсов» Микелуччи Умберто.

Для лучшего математического понимания Байесовской статистики была использована книга «Bayesian Data Analysis» Andrew Gelman. Так же, данный автор описал в научном журнале алгоритм, который мы будем рассматривать в дальнейшем.

Для рассмотрения нового алгоритма Automatic Differentiation Variational Inference взята статья Alp Kucukelbir из научного журнал

1 Последовательные данные

1. 1 Понятие последовательных данных

Последовательные данные (или временные ряды) – это серия точек данных, проиндексированных во временном или ином порядке. Чаще всего временной ряд представляет собой множество, полученное в последовательных равностоящих точках времени [1].

Данные временных рядов и их анализ приобретают все большее значение из-за получения таких данных посредством, например, интернет вещей, цифровизации здравоохранения, развития умных городов и так далее. По мере того, как непрерывный мониторинг и сбор данных становятся все более распространёнными, потребность в анализе временных рядов с использованием как статистических методов, так и методов машинного обучения, повысится. Анализ временных рядов часто сводится к вопросу о причинно-следственной связи, как прошлое повлияло на будущее.

Практическое применение анализа временных рядов и машинного обучения еще в 1980-ых годах включал широкий спектр сценариев:

- Специалисты по компьютерной безопасности задействовали их для выявлений аномалий в качестве метода идентификации против взломов и вмешательств.
- Динамическое изменение масштаба времени, один из доминирующих методов измерения сходства между временными рядами. С увеличением вычислительной мощности стало возможно достаточно быстро вычислять «расстояние» между, например, аудиозаписями.
- Изобретены рекурсивные нейронные сети, которые показали свою эффективность для извлечения шаблонов из поврежденных данных.

Для анализа временных рядов используются различные графические отображения данных, такие как графики, сводные статистики, гистограммы, диаграммы рассеяния. Рассмотрим некоторые из них.

Во временных рядах, как и в данных, можно произвести операцию группировки, то есть выделить отдельные группы значений. Например, в перекрестных данных операции группировки позволяют определить средние значения для значений возраста, пола или места проживания респондентов. В анализе временных рядов так же востребованы групповые операции, позволяющие рассчитывать статистические показатели наборов, например, средимесячные или недельные медианы. Данные одного временного ряда становятся более понятными при разделении на несколько параллельных временных рядов.

Возьмем сведения о ежедневных ценах закрытия четырех основных европейских фондовых индексов, указанных с 1991 по 1998 годах, и отобразим их на графике. На рисунке 1 мы видим пример стандартного графика временного ряда [1].

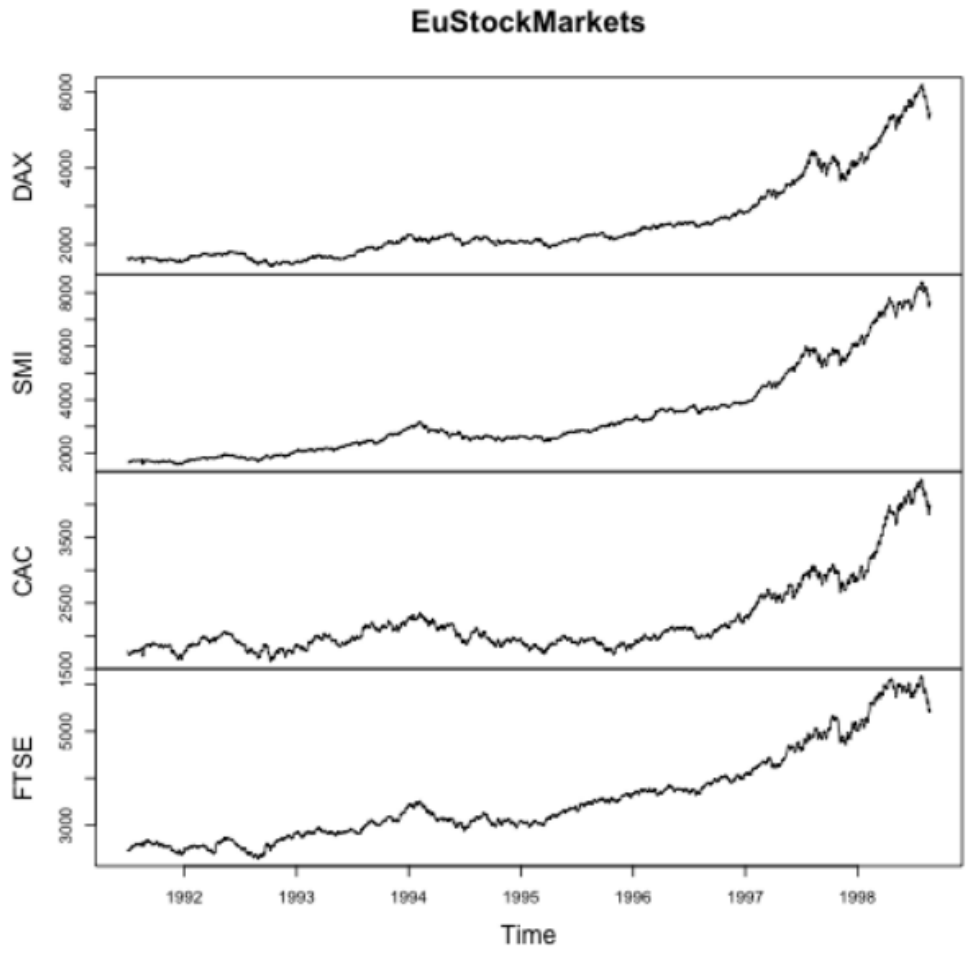

Рисунок 1 – График временного ряда

При помощи гистограммы, как на рисунке 2, мы можем оценить частоту появления данных, или, что более важно, проанализировать изменение значения во времени.

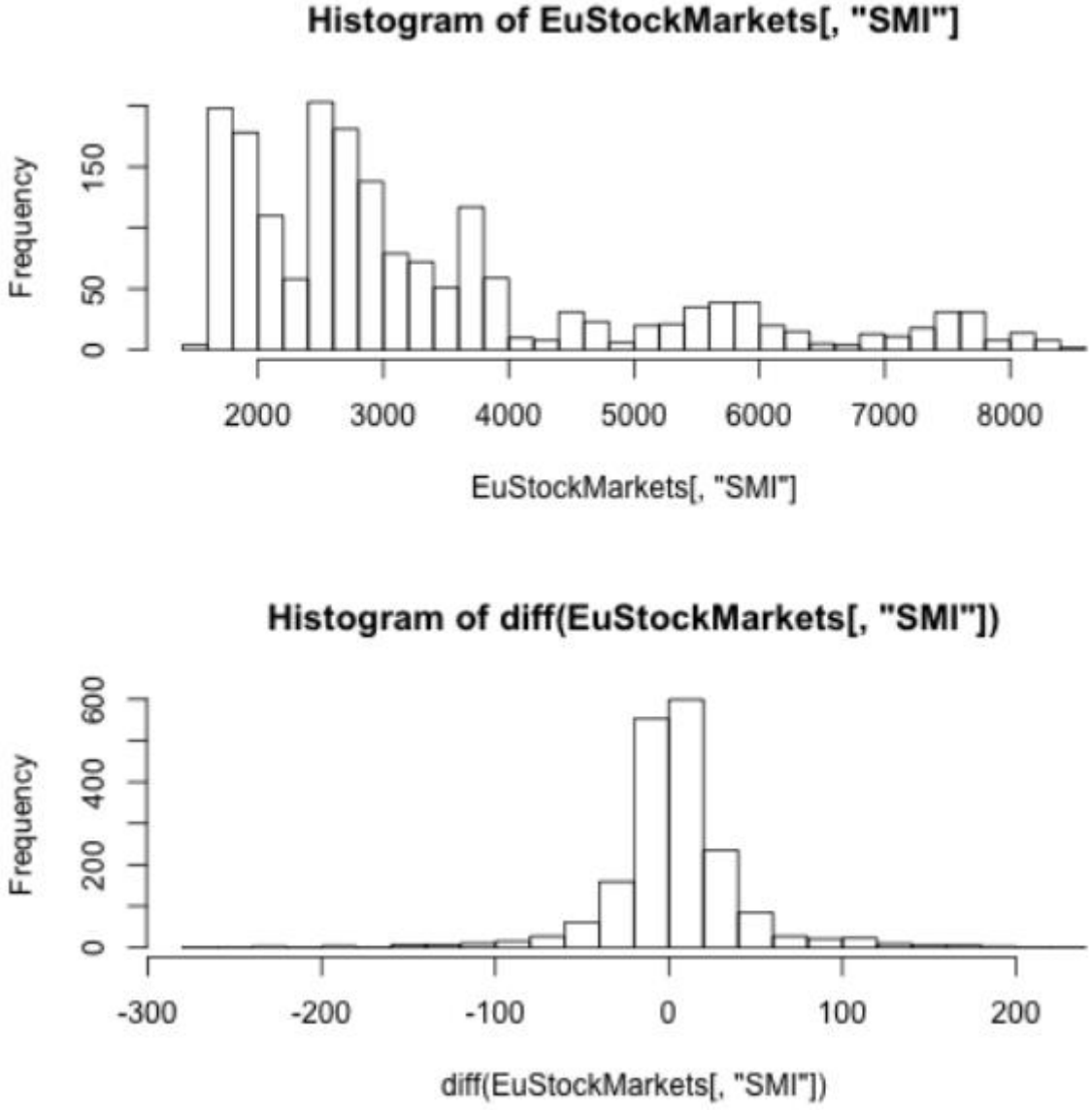

Рисунок 2 – Гистограмма временного ряда. Первая гистограмма – частота значений, вторая гистограмма – частота разностей

Диаграммы рассеяния мы можем использовать для определения взаимосвязи между ценами двух акций в отдельные моменты времени, а также отслеживания их временных изменений. На рисунке 3 присутствуют оба варианта.

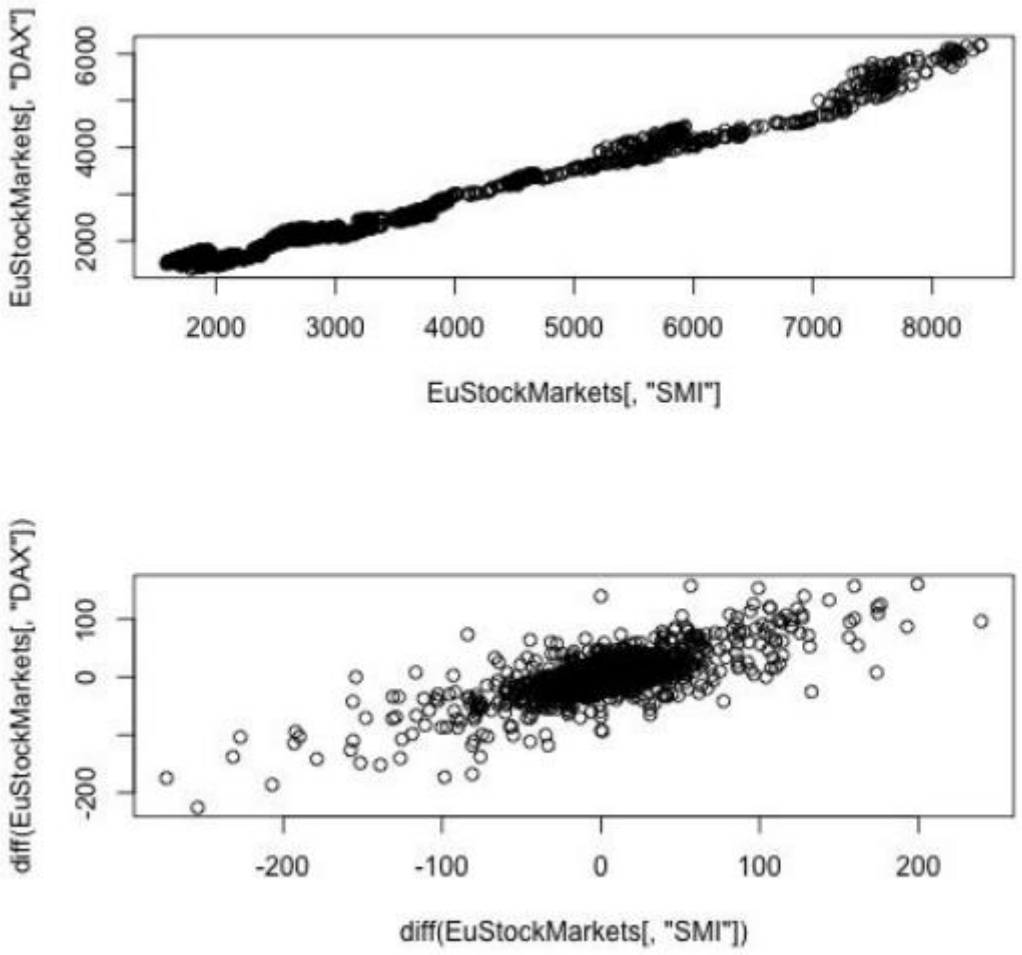

Рисунок 3 – Диаграмма рассеяния. На первой диаграмме – цена двух акций с течением времени, на второй – разница цен двух акций с течением времени

Помимо графического анализа стоит обращать внимание на некоторые характеристики временных рядов, такие как стационарность, корреляция, ложная корреляция.

Один из важных вопросов при анализе временного ряда – это какую систему он описывает, стабильную или изменчивую? Уровень стабильности, или стационарности, важен для оценки того, как долгосрочное поведение в прошлом отражает ее поведение в будущем. После оценки уровня стабильности мы пытаемся определить, присуще ли ему динамическое поведение. Простое определение стационарности процесса заключается в следующем: процесс

считается стационарным, если для всех возможных смещений k распределение $y_t, y_{t+1}, \dots, y_{t+k}$ не зависит от t.

Самокорреляция – значение временного ряда в отдельные моменты времени могут коррелировать с его значениями в другие моменты времени.

Автокорреляция – общий случай самокорреляции, лишенному привязки к конкретному моменту времени. Автокорреляция сводится к решению задачи поиска взаимосвязи между любыми двумя точками общего временного ряда, расположенными на строго заданном расстоянии друг от друга, то есть автокорреляция дает представление о линейной взаимосвязи точек данных, полученных в разные моменты времени, как о функции разницы времени их получения [1].

Ложные корреляции – математическая зависимость, в которой переменные или события связаны, но в следствии совпадения. Ложные корреляции остаются важной проблемой, требующей самого пристального изучения и опровержения. График ложных коореляций изображен на рисунке 4.

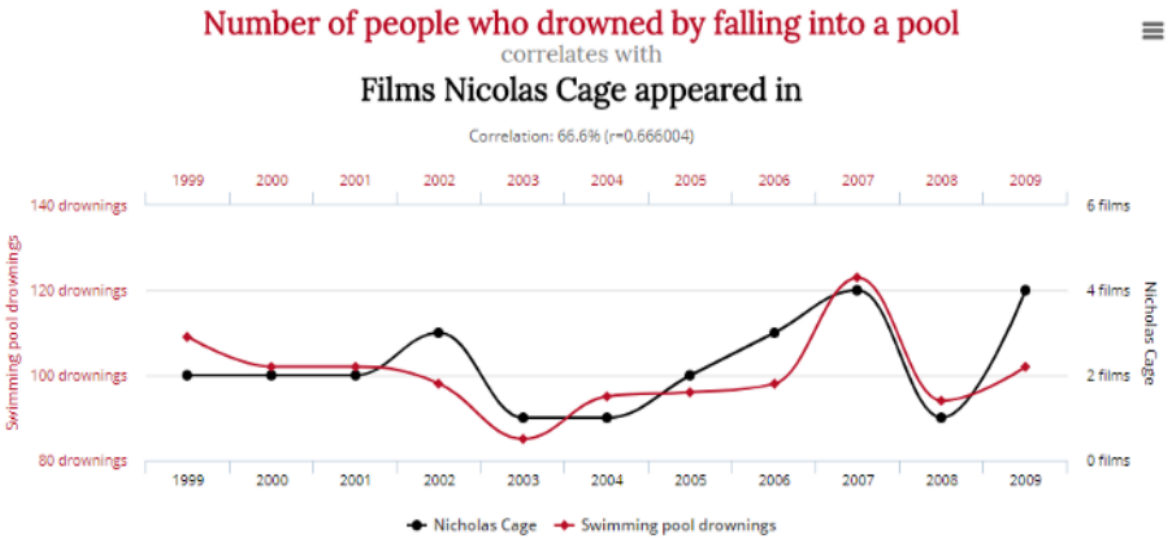

Рисунок 4 – График выхода фильмов с Николасом Кейджем и количество утонувших людей в бассейне

Моделирование данных выступает разновидностью анализа данных, который находит широкое применение при работе с временными рядами. Это

следует из одного недостатка временных рядов: никакие две точки данных в одном и том же временном ряду не могут быть точно сопоставимы, поскольку они относятся к разному времени. Как только мы задумаемся о том, что могло бы произойти в данный момент времени, мы придём к моделированию.

Моделирование временных рядов методами глубокого обучения – это новая, но весьма многообещающая дисциплина науки о данных. С помощью технологии глубокого обучения во временных рядах можно обнаружить сложные динамические процессы, зачастую скрытые для понимания даже опытных специалистов по анализу данных.

Моделирование и прогнозирование – схожие задачи. В обоих случаях сначала нужно сформулировать гипотезу о параметрах и поведении базовой системы, а затем экстраполировать имеющиеся данные для получения новых точек.

Но нужно четко понимать различия между моделированием и прогнозированием:

- Иногда качественные наблюдения проще обрабатывать методами моделирования, а не прогнозирования.
- Моделирование выполняется в определённом масштабе, что позволяет увидеть множество альтернативных сценариев, в то время как прогнозы составляются предельно точно.

1. 2 Вероятностный подход к анализу последовательных данных

Для начала стоит напомнить, что такое вероятность в общем понимании. Вероятность – это число между 0 и 1, которые представляют собой уровень уверенности, что некоторый факт справедлив. Числом 1 представляется абсолютная уверенность, что некоторый факт справедлив. Числом 0 – абсолютная уверенность, что этот факт не справедлив. Промежуточные числа в этом интервале определяют степень уверенности. Число 0,5 означает, что предсказанное событие в одинаковой степень может как осуществиться, так и не осуществиться [2-3].

Распределение вероятностей – это математический объект, который описывает, насколько возможными являются различные события. Эти события ограничены каким-либо образом, то есть представляют собой набор возможных событий, например набор возможных чисел $\{1, 2, 3, 4, 5, 6\}$ для игральных костей (исключая неопределенные случаи). Общепринятым и полезным представлением концепции в статистике является интерпретация данных как генерируемых из некоторого истинного распределения вероятностей с неизвестными параметрами. То есть нам нужно найти значения этих параметров с использованием только частичной выборки из истинного распределения вероятностей. В обобщенном случае у нас нет доступа к такому истинному распределению вероятностей, поэтому необходимо создать модель, чтобы попытаться аппроксимировать это распределение. Вероятностные модели создаются при помощи правильного объединения распределений вероятностей. На рисунке 5 показан пример графика вероятностного распределения.

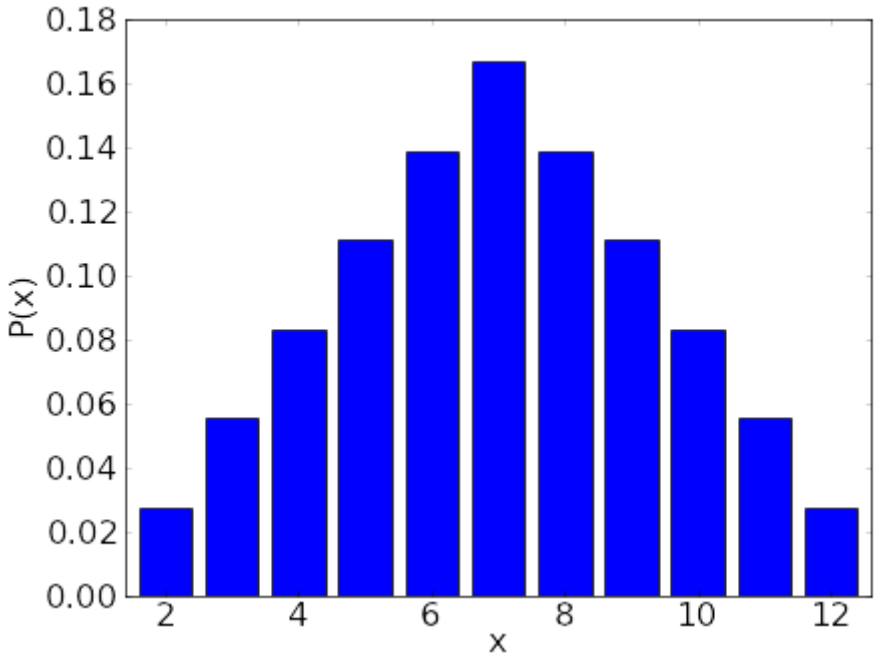

Рисунок 5 – Вероятностное распределение броска двух игральных костей

Для построения сложных моделей активно используется теорема Байеса (1.1)

$$p(\theta, y) = \frac{p(y|\theta)p(\theta)}{p(y)} \quad (1.1)$$

Если заменить элемент θ на «предположение» (гипотезу), а элемент y на «данные», то теорема Байеса показывает, как вычислить вероятность предположения θ при наличии данных y. Для того, чтобы превратить предположение в некоторый объект, нужно использовать вероятностные распределения. В вероятностном подходе для анализа данных мы часто будем использовать теорему Байеса. Ниже приведем названия для элементов данной теоремы [4-5]:

- $p(\theta)$ – априорная вероятность
- $p(y|\theta)$ – правдоподобие
- $p(\theta|y)$ – апостериорная вероятность
- $p(y)$ – предельное правдоподобие

Априорная вероятность должна соответствовать тому, что нам известно о значении параметра θ перед рассмотрением данных y. Если нам ничего не известно, то можно использовать постоянные фиксированные априорные вероятности, которые не содержат какого-либо значимого объема информации.

Правдоподобие определяет, как будут представлены данные в дальнейшем анализе. Это выражение правдоподобности данных с учетом принятых параметров.

Апостериорная вероятность – это результат байесовского анализа, которые отображает все, что известно о задаче (проблеме) с учетом имеющихся данных и используемой модели. Апостериорная вероятность – это распределение вероятностей для параметра θ в используемой модели. Такое распределение это баланс между априорной вероятностью и правдоподобием. С теоретической концептуальной точки зрения апостериорную вероятность можно

воспринимать как обновленную вероятность в свете новых данных. Апостериорная вероятность, полученная в результате одного процесса анализа, может использоваться как априорная вероятность для нового процесса анализа. Это свойство делает байесовский анализ особенно подходящим для анализа данных, которые становятся доступными в определенном последовательном порядке. Примерами могут служить системы раннего оповещения о природных катастрофах, которые обрабатывают в режиме онлайн данные, поступающие с метеорологических станций и спутников.

Правдоподобие – это вероятность исследуемых данных, усредненная по всем возможным значениям, которые могут принимать параметры. В любом случае мы не уделяем особого внимания предельному правдоподобию и будем считать его простым фактором нормализации. Такой подход принят потому, что при анализе распределения апостериорной вероятности нас будут интересовать только относительные, а не абсолютные значения параметров, в итоге получится формула (1.2). На рисунке 6 показан пример работы Байесовской теоремы.

$$p(\theta, y) \propto p(y|\theta)p(\theta) \tag{1.2}$$

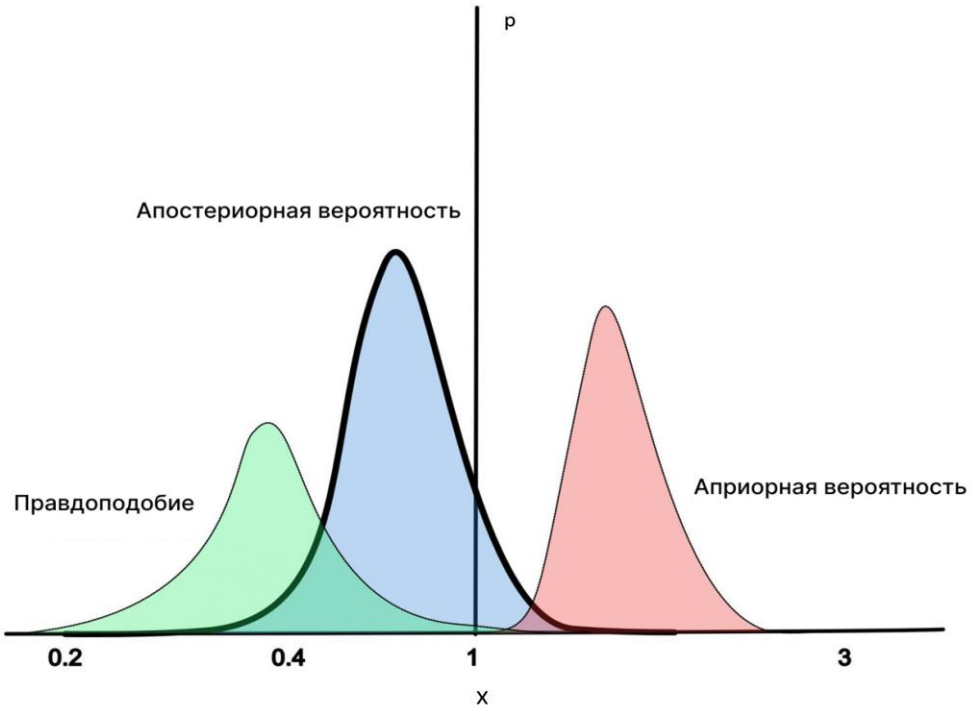

Рисунок 6 – Графический пример работы Байесовской теоремы

2 Нейронные сети

2. 1 Нейронная сеть прямого распространения

Линейные модели для регрессии и классификации базируются на линейных комбинациях фиксированных нелинейных базисных функций $\phi_j(x)$ и имеют форму, показанную на формуле (2.1)

$$y(x, w) = f\left(\sum_{j=1}^{M} \omega_j \phi_j(x)\right) \tag{2.1}$$

где $f(\cdot)$ нелинейная функция активации в случае классификации и идентично в случае регрессии. Наша цель расширить данную модель создав базисную функцию $\phi_j(x)$, которая будет зависеть от параметров и потом позволим этим параметрам изменяться вместе с коэффициентами $\{\omega_j\}$. Конечно, существует много способов построения параметрической нелинейной базисной функции. Нейронные сети используют базисные функции формы, подобной (2.1), поэтому каждая базисная функция является нелинейной функцией для линейных входных данных, где коэффициенты в линейной комбинации являются адаптивными параметрами [6].

Это приводит нас к базовой нейронной сети, которая может быть описана как серия функциональных преобразований. В начале мы строим M линейных комбинаций входных переменных $x_1, \dots, x_D$ в форме, показанную на формуле (2.2)

$$a_j = \sum_{i=1}^{D} \omega_{ji}^{(1)} x_i + \omega_{j0}^{(1)} \tag{2.2}$$

где $j = 1, \dots, M$ и верхний индекс (1) означает, что данные параметры находятся на «первом» уровне нейронной сети. Мы будем ссылаться на параметр $\omega_{ji}^{(1)}$ как

на вес, а на параметр $\omega_{j0}^{(1)}$ как на погрешность. Переменные a_j известны как «активатор». Потом каждая из них преобразуется с помощью дифференцируемой нелинейной «функции активации» $\mathrm{h}(\cdot)$, что дает нам формулу (2.3)

$$z_j = h(a_j) \tag{2.3}$$

Переменные z_j соответствуют результатам в базисной функции (2.1), которые в контексте нейронных сетей, называются «скрытым узлом». Нелинейная функция $h(\cdot)$ обычно являются сигмоидальными функциями, такими как логическая сигмоидная или «tanh» функция. Учитывая (2.1) эти значения снова линейно комбинируются, чтобы получить выходной слой активации, показанной на формуле (2.4).

$$a_k = \sum_{j=1}^{M} \omega_{kj}^{(2)} z_j + \omega_{k0}^{(2)} \tag{2.4}$$

где $k = 1, \dots, K$ и K общее число результатов. Это преобразование соответствует второму уровню нейронной сети, и снова $\omega_{k0}^{(2)}$ является параметром погрешности. Наконец, выходной слой активации преобразуется, используя соответствующую функцию активации, чтобы дать набор выводов нейронной сети y_k. Выбор функции активации основывается на природе данных и предполагаемым распределением искомых данных и следует тем же соображениям, что и для линейных моделей. Таким образом, для стандартных регрессионных задач функция активации идентична, значит $y_k = a_k$. Аналогичным образом, для нескольких задач бинарной классификации, каждый выходной слой активации преобразуется, используя логическую сигмоидную функцию так, что получим формулу (2.5)

$$y_k = \sigma(a_k) \tag{2.5}$$

где

$$\sigma(a_k) = \frac{1}{1 + exp(-a)} \tag{2.6}$$

Мы можем объединить различные этапы для того, чтобы получить общую сетевую функцию, которая для сигмоидной функции слоя активации имеет вид

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{M} \omega_{kj}^{(2)}\, h\left(\sum_{i=1}^{D} \omega_{ji}^{(1)}\, x_i + \omega_{j0}^{(1)}\right) + \omega_{k0}^{(2)}\right) \tag{2.7}$$

где набор всех параметров веса и погрешности сгруппирован в один вектор w. Таким образом нейронная сеть — это нелинейная функция из множества входных $\{x_i\}$ и выходных $\{y_k\}$ значений, регулируемых вектором w, вектором регулируемых параметров. На рисунке 7 мы можем увидеть двухслойную нейронную сеть.

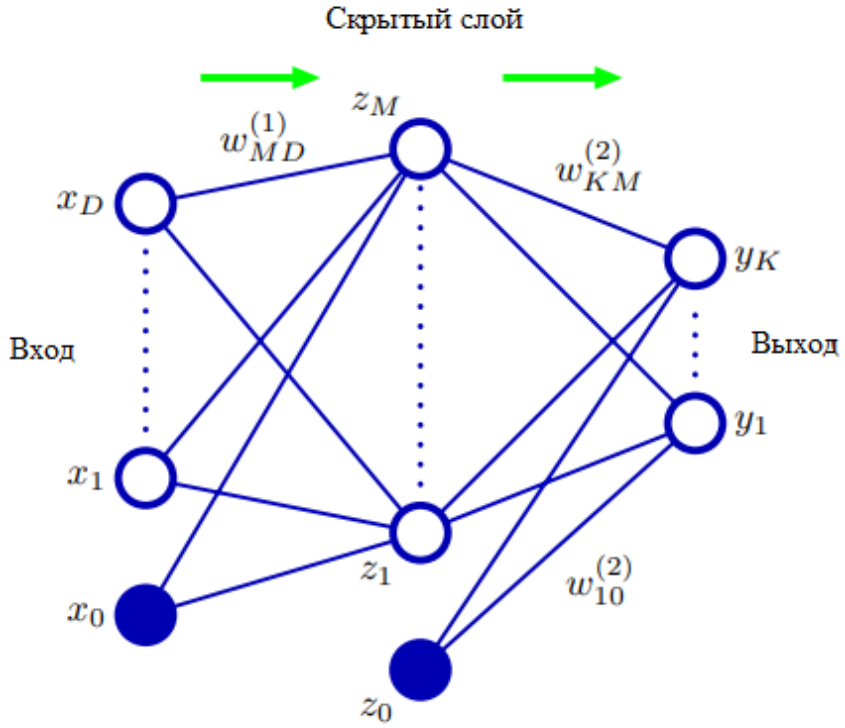

Рисунок 7 – Сетевая диаграмма для двухслойной нейронной сети

Процесс расчета (2.7) может быть интерпретирована как прямое распространение данных через нейронную сеть.

Параметры погрешности в (2.2) могут быть включены во множество весовых параметров при помощи определения дополнительной входной переменной x_0, определим ее значение как $x_o = 1$, тогда (2.2) примет вид, описанный в формуле (2.8)

$$a_j = \sum_{i=0}^{D} \omega_{ji}^{(1)} x_i \tag{2.8}$$

Мы можем аналогичным образом превратить веса второго слоя в погрешности второго слоя, так что общая сетевая функция примет вид, описанный в формуле (2.9)

$$y_k(x, w) = \sigma\left(\sum_{j=0}^{M} \omega_{kj}^{(2)} h\left(\sum_{i=0}^{D} \omega_{ji}^{(1)} x_i\right)\right) \tag{2.9}$$

Как мы могли видеть на рисунке 2, модель нейронной сети содержит два этапа обработки данных, каждая из которых напоминает «персептрон» и по этой причине нейронная сеть известна как многослойный персептрон (MLP или multilayer perceptron). Ключевое различие с персептроном — это то, что MLP использует непрерывные сигмоидальные нелинейности в скрытом слое, в то время как персептрон использует кусочно-постоянные нелинейности. Это значит, что MLP дифференцируется по отношению к параметрам нейронной сети и это свойство имеет решающую роль в обучении нейронной сети [7].

Если функции активации скрытого слоя в нейронной сети приняты как линейные, то для каждой нейронной сети мы всегда можем найти эквивалентную нейронную сеть без скрытого слоя. Это следует из того факта, что композиция последовательных линейных преобразований и есть, само по себе, линейной преобразование. Однако, если число скрытых узлов меньше, чем количество узлов ввода и вывода, то преобразования этой нейронной сети не являются наиболее

общими возможными линейными преобразованиями ввода и вывода, так как данные теряются в уменьшении размерности скрытых узлов.

Архитектура нейронной сети, показанная на рисунке 2, является наиболее часто используемой на практике. Однако ее легко обобщить, рассматривая дополнительные слои обработки, каждая из которых состоит из взвешенной линейной комбинации вида (2.4) с последующим поэлементным преобразованием при помощи нелинейной функции активации. Обратим внимание, что существует некоторая путаница в терминологии, по отношению к подсчету количества слоев в таких сетях. Таким образом сеть на рисунке 2 может быть описана как трехуровневая сеть (которая считает количество узлов в слоях и трактует входные данные как узлы). Мы рекомендуем терминологию, в которой нейронная сеть на рисунке 2 называется двухуровневой нейронной сетью, так как количество слоев адаптивных весов важнее для определения свойств сети.

Другим обобщением сетевой архитектуры является включение соединения «пропускного слоя», каждое из которых связано с соответствующим адаптивным параметром. Для реализации, в двухслойной нейронной сети они будут идти напрямую от входным данным к выходным данным. В принципе, нейронная сеть с сигмоидными спрятанными узлами могут всегда может имитировать соединения пропускного слоя (для ограниченных входные значений) используя достаточно небольшой первый слой весов, что во всем рабочем диапазоне скрытые узлы являются линейными, а затем компенсируется большим значением весов скрытого слоя для выходных данных. На практике, однако, возможно эффективно включать пропускной слой явно.

Кроме того, сеть может быть разрозненной и не все возможные соединения присутствуют.

Поскольку существует прямое соответствие между диаграммой нейронной сети и ее математической функцией, мы можем разработать общие сетевые отображения, за счет рассмотрения более сложных диаграмм нейронных сетей. Однако они должны быть ограничены архитектурой с прямым распространением, другими словами, для тех, у кого нет замкнутых направленных цикло,

нужно убедиться, что выходные данные являются детерминированными функциями входных данных. Это продемонстрировано на простом примере на рисунке 3. Каждый (скрытый или входной) узел в нейронной сети вычисляется как

$$z_k = h\left(\sum_j \omega_{kj}\, z_j\right) \tag{2.10}$$

где сумма проходит по всем узлам, которые отправляют соединения с узлом k (и параметр погрешности включен в суммирование). Для заданного набора значений, примененных к входным данным нейронной сети, последовательное применение (2.10) позволяет активировать все узлы в нейронной сети, подлежащие расчету, включая выходные слои.

Нейронные сети называют универсальными аппроксиматорами. Для примера, двухслойная нейронная сеть с линейными выходными данными может равномерно аппроксимировать любую непрерывную функцию на компактной входной области с произвольной точностью при условии, что нейронная сеть имеет достаточное количество скрытых узлов. Этот результат справедлив для широкого для широкого спектра функций активации скрытых узлов, за исключением полиномиальных. Хотя такие теоремы обнадеживают, ключевая проблема заключается в том, как найти подходящие значения параметров из множества тренировочных данных.

Способность двухуровневой сети моделировать широкий спектр функций изображена на рисунке 8.

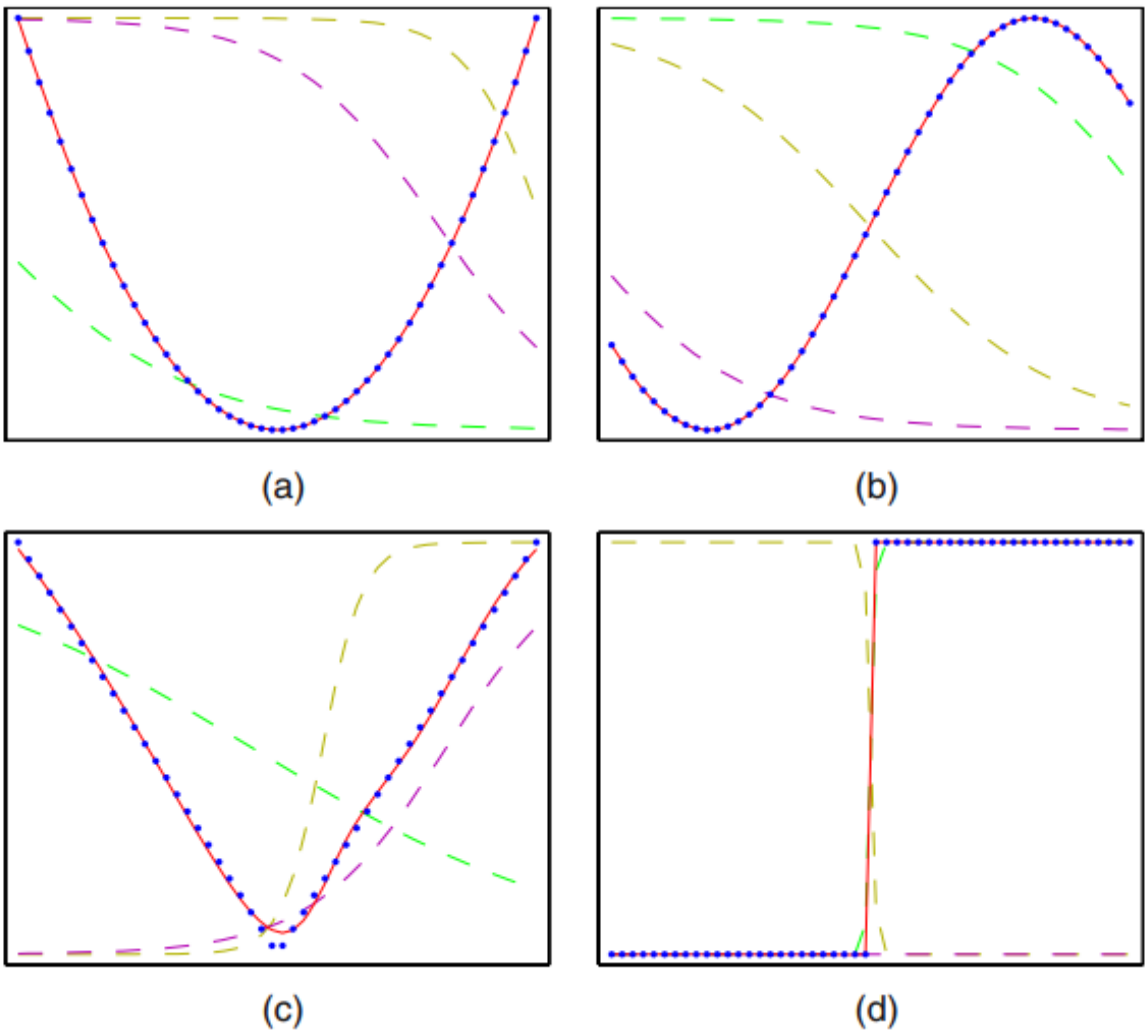

Рисунок 8 – Иллюстрация возможностей многослойного персептрона для аппроксимации четырех различных функций (a) $f(x) = x^2$ (b) $f(x) = sin(x)$ (c) $f(x) = |x|$ (d) $f(x) = H(x)$, где $H(x)$ – ступенчатая функция Хэвисайда. В каждом примере $N = 50$ точки данных показаны синими точками, были выбраны равномерно в интервале $(-1,1)$ и соответствующие вычисленные значения $f(x)$. Эти точки в дальнейшем используются для обучения двухуровневой сети, имеющей скрытые блоки с функциями активации «tanh» и линейными выходными блоками. Результирующие сетевые функции показаны красными кривыми, а выходы трех скрытых блоков показаны тремя пунктирными кривыми.

2. 2 Обучение сети

До сих пор мы рассматривали нейронные сети как общий класс параметрических нелинейных функций из вектора x (входных параметров) в вектор y (выходных параметров). Простой подход к проблеме определения параметров нейронной сети состоит в том, чтобы минимизировать функцию ошибки суммы

квадратов. Учитывая обучающий набор, состоящий из входных параметров вектора x_n, где $n = 1, \dots, N$ вместе с соответствующим набором целевых векторов t_n, мы минимизируем функцию ошибки, описанную в формуле (3.1).

$$E(w) = \frac{1}{2}\sum_{n=1}^{N} ||y(x_n, w) - t_n||^2 \tag{3.1}$$

Однако мы можем предоставить гораздо более общий взгляд на обучение нейронной сети, в первую очередь дав вероятностную интерпретацию выходным данным сети [8-9]. Нам это даст более четкую мотивацию, как для выбора нелинейности выходных узлов, так и для выбора функции ошибки. Мы начнем с обсуждения проблем регрессии, а пока рассмотрим единственную целевую переменную t, которая может принимать любое реальное значение. Мы предполагаем, что t имеет Гауссово распределение с зависимыми от x средним значением, которое задается выходными значениями нейронной сети, так что

$$p(t|x, w) = N(t|y(x, w), \beta^{-1} \tag{3.2}$$

где β точность (обратная дисперсии) Гауссовского шума. Для условного распределения (3.2) достаточно в качестве функции активации выходного устройства взять тождество, так как такая сеть может аппроксимировать любую непрерывную функцию от x до y. Учитывая набор данных из N независимых, одинаково распределенных наблюдений $X = \{x_1, \dots, x_N\}$, вместе с соответствующими целевыми значениями $t = \{t_1, \dots, t_N\}$, мы можем построить соответствующую функцию правдоподобия, описанную в формуле (3.3)

$$p(t|X, w, \beta) = \prod_{n=1}^{N} p(t_n, x_n, w, \beta) \tag{3.3}$$

Взяв отрицательный логарифм, мы получим формулу (3.4)

$$\frac{\beta}{2}\sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2 - \frac{N}{2} ln\, \beta + \frac{N}{2} ln(2\pi) \tag{3.4}$$

с помощью которых мы можем узнать параметры w и β. Начнем с рассмотрения w. Максимизация функции правдоподобия эквивалентная минимизации функции ошибки суммы квадратов, определяемая как описано в формуле (3.5)

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 \tag{3.5}$$

где отброшены аддитивные и мультипликативные константы. Значение w находится минимизацией $E(w)$, которое будет обозначено w_{ML}, так как оно соответствует решению с максимальным правдоподобием. На практике нелинейность функции нейронной сети $y(x_n, w)$ приводит к «не выпуклости» ошибки $E(w)$, и поэтому на практике локальные максимумы правдоподобия могут быть найдены путем соответствия локальным минимумам функции ошибки.

Найдя w_{ML}, значение β может быть найдено за счет минимизации отрицательной логарифмической вероятности, в результате получим формулу (3.6)

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, w_{ML}) - t_n\}^2 \tag{3.6}$$

Обратим внимание, что это можно рассчитать после завершения итеративной оптимизации, необходимой для поиска w_{ML}. Если у нас есть несколько целевых переменных и мы предполагаем, что они независимые, обусловленные x и w с общей точностью шума β, то условное распределение целевых значений определяется выражением (3.7)

$$p(t|x, w) = N(t|y(x, w), \beta^{-1}I) \tag{3.7}$$

Следуя тому же аргументу, что и для одной целевой переменной, мы видим, что максимальные весы правдоподобия определяются путем минимизации функции ошибок суммы квадратов (3.1). Тогда точность шума определяется выражением (3.8)

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}\left||y(x_n, w_{ML}) - t_n|\right|^2 \tag{3.8}$$

где K – количество целевых переменных. От предположения о независимости можно отказаться при помощи немного более сложной задачи оптимизации.

Мы можем рассматривать нейронную сеть, как имеющую на выходе функцию активации, которая тождественна, так что $y_k = a_k$. Соответствующая функция ошибок суммы квадратов имеет свойство, описанное в формуле (3.9)

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{3.9}$$

которое мы будем использовать при обсуждении обратного распространения ошибок в разделе 4. Обратное распространение ошибки.

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 рассмотрим нейронную сеть, имеющую один выход, который описан функцией активации в виду логической сигмоиды (3.10).

$$y = \sigma(a) \equiv \frac{1}{1 + exp(-a)} \tag{3.10}$$

тогда $0 \leq y(x, w) \leq 1$. Мы можем интерпретировать $y(x, w)$ как условную вероятность $p(C_1|x)$, где $p(C_2, x)$ задается уравнением $1 - y(x, w)$. Условное распределение целевых переменных при входных данных является распределением Бернулли в виде формулы (3.11)

$$p(t|x, w) = y(x, w)^t\{1 - y(x, w)\}^{1-t} \tag{3.11}$$

Если мы рассматриваем обучающий набор независимых наблюдений, тогда функция ошибки, которая дается отрицательным логарифмом функции правдоподобия, будет кросс-энтропийной функцией ошибок формы (3.12)

$$E(w) = -\sum_{n=1}^{N}\{t_n \, ln \, y_n + (1 - t_n) \, ln(1 - y_n)\} \tag{3.12}$$

где y_n обозначает $y(x_n, w)$. Отметим, что нет аналога шумовой точности β, так как предполагается, что целевые значения правильно помечены. Однако модель легко расширяется, чтобы учесть ошибки маркировки. Simard et al. (2003) обнаружил, что использование кросс-энтропийной функции ошибок вместо суммы

квадратов для задачи классификации приводит к более быстрому обучению, а так же к лучшему обобщению.

Если у нас есть K отдельных двоичных классификаций для выполнения, то мы можем использовать нейронную сеть, имеющую K выходов, каждая из которых представлена логической сигмоидной функцией активации. С каждой выходом связана метка двоичного класса $t_k \in \{0,1\}$, где $k = 1, \dots, K$. Если мы предположим, что метки классов независимы, учитывая входной вектор, то условное распределение целевых данных будет иметь вид (3.13)

$$p(t|x,w) = \prod_{k=1}^{K} y_k(x,w)^{t_k} \left[1 - y_k(x,w)\right]^{1-t_k} \tag{3.13}$$

Отрицательный логарифм соответствующей функции правдоподобия дает следующую функцию ошибок, в виде (3.14)

$$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\, ln\, y_{nk} + (1 - t_{nk})\, ln(1 - y_{nk})\} \tag{3.14}$$

где y_{nk} обозначает $y_k(x_n, w)$. Снова, производная функций ошибок со спектром активации для конкретных выходных узлов принимает вид (3.9), как и в случае регрессии.

Предположим, что мы используем стандартную двухуровневую нейросеть показанную. Мы видим, что весовые параметры в первом слое сети распределяются между различными выходами, тогда как в линейной модели каждая классификация решается независимо. Первый уровень сети может рассматриваться как выполнение нелинейного извлечения признаков, и совместное использование извлечения между различными выходными данными может сэкономить на вычислениях, а также может привести к улучшению обобщения.

Наконец, мы рассмотрим стандартную задачу много классовой классификации, в которой каждый вход относится к одному из K взаимоисключающих классов. Бинарные целевые переменные $t_k \in \{0,1\}$ имеет схему кодирования 1 из K с указанием класса, а выходы нейронной сети интерпретируются как

$y_k(x,w) = p(t_k = 1|x)$, в итоге имею следующую функцию ошибки в виде формулы (3.15)

$$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn}\, ln\, y_k\,(x_n, w) \tag{3.15}$$

Мы видим, что функция активации узлов выхода, соответствующая канонической ссылке, задается функцией «SoftMax», в виде формулы (3.16).

$$y_k(x,w) = \frac{exp\big(a_k(x,w)\big)}{\sum_j exp\left(a_j(x,w)\right)} \tag{3.16}$$

что удовлетворяет $0 \le y_k \le 1$ и $\sum_k y_k = 1$. Отметим, что $y_k(x,w)$ не изменяются, если ко всем $a_k(x,w)$ добавляется константа, в результате чего функция ошибок остается постоянной для некоторых направлений в пространстве весов. Это вырождение снимается, если к функции ошибок добавить соответствующий член регуляризации.

Еще раз, производная функции ошибок относительно активации для конкретного выходного узла принимает знакомую форму (3.9).

Таким образом, существует естественный выбор как функции активации узла вывода, так и функции ошибки согласования, в зависимости от типа решаемой проблемы. Для регрессии мы используем линейные выходы и ошибку суммы квадратов, для (нескольких независимых) двоичных классификаций мы используем логистические сигмоидные выходы (функции активации для этих выходов) и функцию кросс-энтропийной ошибки, а для много классовой классификации мы для выходов используем «SoftMax», с соответствующей мультиклассовой кросс-энтропийной функцией ошибки. Для задач классификации, включающих два класса, мы можем использовать один логистический сигмоидный выход, или, как альтернатива, мы можем использовать нейронную сеть с двумя выходами, каждая из которых имеет функцию активации «SoftMax». На рисунке 9 рассмотрен график функции ошибки $E(w)$

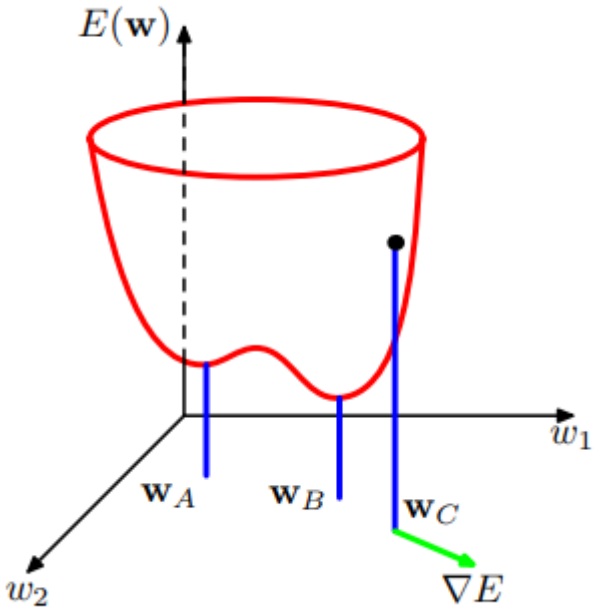

Рисунок 9 – Геометрический вид функции ошибки $E(w)$ как поверхности, расположенной над пространством весов. Точка w_A – это локальный минимум, а w_B – глобальный минимум. В любой точке w_C локальный градиент ошибки задается вектором ∇E.

2. 3 Обратное распространение ошибки

Наша цель в этом разделе – найти эффективный метод расчета градиента функции ошибок $E(w)$ для нейронной сети с прямой связью.

Мы покажем, что это может быть достигнуто с помощью схемы передачи локальных сообщений, в которой информация пересылается поочередно вперед и назад по сети и известна как «Распространение ошибок» или проще «Обратное распространение».

Следует отметить, что термин обратное распространение используется в литературе для обозначения множества разных вещей. Например, многослойную архитектуру персептрона иногда называют сетью обратного распространения. Также термин обратного распространения используется для описания обучения

многослойного персептрона с использованием градиентного спуска, применяемого к функции ошибок суммы квадратов. Для уточнения терминологии полезно более внимательно рассмотреть характер тренировочного процесса. Большинство обучающих алгоритмов включают итеративную процедуру минимизации функции ошибок, с корректировкой весов. Каждый шаг мы можем разбить на два этапа. На первом этапе необходимо вычислить производные функции ошибок по весам. Как мы увидим позже, важный вклад обратного распространения заключается в предоставлении эффективного метода для вычисления таких производных. Так как именно на этом этапе ошибки распространяются по нейронной сети в обратном направлении, мы будем использовать термин обратное распространение специально для расчета производных. На втором этапе производные затем используются для вычисления корректировок, которые необходимо внести в веса. Самый просто из таких приемов – градиентный спуск. Важно понимать, что эти две стадии различны. Таким образом, первый этап, а именно распространение ошибок назад по сети для расчета производных, может быть применен ко многим другим типам сетей, а не только к многослойному персептрону. Его также можно применить к функциям ошибок, отличающихся от суммы квадратов, и к вычислению других производных, таких как матрицы Якобиана и Гессе. Аналогично, второй этап регулировки веса с помощью вычисленных производных могут быть решены с использованием различных схем оптимизации, многие из которых значительно более мощные, чем простой градиентный спуск [10].

2. 4 Расчет производных функции ошибок

Теперь мы выведем алгоритм обратного распространения ошибок для общей сети, имеющей произвольную топологию прямой связи, произвольные дифференцируемые функции активации и широкий класс функции ошибок. Затем полученные формулы будут проиллюстрированы, используя многослойную нейронную сеть, имеющей один слой сигмоидальных скрытых узлов вместе с функцией ошибки суммы квадратов.

Многие функции ошибок, представляющий практический интерес, например те, которые определяются максимумом функции правдоподобия для набора независимых и одинаково распределенных данных, содержат сумму членов, по одному для каждой точки данных в обучающем наборе, поэтому получим формулу (4.1)

$$E(w) = \sum_{n=1}^{N} E_n(w) \tag{4.1}$$

Здесь мы рассмотрим задачу вычисления градиента ∇E для одного такого члена в функциях ошибок. Это может быть использовано непосредственно для последовательной оптимизации, или результаты могут накапливаться по обучающей выборке в случае пакетных методов.

Рассмотрим сначала простую линейную модель, в которой выводы y_k являются линейными комбинациями x_i, так что формула (4.2)

$$y_k = \sum_i \omega_{ki}\, x_i \tag{4.2}$$

вместе с функцией ошибок, которая для конкретного шаблона n принимает форму (4.3)

$$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{tk})^2 \tag{4.3}$$

где $y_{nk} = y_k(x_n, w)$. Градиент этой функции ошибок относительно веса ω_{ji} определяется выражением (4.4)

$$\frac{\partial E_n}{\partial \omega_{ji}} = \big(y_{nj} - t_{nj}\big)x_{ni} \tag{4.4}$$

который может быть интерпретирован как «локальное» вычисление включающие результат «сигнала ошибки» $y_{nt} - t_{nj}$ связанного с выходным концом линии ω_{ji} и переменной x_{ni} связанного с входным концом линии.

В общей сети с прямой связью каждый узел вычисляет взвешенную сумму своих входных данных в форме

$$a_j = \sum_i \omega_{ji} z_i \tag{4.5}$$

где z_i – это активация узла или входа, который отправляет соединение узлом j и ω_{ji} вес, который связан этим подключением. В разделе 2 мы видели, что смещения можно включить в эту сумму, введя дополнительную единицу или вход, с фиксированной функцией активацией +1. Поэтому нам не нужно явно разбираться с предубеждениями явно. Сумма в (4.4) преобразуется нелинейной функцией активации $h(\cdot)$, чтобы дать активацию z_j узла j в виде формулы (4.6)

$$\mathrm{z_j} = \mathrm{h}\left(\mathrm{a_j}\right) \tag{4.6}$$

Обратим внимание, что одна или несколько переменных z_i в сумме (4.5) могут быть входными, и аналогично, узлы j могут быть выходными. Для каждого шаблона в обучающем наборе мы будем предполагать, что мы предоставили соответствующий входной вектор в сеть и вычислили функции активации всех скрытых и входных узлов в сети путем последовательного применения (4.5) и (4.6). Этот процесс часто называют прямым распространением, так как его можно рассматривать как прямой поток данных через сеть.

Теперь рассмотрим расчет производной от E_n по отношению к весу ω_{ji}. Выходы различных узлов будут зависеть от конкретного входного шаблона n. Однако, чтобы не загромождать обозначения, мы будем опускать индекс n в сетевых переменных. Прежде всего отметим, что E_n зависит только от веса ω_{ji} только через суммированный вход a_j в узел j. Следовательно, мы можем применять цепное правило для частных производных, чтобы получить формулу (4.7)

$$\frac{\partial E_n}{\partial \omega_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial \omega_{ji}} \tag{4.7}$$

теперь введем полезное обозначение (4.8)

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{4.8}$$

где δ ошибка по причинам, которую мы скоро увидим. Используя (4.6), мы можем написать

$$\frac{\partial a_j}{\partial \omega_{ji}} = z_i \tag{4.9}$$

подставляя (4.8) и (4.9) в (4.7), мы получим

$$\frac{\partial E_n}{\partial \omega_{ji}} = \delta_j z_i \tag{4.10}$$

Уравнение (4.10) говорит нам, что требуемая производная получается простым умножением значения δ для веса конечного узла выхода и z для веса конечного узла входа (где $z = 1$ в случае смещения). Обратим внимание, что уравнение примет ту же форму, что и для простой линейной модели, рассмотренной во втором разделе. Таким образом, чтобы рассчитать производные, нам нужно только вычислить значение δ_j для каждого скрытого и выходного узла в сети, а затем применить в (4.9)

Как мы могли видеть для выходных узлов у нас есть

$$\delta_k = y_k - t_k \tag{4.11}$$

при условии, что мы используем каноническую ссылку в качестве функции активации в выходном узле. Чтобы рассчитать δ для скрытых узлов, мы снова используем цепное правило для частных производных. В итоге получим формулу (4.12)

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{4.12}$$

где сумма пробегает все узлы k, к которым узел j отправляет соединения. Расположение узлов и весов проиллюстрировано на рисунке 6. Обратим внимание, что узлы, помеченные k, могут включать другие скрытые узлы и/или узлы входа. В записи (4.11) мы используем тот факт, что вариации a_j вызываю вариации функции ошибок только через вариации переменных a_k. Если мы теперь подставим определение δ из (4.7) в (4.11) и воспользуемся формулами (4.4) и (4.5), мы получим следующую формулу обратного распространения ошибок.

$$\delta_j = h'(a_j) \sum_k \omega_{kj}\, \delta_k \tag{4.13}$$

которая говорит нам, что значение δ для конкретного скрытого узла может быть получено путем распространения δ в обратном направлении от узлов выше в сети, как и проиллюстрировано на рисунке 6. Отметим, что суммирование в (4.13) ведется по первому индексу ω_{kj} (соответствуя обратному распространению данных по сети), тогда как в уравнении прямого распространения (2.10) оно проходит по второму индексу. Поскольку мы уже знаем значения δ для выходных узлов отсюда следует, что, рекурсивно применяя (4.13), мы можем рассчитать δ для всех скрытых узлов в сети прямого распространения, независимо от ее топологии. Расчет δ_i представлен на рисунке 10.

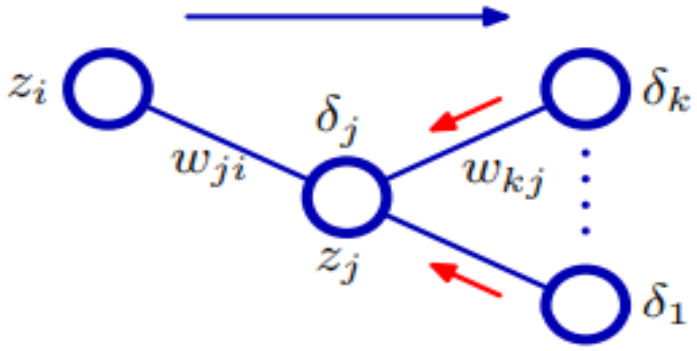

Рисунок 10 – Иллюстрация расчета δ_j для скрытых узлов j с помощью обратного распространения δ от единиц k, к которым блок j отправляет соединения. Синяя стрела обозначает направление данных при прямом распространении, а красные стрелки указывают на обратное распространение данных ошибки

Таким образом процедуру обратного распространения можно резюмировать следующим образом

1. Примените входной вектор x_n к сети и распространите его вперед по сети, используя (4.4) и (4.5), чтобы найти активации всех скрытых и входных узлов.

2. Рассчитайте δ_k для всех выходных узлов, используя (4.11)

3. Выполните обратное распространение δ_k, используя (4.13), чтобы получить δ_j для каждого скрытого узла в сети.

4. Используйте (4.10) для расчета требуемых производных.

Для пакетных методов производная полной ошибки E может быть затем получена путем повторения вышеуказанных шагов для каждого шаблона в обучающем наборе и последующего суммирования через все шаблоны.

$$\frac{\partial E}{\partial \omega_{ji}} = \sum_n \frac{\partial E_n}{\partial \omega_{ji}} \tag{4.13}$$

В приведенном выше выводе мы неявно предполагали, что каждый скрытый и входной узел в сети имеет одну и ту же функцию активации $h(\cdot)$. Однако вывод легко обобщается, что позволит различным узлам иметь индивидуальные функции активации, просто отслеживая, какая форма $h(\cdot)$ соответствует какому узлу.

3 Вероятностный подход в машинном обучении

3. 1 Применение вероятностей в машинном обучении

Глубокое обучение привело к революции в машинном обучении, предоставляя решения для проблем, которые аналитически человеку трудно решить. Однако модели глубокого обучения склонны к переобучению, явление, когда модель хорошо себя показывает на обучающей выборке, но плохо работает на примерах, на выборке для тестирования. Так же модели склонны быть слишком самоуверенными в своих прогнозах, когда они обеспечивают доверительный интервал. Это проблематично для приложений, где такие сбои могут привести к драматическим последствиям, например, автономное вождение, медицинские диагностики или финансы [11].

Следовательно, многие подходы были предложены для снижения этого риска. Среди них байесовская парадигма обеспечивает строгую основу для анализа и обучения нейронной сети, учитывающие неопределенность и многое другое. Байесовская парадигма основана на двух простых идеях. Во-первых, вероятность есть мера веры в событие, а не предел частоты возникновения, когда количество выборок стремится к бесконечности. Во-вторых, идея заключается в том, что убеждения до (априорное распределение) влияют на убеждения после (апостериорное распределение). Байесовская парадигма не только предлагает надежный подход к количественной оценке неопределенности в моделях глубокого обучения, но также обеспечивает математическую основу для понимания стратегий обучения и многих методов регуляризации (добавление ограничений к условию задачу, чтобы решить некорректно поставленную задачу или предотвратить переобучение), которые используются в классических нейронных сетях [12-13].

Байесовские нейронные сети являются стохастическими нейронными сетями, обученные с использованием байесовского подхода.

Цель нейронной сети представить произвольную функцию $y = \Phi(x)$. Традиционные нейронные сети прямой связи и рекуррентные сети построены с

использованием одного входного слоя l_0, последовательности скрытых слоев l_i, $i = 1, \dots, n-1$ и выходного слоя l_n, где $n+1$ количество слоев. В простейшей архитектуре сетей прямого распространения каждый слой l представляется как линейное преобразование, за которым следует нелинейная операция s, также называемая функцией активации, получим набор формул (5.1):

$$\begin{gathered} l_0 = x, \\ l_i = s_i(W_i l_{i-1} + b_i) \ \ \forall i \in [1, n] \\ y = l_n \end{gathered} \tag{5.1}$$

Чтобы спроектировать байесовскую нейронную сеть первым шагом нужно выбрать архитектуру, то есть функциональную модель. Затем нужно выбрать стохастическую модель, т. е. априорное распределение по возможной параметризации модели $p(\theta)$ и априорное уверенность в предсказанной силе модели $p(y|x, \theta)$. На рисунке 11 визуально представлена разница между двумя нейронными сетями.

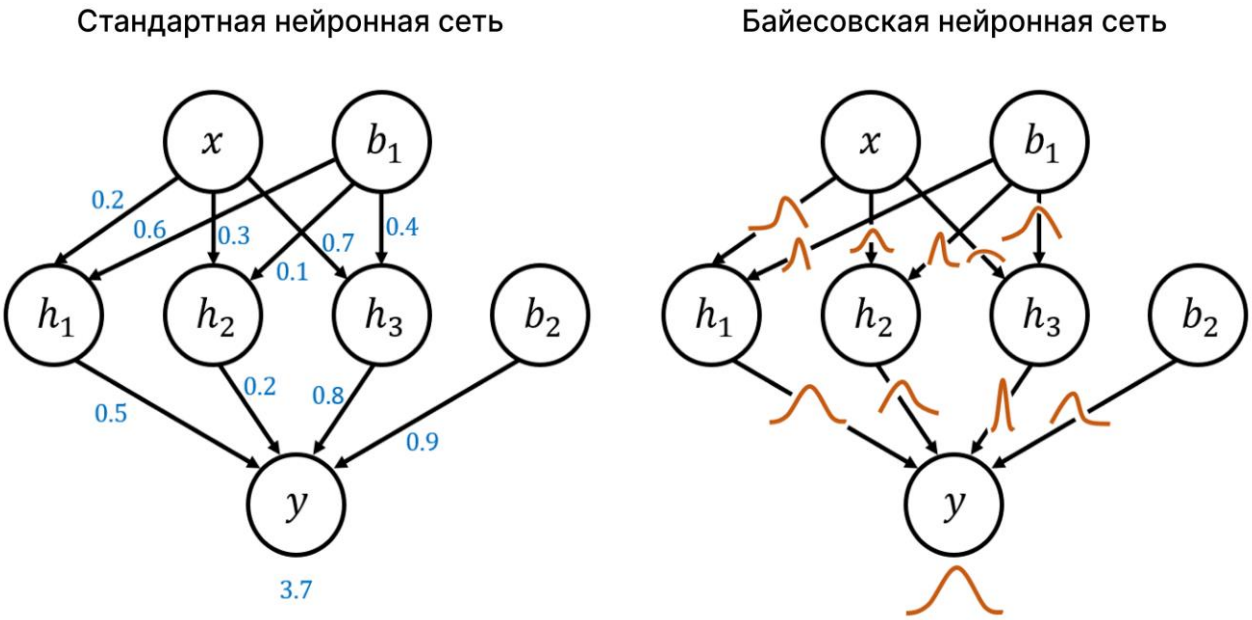

Рисунок 11 – Пример архитектуры стандартной нейронной сети и байесовской нейронной сети

Параметризацию модели можно рассматривать гипотезу H, а обучающим набором данные D. Выбор стохастической модели байесовской нейронной сети в чем-то эквивалентен выбору функции потерь при обучении нейронной сети с точечной оценкой. Обучающие данные D разделим на входные данные D_x и обучающие метки D_y. Применяя теорему Байеса и добиваясь независимости между параметрами модели и входными данными, байесовскую апостериорную вероятность можно записать как формулу (5.2)

$$p(\theta, D) \propto p\left(D_y \middle| D_x, \theta\right) p\ (\theta) \tag{5.2}$$

Байесовская апостериорную вероятность представляет собой многомерное и невыпуклое распределение вероятностей. Эта сложность делает вычисление и выборку с использованием стандартных методов неразрешимой проблемой, особенно потому, что вычисление доказательств $\int_\theta p\left(D_y \middle| D_x, \theta'\right) p d\theta'$ трудоемкое [14-15]. Чтобы решить эту проблему были введены два широких подхода: цепь Маркова Монте-Карло (в частности, мы рассмотрим один из алгоритмов No-U-Turn или «Нет разворота») и вариационный вывод (мы рассмотрим Automatic Differentiation Variational Inference или Вариационный вывод автоматического дифференцирования).

При использовании байесовской нейронной сети для прогнозирования особый интерес представляет распределение вероятностей $p(y|x, D)$, которое количественно определяет неопределённость модели при ее прогнозировании [16]. Учитывая $p(\theta, D)$ и $p(y|x, D)$ могут быть рассчитаны как в формуле (5.3)

$$p(y|x, D) = \int_\theta p(y|x, \theta') p(\theta'|D) d\theta' \tag{5.3}$$

На практике $p(y|x, D)$ выбирается косвенно с использование уравнения (5.4)

$$\begin{gathered} \theta \sim p(\theta), \\ y = \Phi_\theta(x) + \epsilon \end{gathered} \tag{5.4}$$

Окончательный прогноз можно обобщить по статистике, рассчитанной с использование метода Монте-Карло. Большой набор весов θ_i выбирается из

апостериорного распределения и используется для вычисления ряда возможных выходных значений y_i.

Обычно для этих выборок вычисляются агрегаты, чтобы суммировать неопределённость байесовской нейронной сети и получить оценку для выходных данных y. Эта оценка обозначается через $\hat{y}$. Опишем ее формулой (5.5)

$$\hat{y} = \frac{1}{\Theta} \sum_{\theta_i \in \Theta} \Phi_{\theta_i}(x) \tag{5.5}$$

где Θ – выборка из распределения $\mathrm{p}(\theta, D)$.

При выполнении классификации средний прогноз модели даст относительную вероятность каждого класса, которую можно считать мерой неопределенности [17], описанную формулой (5.6)

$$\hat{p} = \frac{1}{\Theta} \sum_{\theta_i \in \Theta} \Phi_{\theta_i}(x) \tag{5.6}$$

Окончательный прогноз принимается за наиболее вероятный класс.

Одним из основных критических замечаний по поводу байесовских методов является то, что они полагаются на предшествующие знания [18]. Это особенно верно в отношении глубокого обучения, поскольку получить какое-либо представление о правдоподобной параметризации для данной модели до обучения очень сложно.

3. 2 Алгоритм NUTS

Гамильтониан Монте-Карло (HMC) — это алгоритм Монте-Карло с цепями Маркова (MCMC), который позволяет избежать поведения случайного блуждания и чувствительности к коррелированным параметрам, от которых страдают многие Методы MCMC путем выполнения ряда шагов, основанных на информации о градиенте первого порядка. Эти функции позволяют ему гораздо больше сходиться к многомерным целевым распределениям быстрее, чем более простые методы, такие как случайное блуждание по Метрополису или выборка Гиббса. Однако, производительность HMC очень чувствительна к двум заданным пользователем параметрам: размеру шага и желаемому количеству

шагов L. В частности, если L слишком мало, алгоритм демонстрирует нежелательное поведение случайного блуждания, в то время как, если L слишком велико, алгоритм тратит вычисления впустую.

No-U-Turn Sampler (NUTS) [19], расширение HMC, которое устраняет необходимость задавать количество шагов L. NUTS использует рекурсивный алгоритм для построения набора вероятных точки-кандидаты, которые охватывают широкий диапазон целевого распределения, автоматически останавливаясь, когда он начинает удваивать назад и повторять свои шаги. Эмпирически NUTS работает, по крайней мере, как эффективно, как (а иногда и более эффективно, чем) хорошо настроенный стандартный метод HMC, без вмешательства пользователя или дорогостоящей настройки. Мы также выводим метод для адаптации параметра размера шага на лету на основе первичного-двойного усреднения. Таким образом, его можно использовать вообще без ручной настройки.

Алгоритм NUTS реализуется через блуждание точки начиная с позиции θ. Для начала задается целевое распределение, описанное формулой (5.7), за предел которых $\tilde{\theta}$ не должно выходить

$$\exp\left\{\mathcal{L}(\theta) - \frac{1}{2} r \cdot r\right\} \tag{5.7}$$

где $\mathcal{L}(\theta) = V = -\ln p(\theta)$. Воспользуемся методом Стёрмера-Верле для блуждания точки

$$\begin{gathered} r^{\left\{t+\frac{\epsilon}{2}\right\}} = r^t + \left(\frac{\epsilon}{2}\right) \nabla_\theta \mathcal{L}(\theta^t), \\ \theta^{\{t+\epsilon\}} = \theta^t + \epsilon r^{\left\{t+\frac{\epsilon}{2}\right\}}, \\ r^{\left\{t+\frac{\epsilon}{2}\right\}} = r^{\left\{t+\frac{\epsilon}{2}\right\}} + \left(\frac{\epsilon}{2}\right) \nabla_\theta \mathcal{L}(\theta^{\{t+\epsilon\}}) \end{gathered} \tag{5.8}$$

в формуле (5.8) мы находим новые положения θ и новый импульс r. Так же мы должны абсолютно случайно и равнозначно выбирать направление движения при помощи формулы (5.9)

$$u_j = Uniform(\{-1, 1\}) \tag{5.9}$$

При достижении условия (5.10)

$$\left(\tilde{\theta} - \theta\right) \cdot \tilde{r} \leq 0 \tag{5.10}$$

работа алгоритма прекращается. Данное условие нам говорит о том, что блуждание точки начинает идти в обратном направлении, что нам не нужно.

Теперь мы должны получить наше новое распределение вероятности при помощи формулы (5.11)

$$p(\theta, r) = C \cdot \exp(-H) \tag{5.11}$$

где $H = \frac{1}{2} r \cdot r + V$ – функция Гамильтона, а C – нормировочная постоянная распределения. На рисунке 12 представлена краткая блок-схема алгоритма, а на рисунке 13 его результат.

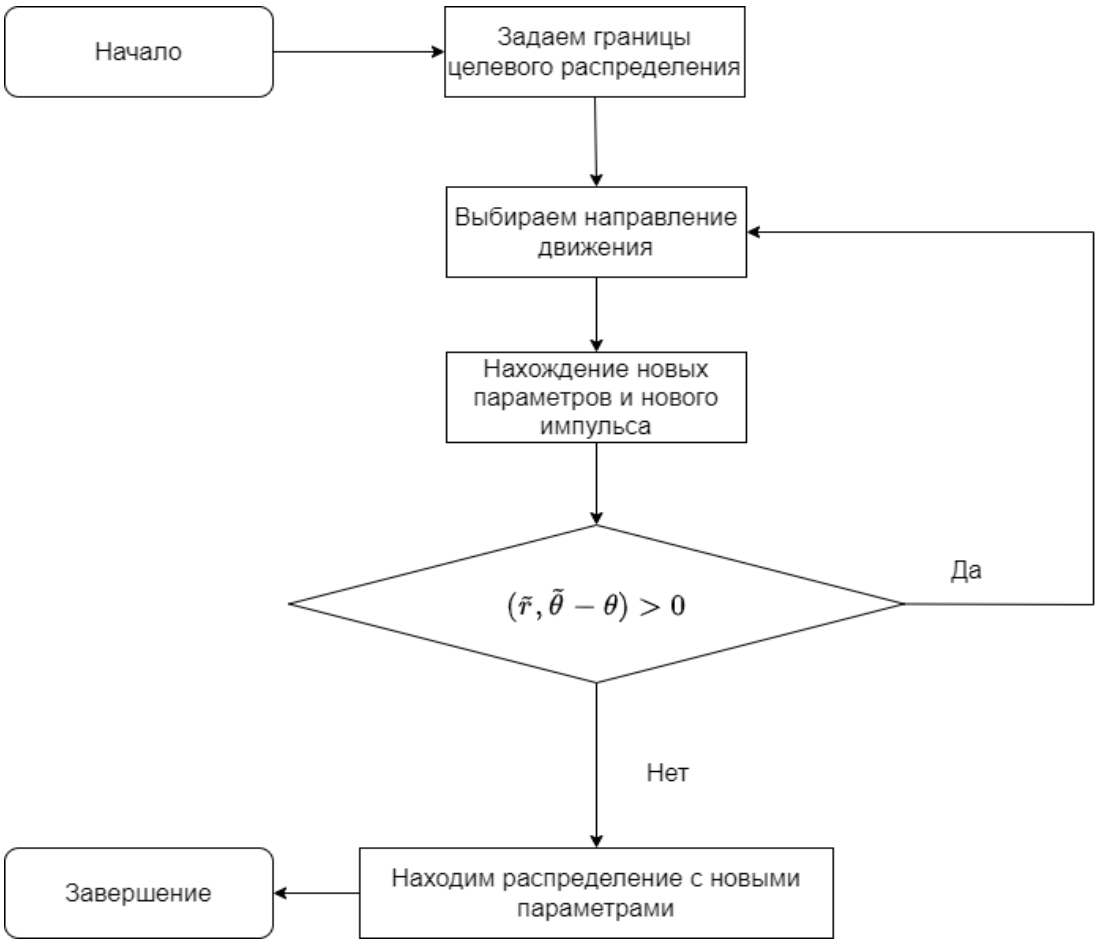

Рисунок 12 – Блок-схема алгоритма NUTS

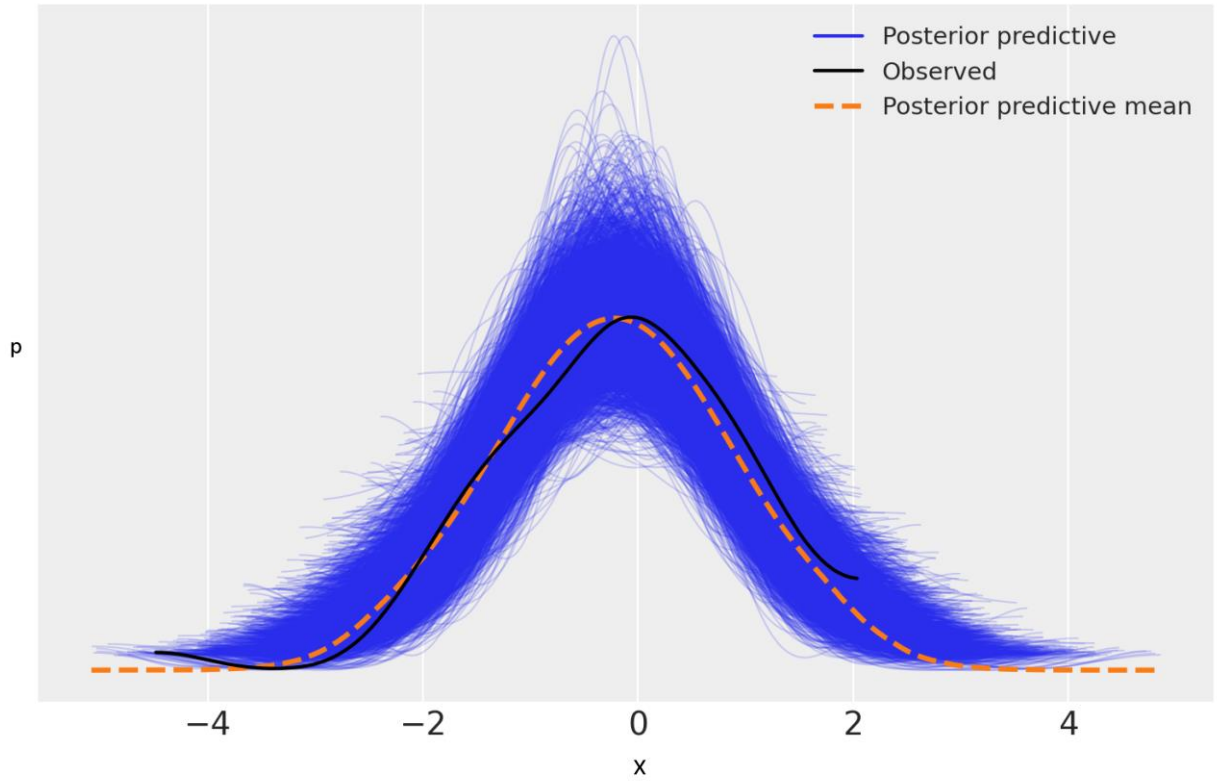

Рисунок 13 – Результат работы алгоритма NUTS

3. 3 Алгоритм AVDI

Алгоритмы MCMC являются лучшими инструментами для выборки из точных апостериорных данных. Однако отсутствие масштабируемости сделало их менее популярными для байесовский нейронных сетей, учитывая размер рассматриваемых моделей. Variational Inference (VI) [20] или вариационный вывод, который масштабируется лучше, чем алгоритмы MCMC, приобрел значительную популярность. Вариационный вывод не является точным методом. Вместо того, чтобы допускать выборку из точного апостериорного распределения, идея состоит в том, чтобы иметь распределение $q_\phi(H)$, называемое вариационным распределением, параметризованное набором параметров ϕ. Затем значения параметров ϕ изучаются так, чтобы вариационное распределение $q_\phi(H)$ было как можно ближе к точному апостериорному $p(H|d)$. Обычно используется мера близости вычисляемая по формуле (5.12)

$$ELBO(\phi) = E_{q(\theta;\phi)}[\log p(data, \theta) - \log q(\theta;\ \phi)] \quad (5.12)$$

где $E_{q(\theta;\phi)}$ мера близости дивергенция Кульбака-Лейблера (KL-дивергенция), которая измеряет различия между распределениями вероятностей на основе теории информации Шеннона. Наша задача сделать так, чтобы два вероятностных распределения были максимально близки друг к другу. Алгоритм использует различные оптимизации, такие как аппроксимации, подбор более оптимальных распределений q. На рисунке 14 представлен результат работы алгоритма.

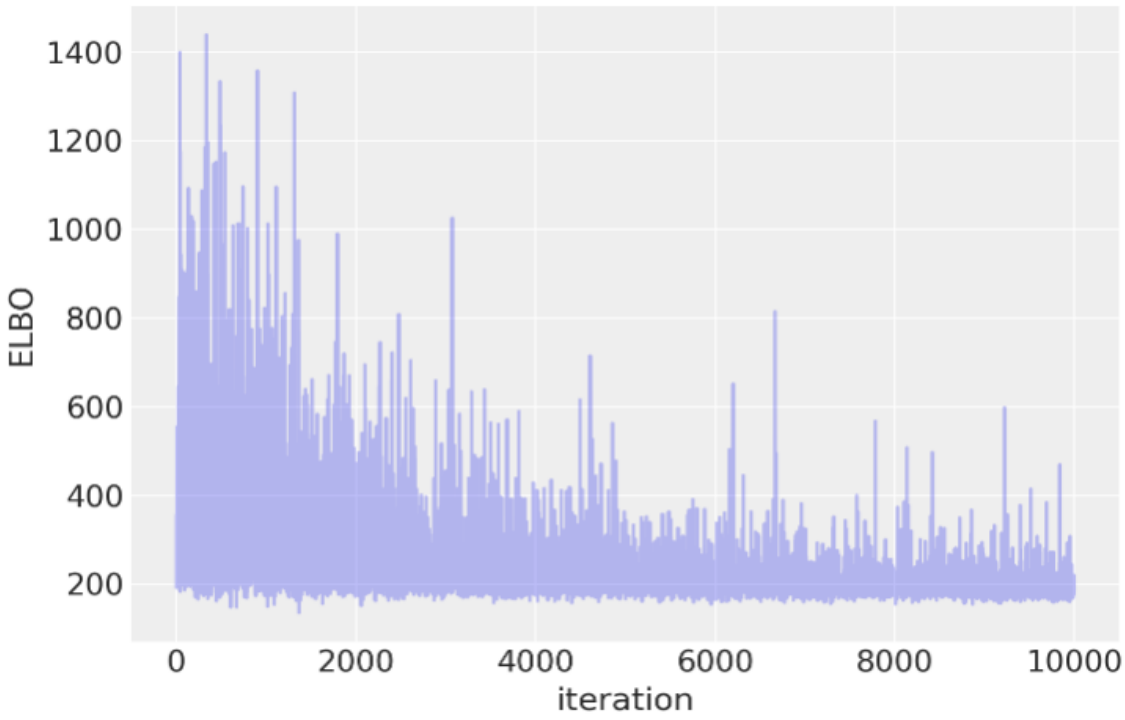

Рисунок 14 – Результат работы алгоритма NUTS. Чем меньше значение ELBO, тем точнее будет работать наша модель

3. 4 Сравнение двух подходов

Для того, чтобы сравнить два подхода мы должны создать две нейронных сети, классическую и байесовскую с одинаковыми архитектурами. После этого взять равное количество данных, обучить по ним наши нейронные сети и провести прогнозирование, в нашем случае классификацию.

Начнем с классической нейронной сети. В большинстве своем мы будем использовать две библиотеки для программирования Python – Keras и Sklearn.

Зададим выборку данных, разделим ее на данные для обучения и для тестирования.

```
X, Y = make_moons(noise=0.2, random_state=0, n_samples=1000)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5)
```

Зададим архитектуру нашей нейронной сети с двумя нейронами во входном слое, два слоя с пятью нейронами в каждом, с функцией активацией «гиперболический тангенс» и выходной слой в один нейрон с «сигмоидальной» функцией активации.

```
model = Sequential()
model.add(Dense(2, input_dim=2))
model.add(Dense(5, activation='tanh'))
model.add(Dense(5, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics= ['accuracy'])
```

Обучим нашу классическую нейронную сеть.

```
results = model.fit(X_train, Y_train, epochs=10000, verbose=0)
```

Сделаем прогноз, то есть классифицируем тестовые данные и выведем результат, отображенный на рисунке 15.

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.17777690291404724
Test accuracy: 0.949999988079071
Wall time: 106 ms
```

Рисунок 15 – Результат работы классической нейронной сети

Теперь перейдем к построению байесовской нейронной сети. Для нее мы будем использовать две библиотеки, для построения модели Pymc3, для анализа результатов и самой модели будем использовать Arviz на языке программирования Python.

Так же, как и в прошлом примере для начала создадим выборку данных и выделим в том же соотношению, как и прошлый раз, данные для обучения и для тестирования.

```
X, Y = make_moons(noise=0.2, random_state=0, n_samples=1000)
X = scale(X)
X = X.astype(floatX)
Y = Y.astype(floatX)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5)
```

Отобразим наши исходные данные. На рисунке 16 представлен результат программного кода.

```
fig, ax = plt.subplots()
ax.scatter(X[Y == 0, 0], X[Y == 0, 1], color="C0", label="Класс 0")
ax.scatter(X[Y == 1, 0], X[Y == 1, 1], color="C1", label="Класс 1")
sns.despine()
ax.legend()
ax.set(xlabel="X", ylabel="Y", title="Исходные данные");
```

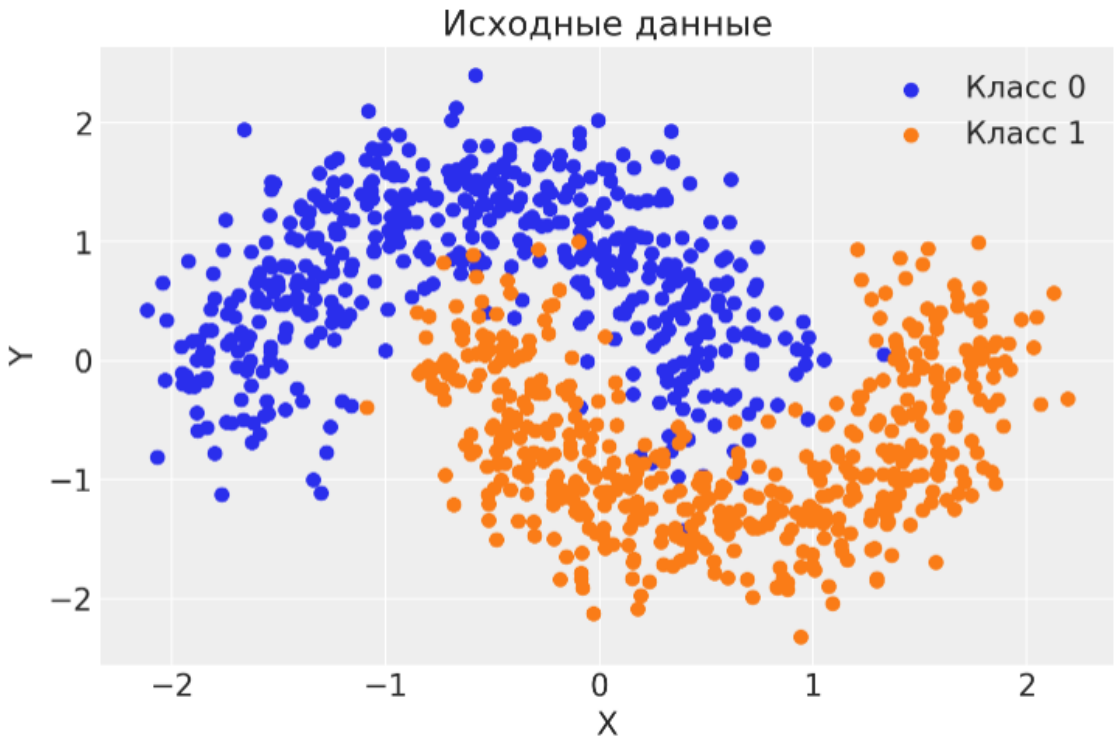

Рисунок 16 – Исходные данные для классификации

Построим нашу байесовскую нейронную сеть. Архитектура будет идентичная с единственным отличием, вместо весов мы будем использовать вероятностные распределения. Для двух скрытых слоев и входного слоя мы будем использовать нормальное распределение, для выходного слоя мы будем использовать распределение Бернулли (так как мы будем выполнять задачу классификации и нам нужен ответ вида 0/1 или).

```
def construct_nn(ann_input, ann_output):
    n_hidden = 5

    init_1 = rng.standard_normal(size=(X_train.shape[1], n_hidden)).astype(floatX)
    init_2 = rng.standard_normal(size=(n_hidden, n_hidden)).astype(floatX)
    init_out = rng.standard_normal(size=n_hidden).astype(floatX)

    coords = {
        "hidden_layer_1": np.arange(n_hidden),
```

```
        "hidden_layer_2": np.arange(n_hidden),
        "train_cols": np.arange(X_train.shape[1]),
        "obs_id": np.arange(X_train.shape[0]),
    }
    with pm.Model(coords=coords) as neural_network:
        ann_input = pm.Data("ann_input", X_train)
        ann_output = pm.Data("ann_output", Y_train)



        weights_in_1 = pm.Normal(
            "w_in_1", 0, sigma=1, testval=init_1, dims=("train_cols", "hidden_layer_1")
        )
        weights_1_2 = pm.Normal(
            "w_1_2", 0, sigma=1, testval=init_2, dims=("hidden_layer_1",
"hidden_layer_2")
        )
        weights_2_out = pm.Normal("w_2_out", 0, sigma=1, testval=init_out,
dims="hidden_layer_2")

        act_1 = pm.math.tanh(pm.math.dot(ann_input, weights_in_1))
        act_2 = pm.math.tanh(pm.math.dot(act_1, weights_1_2))
        act_out = pm.math.sigmoid(pm.math.dot(act_2, weights_2_out))

        out = pm.Bernoulli(
            "out",
            act_out,
            observed=ann_output,
            total_size=Y_train.shape[0],
            dims="obs_id",
```

```
        )
    return neural_network
neural_network = construct_nn(X_train, Y_train)
```

Используя алгоритм AVDI, который мы рассматривали до этого, оптимизируем и создадим экземпляры нашего вероятностного распределения. Так как до этого мы задали наши данные, то есть $p(D|H)$, то алгоритм будет пытаться подстроить вероятностное распределение под эти данные и нашу гипотезу.

```
with neural_network:
    inference = pm.ADVI()
    approx = pm.fit(method=inference, n=10000)
```

Результат алгоритма можно просмотреть на рисунке 12. Проведем задачу классификации на данных для тестирования. На рисунке 17 покажем график, как наша модель видит данные, как и с какой вероятностью она относит точки к конкретному классу.

```
pred = sample_proba(X_test, 500).mean(0) > 0.5
```

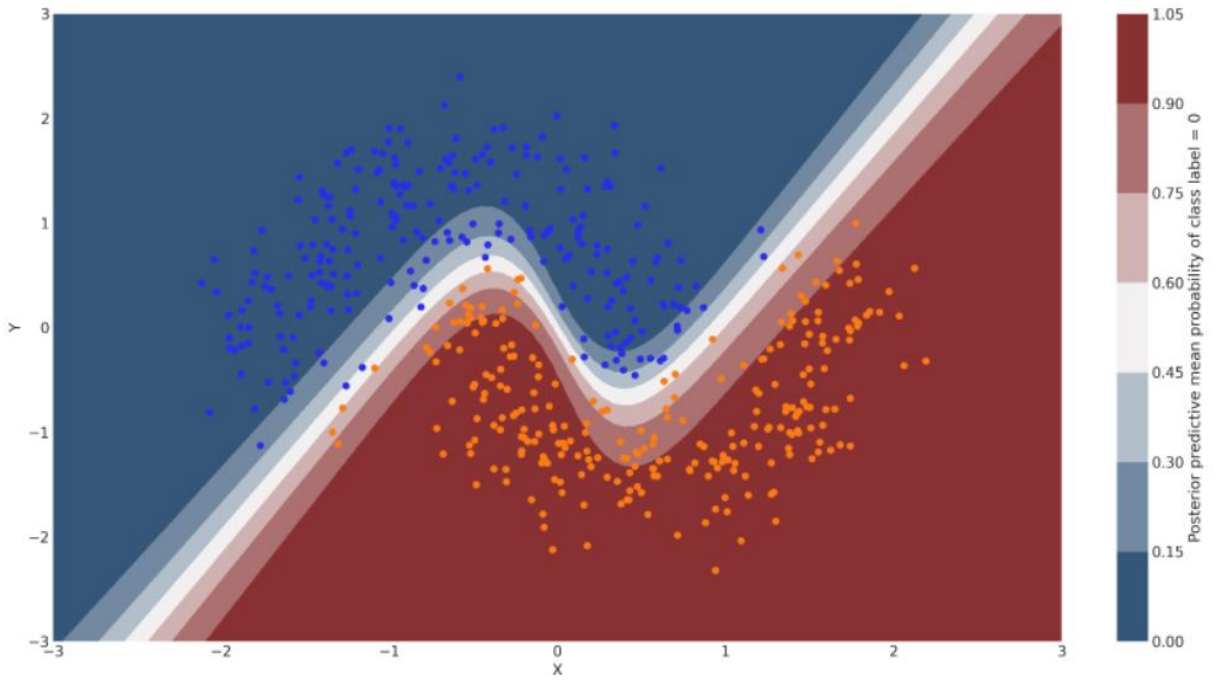

Рисунок 17 – Результат классификации байесовской нейронной сети. Шкала справа показывает степень уверенности нейронной сети в классификации

Две рабочие нейронных сети готовы. Теперь проведем их сравнение по количеству итераций, скорости работы и точности. Получим следующие результаты, представленные в таблице 1, введем обозначения – стандартная нейронная сеть (СНС) и байесовская нейронная сеть (БНС).

Таблица 1 – Сравнение стандартной нейронной сети и байесовской нейронной сети

	СНС	БНС	СНС	БНС	СНС	БНС	БНС
Количество итераций	500	500	1000	1000	10000	10000	50000
Время (с)	4.6	1.2	9.4	1.4	95.0	6.0	20.0
Точность (%)	85.7	85.2	95.4	88.2	95.0	88.2	95.2

Как мы можем видеть, байесовская нейронная сеть показывает свою эффективность благодаря алгоритму AVDI на данном типе данных. Как мы уже отметили байесовская нейронная сеть может быть крайне полезна в некотором спектре задач, и так как вероятностный подход в машинном обучении относительно новый, с дальнейшем развитием алгоритмов и библиотек ситуация может только улучшится.

Заключение

При выполнении данной выпускной квалификационной работы были решены следующие задачи:

1. При рассмотрении сгенерированного временного ряда, для которого требуется решить задачу классификации, были использованы одна или несколько переменных.

2. Были реализованы две модели для решения задач классификации: стандартная нейронная сеть и байесовская нейронная сеть. Байесовская нейронная сеть показала, что ей требуется гораздо меньше времени и вычислительных ресурсов для достижения той же точности, что и стандартная нейронная сеть. Байесовская нейронная сеть будет гораздо лучше справляться во временных рядах с большой неопределенностью, используя вероятностный подход.

3. Отметим, что байесовская нейронная сеть представляет интерес для решения задач классификации для экономических прогнозов.

Список использованных источников

1 Эйлин, Н. Практический анализ временных рядов / Н. Эйлин; пер. с анг. Д. А. Клюшина. – Москва: Диалектика, 2021. - 538 с.

2 Гнеденко, Б.В. Курс теории вероятностей / Б. В. Гнеденко. — Москва: Наука, 1988. - 320 с.

3 Гмурман, В. Е. Теория вероятностей и математическая статистика: учеб. пособие для вузов / В. Е. Гмурман. – М.: Высш. образование, 2006. - 479 с.

4 Мартин, О. Байесовский анализ на Python / О. Мартин; пер. с анг. А. В. Снастина. – Москва: ДМК Пресс, 2020. - 339 с.

5 Дауни, А. Б. Байесовские модели / А. Б. Дауни; пер. с анг. В. А. Яроцкого. – Москва: ДМК Пресс, 2018. - 181 с.

6 Паттерсон, Д. Глубокое обучение с точки зрения практика / Д. Паттерсон, А. Гибсон; пер. с англ. А. А. Слинкина. – Москва: ДМК Пресс, 2018. - 415 с.

7 Bishop, C. M. Pattern Recognition and Machine Learning / C. M. Bishop – New York: Springer New York, 2006. - 738 p.

8 Bisong, E. Building Machine Learning and Deep Learning Models on Google Cloud Platform / E. Bisong – Berkeley, CA: Apress, 2019. – 709 p.

9 Гудфеллоу, Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль; пер. с англ. А. А. Слинкина – Москва: ДМК Пресс, 2018. - 651 с.

10 Микелуччи, У. Прикладное глубокое обучение / У. Микелуччи; пер. с англ. А. Логунова. - Санкт-Петербург: БХВ-Петербург, 2020. - 368 с.

11 Gelman, A. Bayesian Data Analysis / A. Gelman, J. B. Carlin, H. S. Stern - Oxford, United States: CRC Press, 2013. – 675 p.

12 McElreath, R. Statistical Rethinking / R. McElreath – Florida, United States: Chapman and Hall/CRC Press, 2018 – 612 p.

13 Kruschke, J. K. Doing Bayesian Data Analysis / J. K. Kruschke – Massachusetts, United States: Academic Press, 2010 – 648 p.

14 Downey, A. B. Think Bayes / A. B. Downey – Massachusetts, United States: O'Reilly Media, 2013 – 214 p.

15 Rasmussen, C. E. Gaussian Processes for Machine Learning / C. E. Rasmussen, C. K. I. Williams – Massachusetts, United States: MIT Press, 2006 – 216 p.

16 Lecun, Y. Gradient-based learning applied to document recognition / Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner // Proceedings of the IEEE. – 1998. - № 11. - P. 2278–2324.

17 Belkin, M. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples / M. Belkin, P. Niyogi, V. Sindhwani // Journal of Machine Learning Research. – 2006. - № 7. - P. 2399–2434.

18 Patil, D. H. PyMC: Bayesian stochastic modelling in python / Patil, D. Huard, C. Fonnesbeck // Journal of Statistical Software. – 2010 - № 35. – P.1–81.

19 Gelman, A. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo / A. Gelman // Journal of Machine Learning Research. – 2014. – №15. – P. 1593-1623.

20 Kucukelbir, A. Automatic Differentiation Variational Inference / A. Kucukelbir // Journal of Machine Learning Research. – 2017. - №18. – P. 1-45.