Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

Факультет «Радиоэлектроника и лазерная техника»

Кафедра «Технологии приборостроения»

Проект на тему:

«Предиктор ВАХ слоистой квантоворазмерной структуры на основе искусственной нейронной сети»

Выполнил:

Веселенко Н. Ю.

Группа РЛ6-71Б

Проверил: доцент кафедры РЛ6, к. т. н.

Ветрова Н. А.

Москва, 2020

Оглавление

Введение

Цель работы: предсказание вольтамперных характеристик слоистой квантов размерной структуры на основе искусственной нейронной сети или получение нелинейной зависимости (графика ВАХ слоистой квантов размерной структуры) при помощи ИНС.

Задачи работы:

1. Построение модели искусственной нейронной сети.
2. Обучение искусственной нейронной сети на тренировочных данных и получение ВАХ на тестовых.

Актуальность: почти во всех сферах жизни активно интегрируются нейронные сети и используется машинное обучения, наноэлектроника не исключение.

1. Искусственная нейронная сеть

1.1 Что такое нейронная сеть?

Нейронная сеть (также **искусственная нейронная сеть, ИНС**) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. [1]

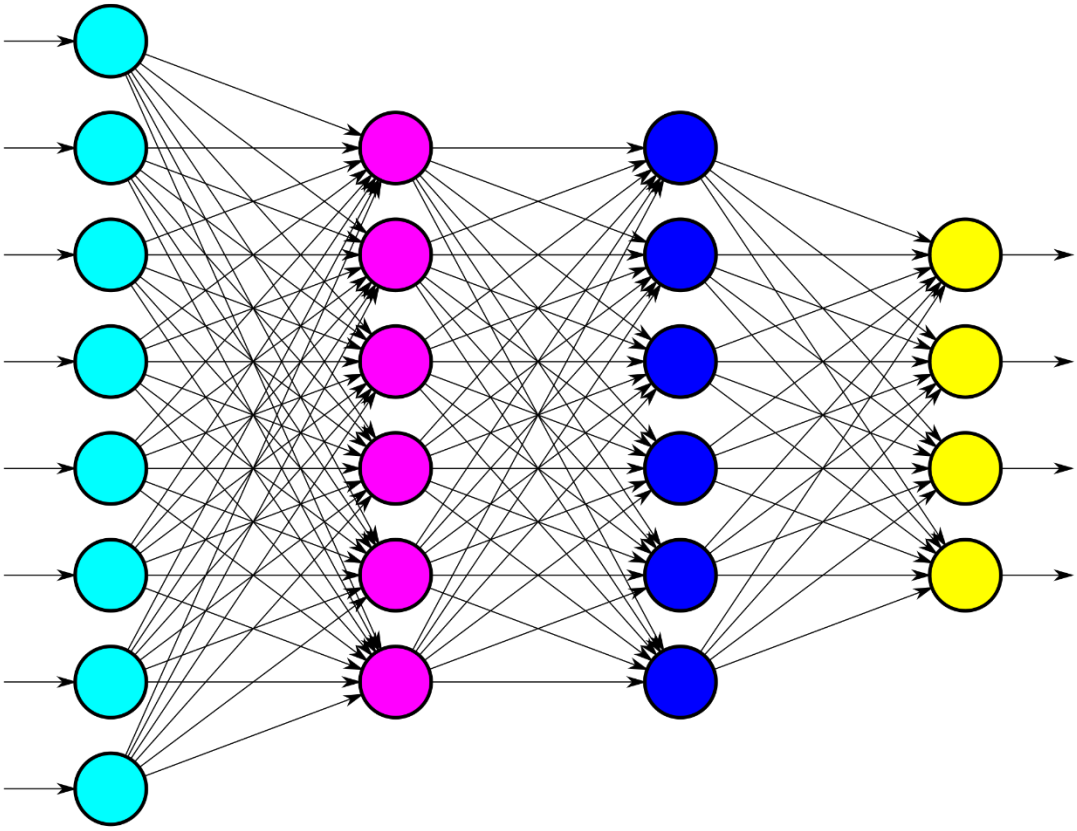

Рисунок 1 – Графическое представление нейронной сети

Так же можно сформулировать другое определение нейронной сети исходя из рисунка 1: нейронная сеть — это последовательность нейронов, соединенных между собой синапсами. Нейроны – это разноцветные окружности, синапсы – связи между нейронами.

1.2 В каких задачах используются нейронные сети?

Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Самыми распространенными применениями нейронных сетей является:

- **Классификация** — распределение данных по параметрам.

Классическим примером является задача по классификации ирисов Фишера, где основными параметрами цветков являлись: 1) длина наружной доли околоцветника; 2) ширина наружной доли околоцветника; 3) длина внутренней доли околоцветника; 4) ширина внутренней доли околоцветника.

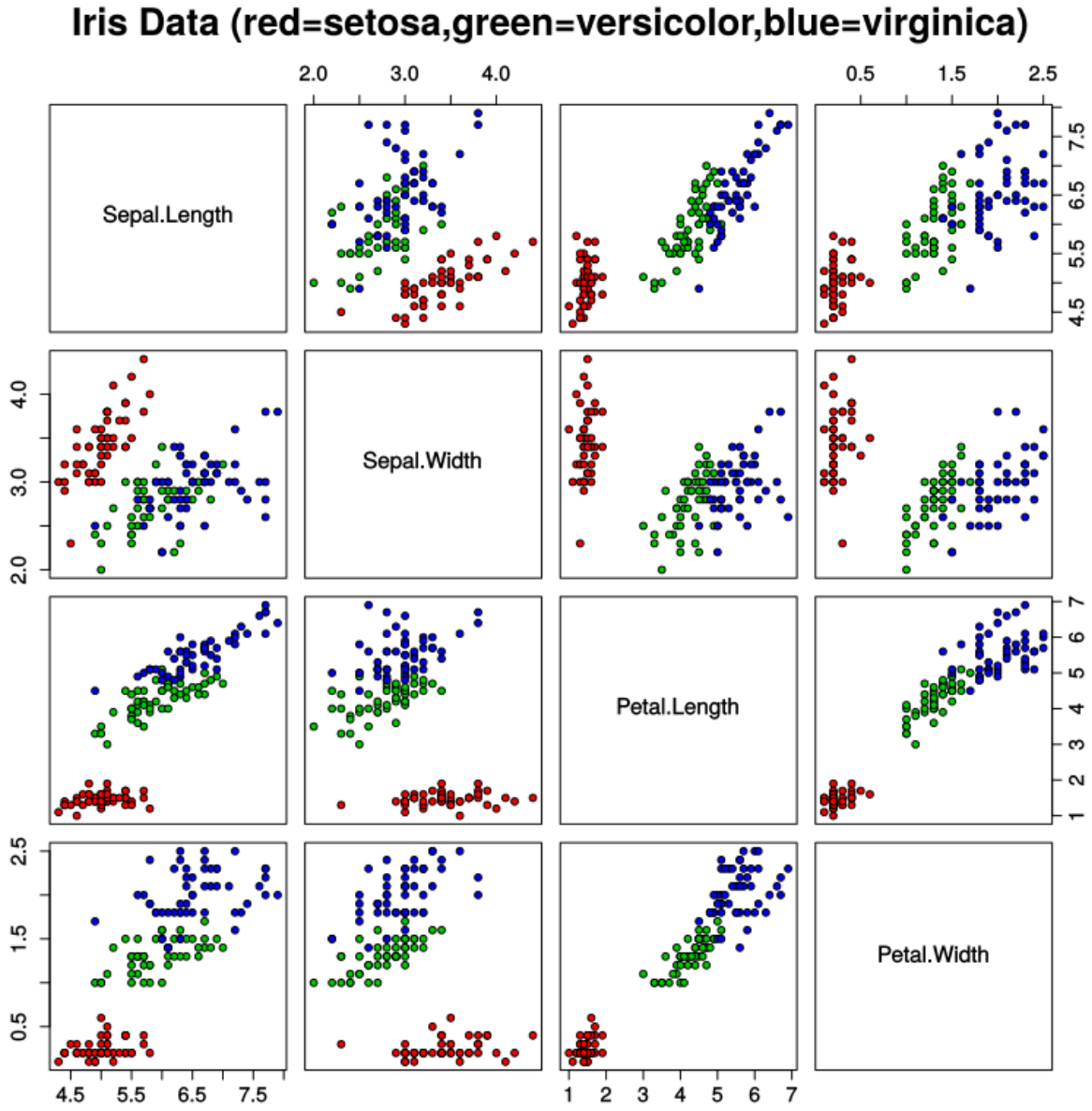

Рисунок 2 – Классификация ирисов Фишера

- **Предсказание** — возможность предсказывать следующий шаг.

Например, рост или падение акций, основываясь на ситуации на фондовом рынке.

- **Распознавание** — в настоящее время, самое широкое применение нейронных сетей.

Классическая задача по распознавания цифр с изображения.

1.3 Структура нейронной сети

В общем виде каждая нейронная сеть представляет из себя нейрон и синапс.

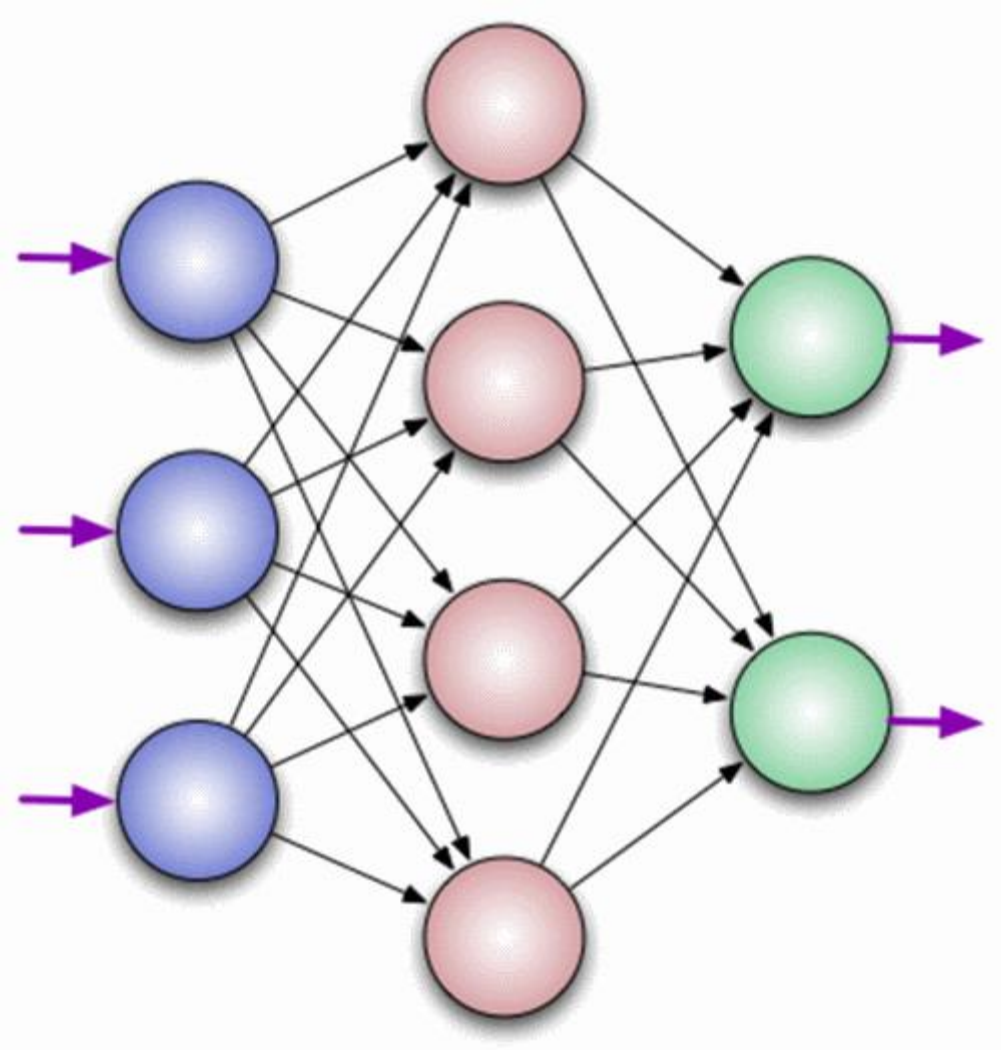

Рисунок 3 – Трехслойная нейронная сеть

- **Нейрон** - это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Все нейроны делятся на 3 типа: 1) входные (синие на рисунке 3); 2) выходные (зеленые на рисунке 3) и 3) скрытые (розовые на рисунке 3). Нейроны одного типа образующие кластер называются – слоем. Количество скрытых слоев может изменяться от 0 до 3 (чаще всего не более 3) в зависимости от задачи. Так же конфигурируется количество нейронов в каждом слое. Каждый нейрон может получить на вход данные(сигналы) и отдавать на выход. Так же нейрон несет в

себе математическую функцию активации, о которой будет рассказано позднее.

- **Синапс** - это связь между двумя нейронами. У синапсов есть 1 параметр — весовой коэффициент или вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Значение весов варьируется от -1 до 1, но чаще от -0.99 до 0.99 и формируется на моменте создания нейронной сети случайным образом. Стоить отметить что синапсы имеются только между входным-скрытым, скрытым-скрытым, скрытым-выходным слоями. Входной слой получает данные извне, выходной отдает сигналы.

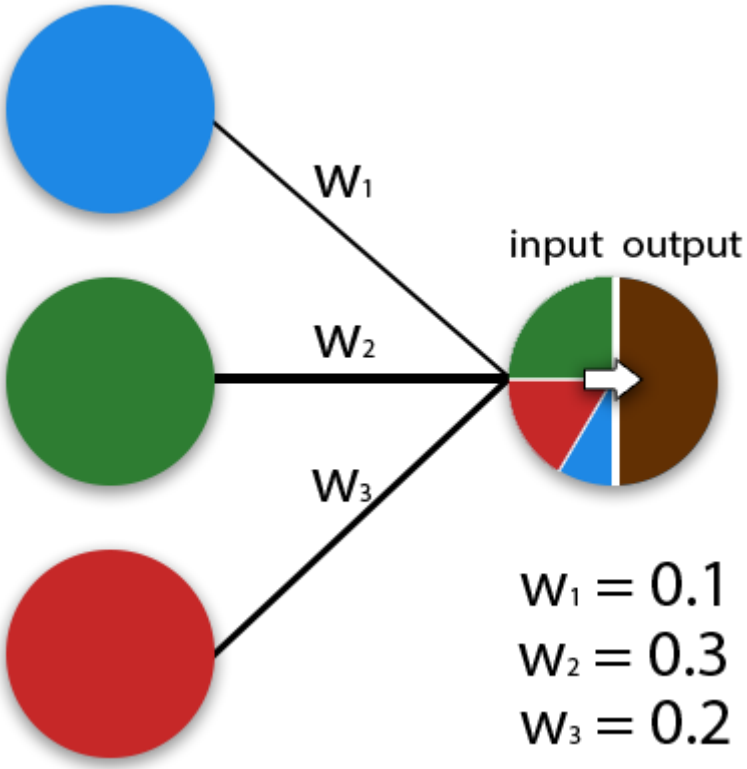

Рисунок 4 – Отображение весовых коэффициентов синапсов

1.4 Принцип работы нейронной сети

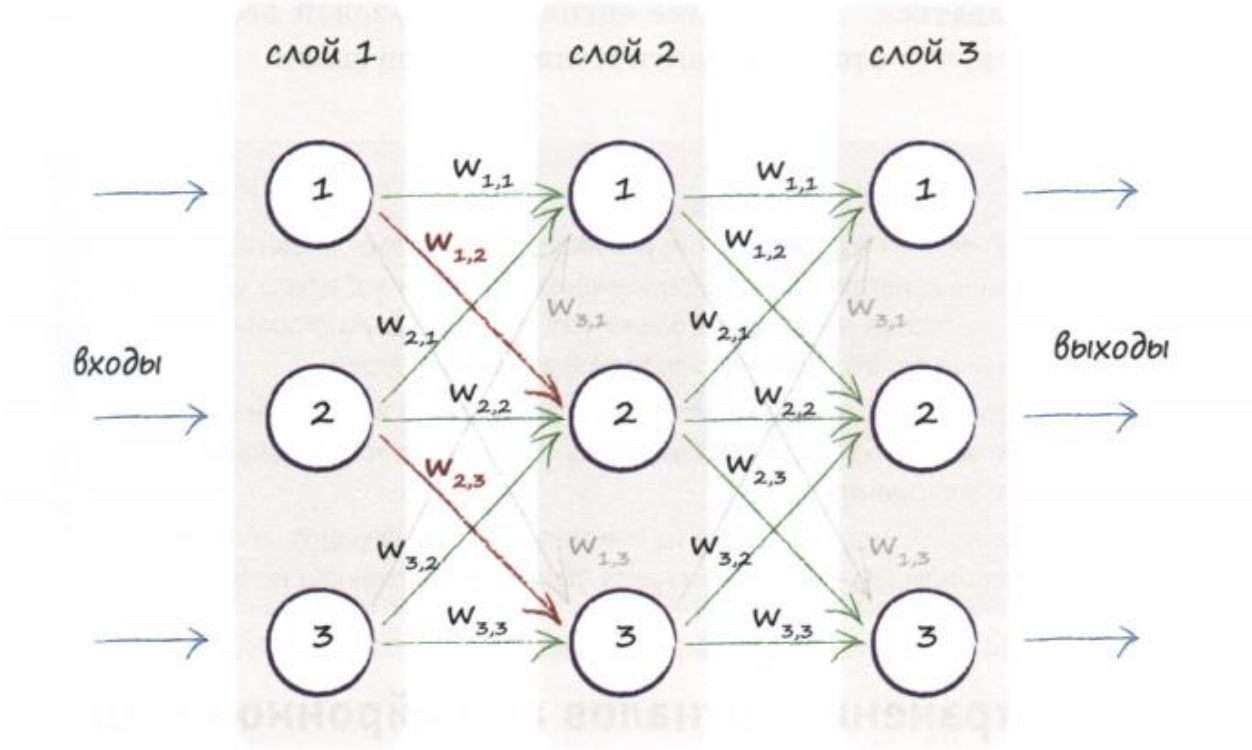

Рисунок 5 – Трехслойная нейронная сеть

В данной части будет рассказано про то, как сигнал проходит через слои нейронной сети, как формируется количество нейронов в каждом из слоев.

Перед тем, как на вход нейронной сети будут подаваться данные, формируются значения весовых коэффициентов случайным образом в диапазоне от -0.99 до 0.99. Каждый нейрон сети с рисунка 5 имеет по 3 синапса, каждый из которых соединяется с нейроном из следующего слоя. Весы для каждого слоя образуют матрицу собственную матрицу W, где w_{ij} – весовой коэффициент i-ого нейрона текущего слоя связанный с j-ым нейроном следующего слоя.

Вспомним, что каждый нейрон несет в себе функцию активации, истоки которой имеют корни биологического аналога нейрона. Биологические нейронные клетки получают сигнал и передают его дальше. Но не каждый входной сигнал возбуждает нейрон (преобразуется в выходной сигнал).

Сигнал должен иметь величину, превосходящую порог. И представить его можно в виде **ступенчатой функции.**

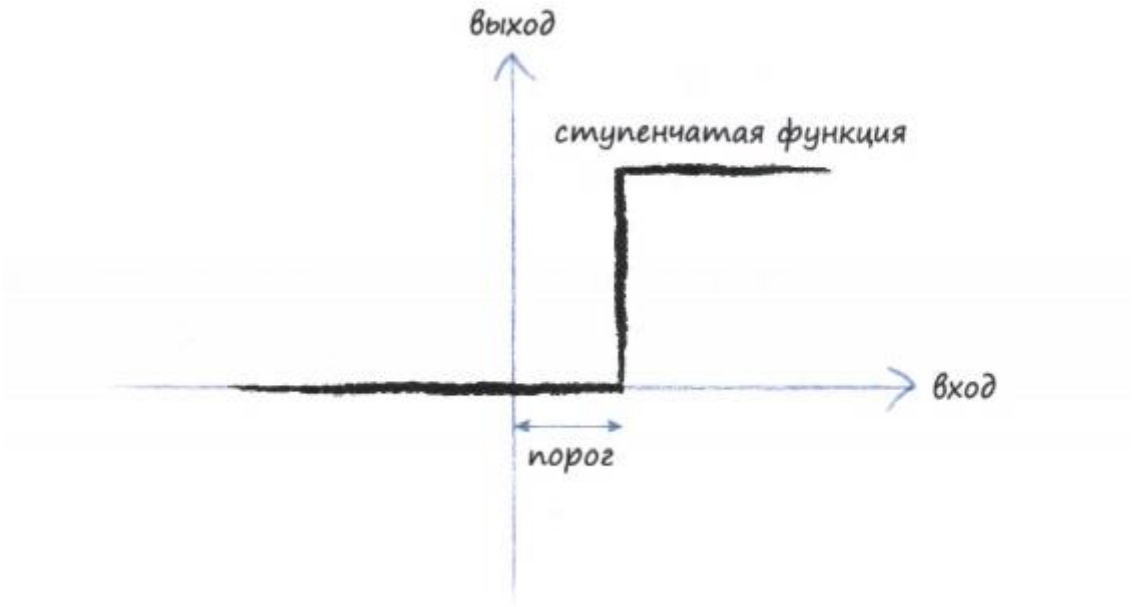

Рисунок 6 – Ступенчатая функция

Можно заметить, что слабые входные сигналы «не пройдут порог», т.е. не смогу преобразоваться в выходной сигнал. Поэтому ступенчатую функцию мы заменим на функцию **сигмоиды.**

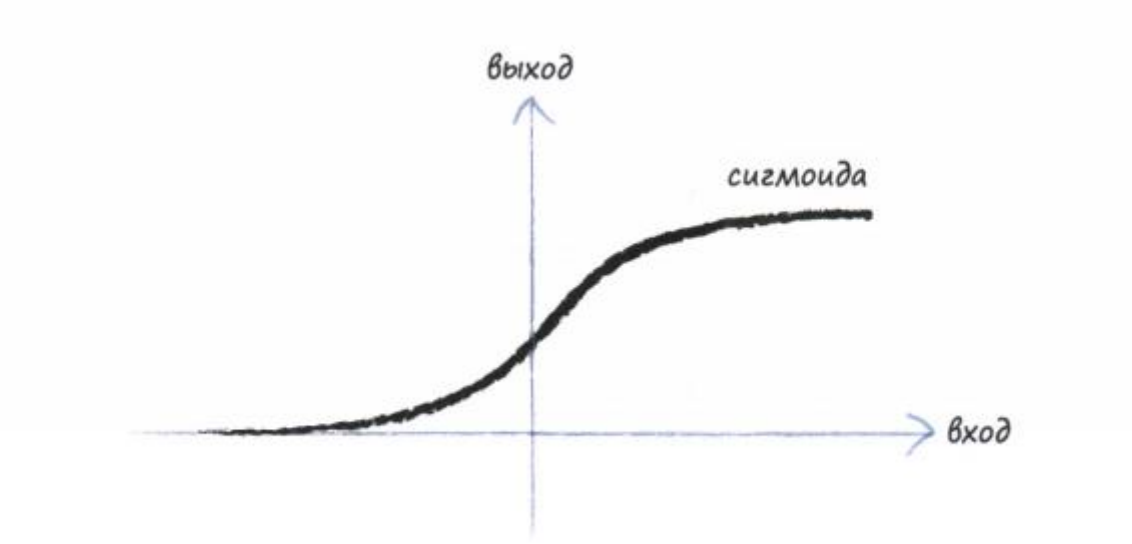

Рисунок 7 – Сигмоидная функция

Математически сигмоида представляется следующим образом:

$$y = \frac{1}{1+ e^{-x}}, \qquad (1)$$

где x-значение входного сигнала, а y-значение выходного сигнала.

Применяются и другие функции в качестве функции активации, но в своей нейронной сети я использовал только сигмоиду.

В итоге, мы имеем 2 матрицы весовых коэффициентов и знаем, что нейроны несут в себе функцию активации – сигмоиду. Стоить отметить, что нейроны входного слоя не имеют функции активации, это важно запомнить. Входной сигнал просто проходит через входные нейроны и по синапсу переходит на нейроны следующего слоя.

Теперь пошагово разберем, что же происходит при подаче входных данных (сигналов) нейронной сети.

Мы подаем на входной слой входной сигнал, который представлен в виде вектора размерностью n, где n – количество нейронов во входном слое. Важно отметить, что входные сигналы должны быть больше 0 и меньше 1. Как уже ранее говорилось, сигнал просто проходит через входной слой на следующий слой нейронов по синапсу. Каждый синапс имеет свой весовой коэффициент, значит сигнал проходя по синапсу должен быть умножен на вес синапса, по которому он проходит. Как видно на рисунке 5, во втором слое в каждый нейрон 2 слоя имеет синапс с нейроном 1 слоя, но в итоге в каждый нейрон 2 слоя должен прийти лишь один сигнал, это значит, что мы должны сложить сигналы, прошедшие по синапсу и тогда мы получим 1 сигнал.

Разберем более подробно на примере 1-го нейрона 2-го слоя. Мы имеем 3 сигнала I_1, I_2 и I_3, которые поступили на 3 нейрона входного слоя. Они проходят по синапсам на 1 нейрон 2-го слоя (т.е. должны быть умножены на вес синапса). Теперь мы имеем: $I_1 * w_{1,1}$, $I_2 * w_{2,1}$ и $I_3 * w_{3,1}$. И для получения одного сигнала мы должны сложить все 3 сигнала: $I_1 * w_{1,1} + I_2 * w_{2,1} + I_3 * w_{3,1}$. И так для всех 3 нейронов 2-го слоя:

$$I_1 * w_{1,1} + I_2 * w_{2,1} + I_3 * w_{3,1}$$
$$I_1 * w_{1,2} + I_2 * w_{2,2} + I_3 * w_{3,2}$$
$$I_1 * w_{1,3} + I_2 * w_{2,3} + I_3 * w_{3,3}$$

Если вспомнить, то похожие действия делаются при скалярном произведении матриц. А ранее я говорил, что мы имеем две матрицы W с весовыми коэффициентами и вектор сигналов. Таким образом, применяя скалярное произведение, мы получим на вход вектор новых сигналов:

$$I_2 = W_1 * I_1$$

Помним про то, что нейроны со 2 слоя и далее несут в себе функцию активации, то есть каждое значение вектора I_2 должно быть пропущено через функцию активации (сигмоиду) и имеем O_2.

Те же самые действия производятся и для 3 слоя нейронов –выходного.

$$I_3 = W_2 * O_2$$

На выходе нейронной сети мы получим O_3.

Это еще не все. В данной части был разобран лишь принцип того, как входной сигнал проходит через нейронную сеть на выход.

1.5 Обучение нейронной сети

Если нейронной сети просто подавать данные на вход, то мы ничего не получим, раз за разом мы будем получать какие-то непонятные случайные данные на выходе.

Обучение нейронной сети заключается в уравновешивании весовых коэффициентов синапсов. В проекте использовалась нейронная сеть прямого распространение, обучение происходило по принципу «обучение с учителем», когда на вход подаются данные и так же есть истинное значение, которое должно быть получено на выходе – целевое значение. Принцип обучения заключается в использовании **обратного распространения ошибки** и **градиентного спуска** для уравновешивания весов.

Разберем, как происходит процесс обратного распространения ошибки. Снова обратимся к рисунку 5, для абстрактного разбора. Вспомним, что как ранее мы считали, на выходе нейронной сети мы имеем вектор выходного сигнала O_3. При обучении с учителем, мы будем иметь вектор целевых значений T, имеющий такую же размерность, что и O_3. Ошибка на выходном слое: E_3= T - O_3. В данном случае, мы получаем значение ошибки, которая была получена в результате прохождения сигнала через синапсы 2 и 3 слоев (или скрытого и выходного). Чтобы получить ошибку на синапсах 1 и 2 слоев, необходимо транспонировать матрицу весов W_1 и умножить на значение ошибки, полученное ранее: $E_2= W_1^T * E_3$. Транспонирование необходимо для того, чтобы получить вклад каждого весового коэффициента, внесенного в полученную ошибку. В случае, если слоев больше 3, то мы продолжаем передавать ошибку назад по слоям по такому же принципу.

Пришло время разобрать саму сложную тему в обучении нейронных сетей – градиентный спуск. Именно благодаря ему мы можем изменять весовые коэффициенты. **Градиентный спуск** — метод нахождения локального минимум функции с помощью движения вдоль градиента. Идея заключается в том, что мы должны минимизировать величину ошибки посредством градиентного спуска, т.е. изменением весовых коэффициентов.

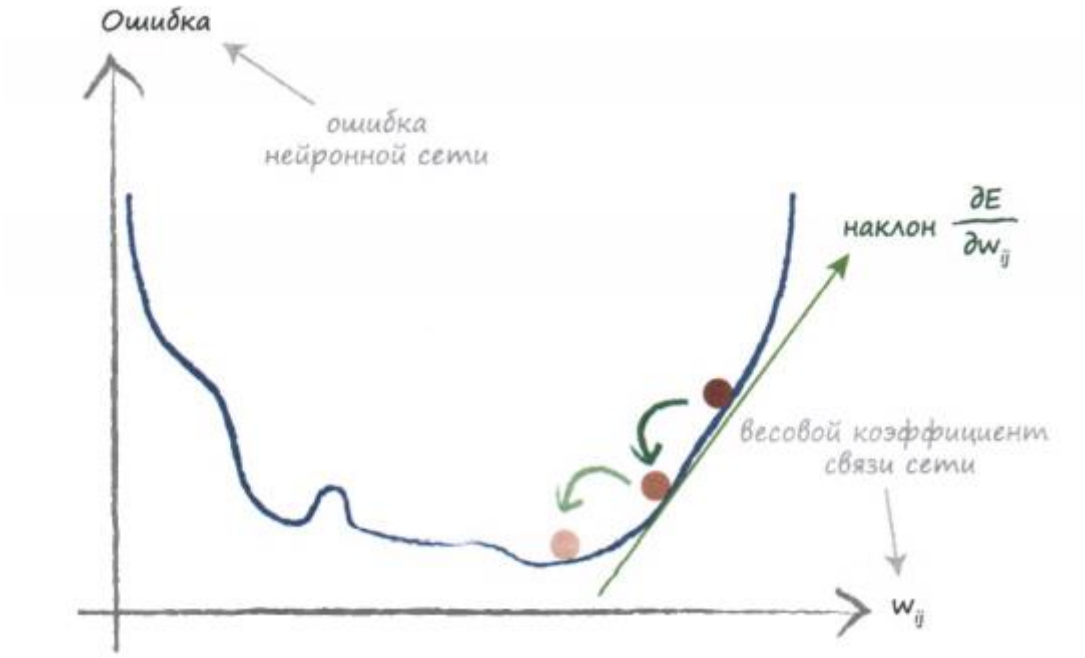

Рисунок 8 – Графическое отображение градиентного спуска

Перед тем как мы разберем то, как вычисляется $\frac{\partial E}{\partial w_{ij}}$, определим то, что функция ошибки имеет вид: (целевое значение – фактическое)^2.

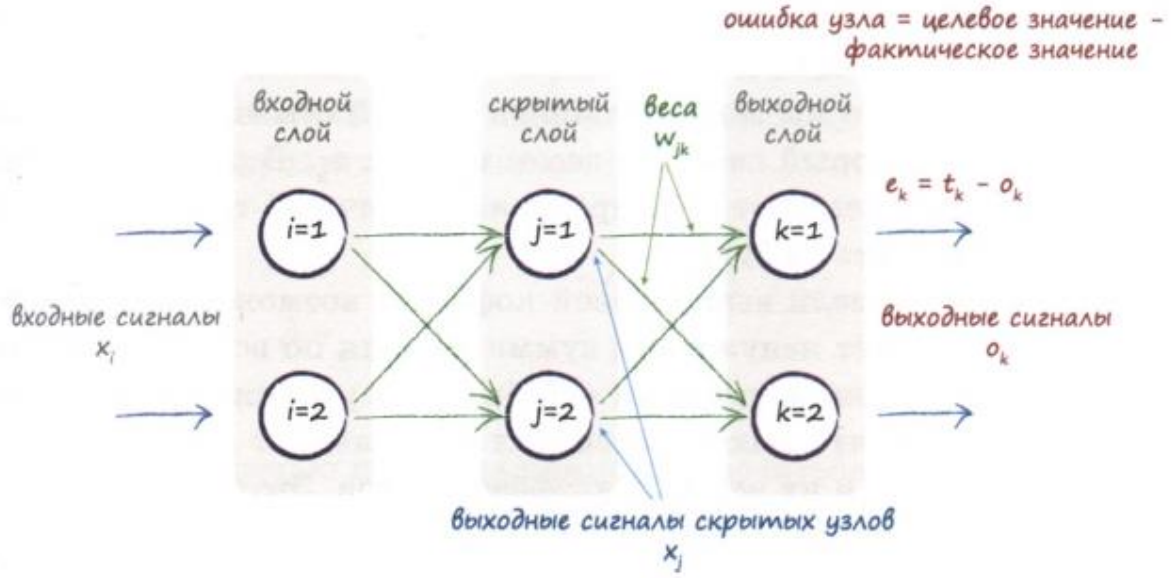

Рисунок 9 – Трехслойная нейронная сеть по 2 нейрона в каждом слое

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_n (t_n - o_n)^2$$

Суммирование происходит по всем n выходным узлам (нейронам). Сразу же мы можем упростить, т.к. видно, что выходной сигнал o_n зависит лишь от связей, которые с ним соединены.

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(t_k - o_k)^2$$

Умножим на $\frac{\partial o_k}{\partial o_k}$ и получим:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} * \frac{\partial o_k}{\partial w_{jk}}$$

После чего возьмем производную первого множителя (дифференцирование сложных функций) (t_k −константа):

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) * \frac{\partial o_k}{\partial w_{jk}}$$

o_k- это выходной сигнал на выходном слое, для каждого k-го нейрона (рисунок 9). Как мы помним, он у нас получается прохождением через функцию сигмоиды, в то время как перед попаданием в нейрон, у нас сигнал приходит со скрытого слоя, проходя по синапсам. $o_k = \text{сигмоида}(\sum_j w_{jk} * o_j)$, где o_j- выходной сигнал скрытого слоя. Теперь мы имеем:

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) * \frac{\partial}{\partial w_{jk}} \text{сигмоида}(\sum_j w_{jk} * o_j)$$

Нам необходимо продифференцировать функцию сигмоиды:

$$\frac{\partial}{\partial x} \text{сигмоида}(x) = \text{сигмоида}(x) * (1 - \text{сигмоида}(x))$$

Производная сигмоиды давно известна, и нет смысла разбирать то, как она получается.

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) * \text{сигмоида}\left(\sum_j w_{jk} * o_j\right) * \left(1 - \text{сигмоида}\left(\sum_j w_{jk} * o_j\right)\right) * \frac{\partial}{\partial w_{jk}}\left(\sum_j w_{jk} * o_j\right)$$

Не забываем про то, что нам надо продифференцировать функцию внутри сигмоиды:

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) * \text{сигмоида}\left(\sum_j w_{jk} * o_j\right) * \left(1 - \text{сигмоида}\left(\sum_j w_{jk} * o_j\right)\right) * o_j$$

Мы вправе избавить от множителя 2, т.к. нас интересует лишь направление функции градиента ошибки. И в итоге мы получаем функцию градиента ошибки выходного слоя:

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) * \text{сигмоида}\left(\sum_j w_{jk} * o_j\right) * \left(1 - \text{сигмоида}\left(\sum_j w_{jk} * o_j\right)\right) * o_j$$

Запишем выражение для вычисления градиента ошибки скрытого слоя:

$$\frac{\partial E}{\partial w_{ij}} = -e_j * \text{сигмоида}\left(\sum_i w_{ij} * o_i\right) * \left(1 - \text{сигмоида}\left(\sum_i w_{ij} * o_i\right)\right) * o_i$$

В мы заменили ошибку выходного слоя на ошибку скрытого слоя, которая рассчитывается посредством обратного распространения ошибки.

И так, мы получили две функции градиента ошибки, но стоит вспомнить, что изменение весовых коэффициентов обратно пропорционально направлению градиента, т.е. мы избавимся от знака минус. И добавим коэффициент обучения α, который определяет шаг градиентного спуска.

Запишем теперь все в матричном виде:

$$\Delta w_{ij} = \alpha * \mathrm{E}_j * O_j * (1 - O_j) * O_i^T, \qquad (2)$$

где Δw_{ij} −величина изменения весовых коэффициентов синпса, α −коэффициент обучения, E_j- вектор величины ошибки текущего слоя, O_j − вектор выходного сигнала текущего слоя, O_i − векторы выходного сигнала предыдущего слоя.

На каждой итерации обучения нейронной сети мы будем приближаться к наилучшему значению весовых коэффициентов.

Важно понимать, что для качественного обучения необходимо иметь сформированный дата сет большого объема.

1.6 Программная реализация нейронной сети на языке программирования Python

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой набор полезных функций.

Для реализации нейронной сети мы воспользуемся принципом «объектно-ориентированного программирования» (ООП) Питона. Он заключается в том, что мы можем создать класс, который имеет поля (характеристики класса) и методы (функции, которые может выполнять класс).

Класс нейронной сети лежит в файле *class_neural.py*

Листинг 1 – Инициализация класса нейронной сети

```
# определение класса нейронной сети
class neuralNetwork:
    # инициализирование нейронной сети
    def __init__(self, input_nodes: int, hidden_nodes: List[int],
output_nodes: int, learning_rate: float) -> NoReturn:
        # задаем количество узлов во входном, скрытом и выходном слоях
        self.inodes = input_nodes
        self.hnodes = hidden_nodes
        self.onodes = output_nodes
        # коэффициент обучения
        self.lr = learning_rate
        # список матриц весовых коэфициентов
        self.w = self.create_w()
        # использование сигмоиды в качестве функции активации
        self.activation_function = lambda x: sc.expit(x)

# создание списка матриц весовых коэфициентов
def create_w(self) -> List[Union[np.ndarray, float]]:
    list_w = []
    # если длина скрытых слов 0, то мы создаем одну матрицу весов с размерам
колличество выходных нейронов и входных
    if len(self.hnodes) == 0:
        # записываем в список весов сгенерированные случайным образом весовые
коэффициенты с применением
        # нормального распределения и сдвигом -0.5
        list_w.append(np.random.normal(0.0, pow(self.onodes, -0.5),
(self.onodes, self.inodes)))
    else:
        # иначе проходим по списку колличества нейронов скрытом слое и
генерируем весовые коэффициенты
        for i in range(len(self.hnodes)):
            if i == 0:
                list_w.append(np.random.normal(0.0, pow(self.hnodes[i], -
```

```
0.5), (self.hnodes[i], self.inodes)))
                if len(self.hnodes) > 1:
                    list_w.append(np.random.normal(0.0, pow(self.hnodes[i +
1], -0.5), (self.hnodes[i + 1],

self.hnodes[i])))
                else:
                    list_w.append(np.random.normal(0.0, pow(self.onodes, -
0.5), (self.onodes, self.hnodes[i])))
            elif i == len(self.hnodes) - 1:
                list_w.append(np.random.normal(0.0, pow(self.onodes, -0.5),
(self.onodes, self.hnodes[i])))
            else:
                list_w.append(np.random.normal(0.0, pow(self.hnodes[i + 1], -
0.5), (self.hnodes[i + 1],

self.hnodes[i])))
    return list_w
```

Листинг 2 – Метод опроса нейронной сети для пропускания входных данных через нейронную сеть и получения выходных

```
# опрос нейронной сети
def query(self, inputs_list: List[float]) -> List[float]:
    # преобразовать список входных значений в двухмерный массив
    inputs = np.array(inputs_list, ndmin=2).T
    # получение выходного сигнала
    for hn in self.w:
        inputs = self.activation_function(np.dot(hn, inputs))
    return inputs
```

Листинг 3 – Метод тренировки нейронной сети

```
# метод для тренировки нейронной сети
def train(self, inputs_list: List[float], targets_list: List[float]) ->
NoReturn:
    # преобразовать список входных значений в двухмерный массив
    inputs = np.array(inputs_list, ndmin=2).T
    targets = np.array(targets_list, ndmin=2).T
    outputs_list = []
    outputs = 0
    i = 0
    # получение выходных сигналов на каждом слое
    for hn in self.w:
        if i == 0:
            outputs = self.activation_function(np.dot(hn, inputs))
        else:
            outputs = self.activation_function(np.dot(hn, outputs))
        outputs_list.append(outputs)
        i += 1
    # получение ошибки в зависимости от целевого значения на выходе
    output_errors = targets - outputs
    # процесс градиентного спуска в зависимости от колличества скртых слоев
    if len(self.hnodes) == 0:
        self.w[0] += self.lr * np.dot((output_errors * outputs_list[0] * (1.0
- outputs_list[0])),
                                      np.transpose(inputs))
    else:
        i = 0
        outputs_list = outputs_list[::-1]
        for out in outputs_list:
            if i == 0:
                self.w[-(i + 1)] += self.lr * np.dot((output_errors * out *
```

```
(1.0 - out)),
                                                np.transpose(outputs_list[i +
1]))
            elif i == len(outputs_list) - 1:
                if len(self.w) > 2:
                    errors = np.dot(self.w[1].T, errors)
                    self.w[0] += self.lr * np.dot((errors * out * (1.0 -
out)), np.transpose(inputs))
                elif len(self.w) == 2:
                    errors = np.dot(self.w[-1].T, output_errors)
                    self.w[-(i + 1)] += self.lr * np.dot((errors * out * (1.0
- out)), np.transpose(inputs))
            elif i == 1:
                errors = np.dot(self.w[-1].T, output_errors)
                self.w[-(i + 1)] += self.lr * np.dot((errors * out * (1.0 -
out)),

np.transpose(outputs_list[i + 1]))
            else:
                errors = np.dot(self.w[-i].T, errors)
                self.w[-(i + 1)] += self.lr * np.dot((errors * out * (1.0 -
out)),

np.transpose(outputs_list[i + 1]))
            i += 1
```

2. Квантоворазмерная структура

2.1 Описание РТС

Квантоворазмерные структуры – это структуры, в которых свободные носители заряда локализованы в одном, двух или трех координатных направлениях в области с размерами порядка дебройлевской длины волны электрона.

Резонансно туннельная структура — это двойной потенциальный барьер с квантовой ямой. Толщины потенциальных барьеров и квантовой ямы таковы, что возможно эффективное туннелирование через каждый из барьеров, и движение электрона поперек ямы квантуется, чему соответствуют дискретные уровни энергии в яме. Эти явления наблюдаются в полупроводниках при толщинах барьеров и ямы порядка десятков и сотен ангстрем, т. е. сравнимых с длиной волны де Бройля.

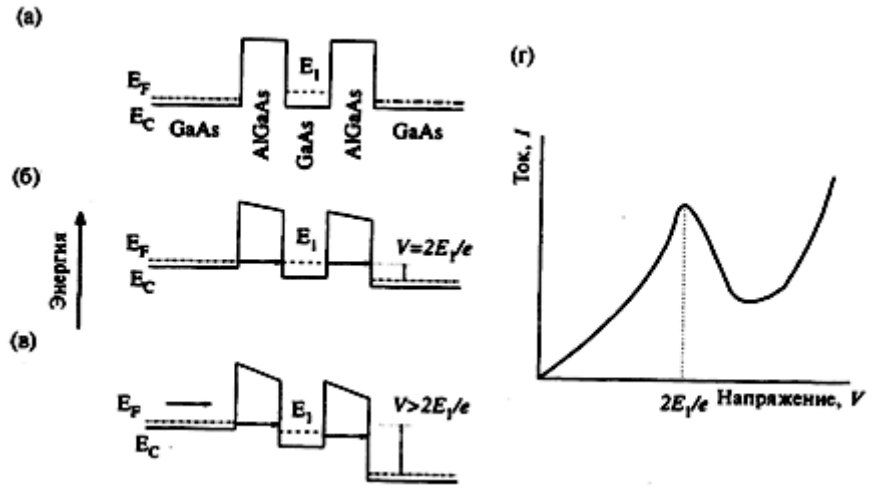

Рисунок 10 – Общий вид резонансно туннельной структуры и ВАХ

За протекание тока через резонансно-туннельную структуру отвечает последовательное туннелирование: при приложении потенциала к структуре электроны, энергия которых совпадает с разрешенным уровнем в квантовой яме, туннелируют на этот уровень сквозь левый барьер, а затем происходит еще одно туннелирование через правый барьер

2.2 Получение ВАХ РТС численным методом. Основное уравнения для получения значений тока.

Следующее уравнение является основным для получения значений тока, и будет фигурировать в задаче получения дата сета для обучения нейронной сети:

$$I = \frac{q}{h}\int_{-\infty}^{+\infty} dE * T(E)[f_1(E) - f_2(E)], \quad (3)$$

где q- заряд электрона, h – постоянная планка, $f_1(E)$ – функция Ферми для истока, $f_2(E)$ – функция Ферми для стока, $T(E)$ – функция (коэффициент) пропускания.

Функции Ферми имеют вид:

$$f_1(E) = \frac{1}{1+exp[(E-\mu_1)/k_BT]} = f_0(E-\mu_1), \quad (4)$$

$$f_1(E) = \frac{1}{1+exp[(E-\mu_2)/k_BT]} = f_0(E-\mu_2), \quad (5)$$

где k_B - постоянная Больцмана, E – энергия, μ_1- электрохимический потенциал истока, μ_2- электрохимический потенциал, T – температура.

Функция пропускания имеет вид:

$$T(E) = Tr[\Gamma_1 G \Gamma_2 G^+] = Tr[\Gamma_2 G \Gamma_1 G^+], \quad (7)$$

где G – функция Грина, G^+- эрмитово сопряженная функция Грина (транспонирование и сопряжение), Γ_1 и Γ_2 – уширение для истока и стока соответственно.

Функция Грина:

$$G = [EI - H - \Sigma_1 - \Sigma_2]^{-1}, \quad (8)$$

где E – это энергия, I – единичная матрица, H – матрица Гамильтониана, Σ_1и Σ_2 - матрицы собственно-энергетических функций для истока и стока соответсвенно.

Уширение:

$$\Gamma_1 = i[\Sigma_1 - \Sigma_1^+], \quad (9)$$

$$\Gamma_2 = i[\Sigma_2 - \Sigma_2^+]. \quad (10)$$

Матрицы собственно-энергетических функций:

$$\Sigma_1 = -t_0 exp(+ ika), \tag{11}$$

$$\Sigma_2 = -t_0 exp(+ ika), \tag{12}$$

где $t_0 = \frac{\hbar^2}{2ma^2}$, а - а – расстояние между точками (шаг).

Гамильтониан имеет вид трехдиагональной матрицы:

$$[\boldsymbol{H}] = \begin{bmatrix} \mathbf{2t_0+U_1} & \mathbf{-t_0} & \dots & \mathbf{0} \\ \mathbf{-t_0} & \mathbf{2t_0+U_2} & \dots & \mathbf{0} \\ \dots & \dots & & \dots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{2t_0+U_{N_p}} \end{bmatrix} \tag{13}$$

3. Получение ВАХ резонансно туннельной структуры при помощи искусственной нейронной сети

3.1 Создание дата сета

Одним из наиболее важных аспектов нашей задачи является создание хорошего и объемного дата сета. Главной задачей в создании дата сета было подобрать такие значения выходных данных (изменяемых параметров), чтобы при использовании уравнения 3, мы получали максимально характеристический график, как на рисунке 10г. Аналитическим путем такие граничные значения были подобраны. На вход нейронной сети планируется подавать 4 изменяющихся параметра: ширина потенциальной ямы, эффективная масса, энергия Ферми и потенциальная энергия. Предельные значения следующие: a=[2.90e-10… 3.05e-10] с шагом 0.01e-10, m=[0.250*9.1e-31…0.300*9.1e-31] с шагом 0.001*9.1e-3, Ef=[0.050…0.110] с шагом 0.001 и верхнее значение потенциальной энергии UB=[0.2…0.5] с шагом 0.1. В итоге, было сформировано 2 файла *input_200k.csv* и *output_200k.csv*, в каждом из которых приблизительно 200 тысяч тренировочных данных.

Отдельно необходимо отметить формат тренировочных данных. Как видим, у ширины потенциальной ямы, эффективной массы, а также у значений тока, довольно большой отрицательный порядок, что не есть хорошо, если такие малые числа подавать нейронной сети (сети трудно обрабатывать малые значения). Поэтому все значения, перед записью приводятся в нужный порядок. Ширина потенциальной ямы домножется на 1e9, эффективная масса - на 1e30, а ток - на 1e5.

Так же, не эффективным является подача 0 значения нейронной сети на вход, поэтому в массиве потенциальной энергии все значения равные 0, заменяются на значения близкие к нулю, а именно 0.01.

Программная часть по формированию дата сета реализована на языке программирования MATLAB. Основной причиной, по которой MATLAB был

выбран для создания дата сета является довольно быстрая работа с матричными данными. В файле *write_file.m* реализована часть по записи данных в файл формата csv. В файле *create_I.m* реализована функция, которая по входным данным и при помощи уравнения 3 формирует значения тока и отдает их на выход.

3.2 Разбор современных методов, применяемых в нейронных сетях

В 1 части была разобрана нейронная сеть, ее основной принцип работы, обучения и простейшая программная реализация.

Область машинного и глубокого (или глубинного) обучения не стоит на месте, нейронные сети развивают, улучшают, усложняют, чтобы они были более точными в своих результатах и применимы к широкому спектру задач, хотя и основной принц остается прежним.

Подобная программная реализация, как в листинге 1 уже устарела, вместо самостоятельного написания каждого слоя нейронной сети, градиентного спуска, используют различные фреймворки (PyTorch, TensorFlow, Keras и др.), в которых это все уже реализовано.

И так, разберем те нововведения, использованные в нейронной сети, которая использовалась в нашей задаче.

Вместо градиентного спуска теперь используется стохастический градиентный спуск (stochastic gradient descent (SGD)). Для данного метода формируется некий контейнер тренировочных данных выбранных из всего дата сета случайным образом размером n меньшим, чем размер всего дата сета – мини пакет (mini batch (дословный перевод)). После мы пропускаем пакет через нейросеть, то есть на выходе мы получаем n выходных данных и вычисляем ошибку (или функцию потерь (рассматривается дальше)) на всем пакете для нахождения градиента. Вот в чем основная суть стохастического градиентного спуска.

Основной проход сигнала через нейронную сеть такой же. Только теперь входные данные полученные на выходном слое после прохода по синапсу не проходят через функцию активацию, а просто отдаются на выход.

После получения выходных значений вместо формирования столбца ошибок на каждом выходном нейроне, мы формируем функцию потерь на данном шаге.

Функция потерь, применяемая для нашей задачи, называется «L1 loss» и имеет вид:

$$L = \frac{1}{n}\sum_s \sum_i |y_i - gt_i|, \quad (14)$$

где y_i – выходное значение на i-ом нейроне, gt_i – целевое значение, которое должно быть получено на i-ом нейроне, n – размер пакета. Как видно из уравнения, мы сначала суммируем по выходным нейронам, а после по s (это конкретный индекс данных из пакета).

После того, как мы вычислили значение функции потерь, вычисляем ее градиент, и при помощи обратного распространения ошибки вычисляем значение ошибки на всех слоях, после чего таким же образом умножаем коэффициент обучения на полученные градиенты и изменяем значения весов.

Одним из новых методов пакетная нормализация (batch-normalization) - метод, который позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей. Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию. Нормализация входного слоя нейронной сети обычно выполняется путем масштабирования данных, подаваемых в функции активации. Например, когда есть признаки со значениями от 0 до 1 и некоторые признаки со значениями от 1 до 1000, то их необходимо нормализовать, чтобы ускорить обучение. Нормализацию данных можно выполнить и в скрытых слоях нейронных сетей, что и делает метод пакетной нормализации.

Пусть на вход некоторому слою нейронной сети поступает вектор размерности d: x=($x^{(1)}$,…, $x^{(d)}$). Нормализуем данный вектор по каждой размерности k:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E(x^{(k)})}{\sqrt{D(x^{(k)})}}, \quad (15)$$

где математическое ожидание и дисперсия считаются по всей обучающей выборке (в данном случае по мини пакету). Такая нормализация входа слоя нейронной сети может изменить представление данных в слое. Чтобы избежать данной проблемы, вводятся два параметра сжатия и сдвига нормализованной величины для каждого $x^{(k)}$: $\gamma^{(k)}, \beta^{(k)}$— которые действуют следующим образом: $y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$. Данные параметры настраиваются в процессе обучения вместе с остальными параметрами модели.

3.3 Разбор структуры нейронной сети, использованной в проекте

Листинг 4 – Нейронная сеть, использованная в проекте (*neural_network.py)*

```
import torch
import torch.nn as nn


class NeuralNetwork(nn.Module):
    def __init__(self, input_nodes, hidden_nodes, output_nodes):
        super(NeuralNetwork, self).__init__()
        self.bn1 = nn.BatchNorm1d(input_nodes)
        self.fc1 = nn.Linear(input_nodes, hidden_nodes)
        self.bn2 = nn.BatchNorm1d(hidden_nodes)
        self.fc2 = nn.Linear(hidden_nodes, output_nodes)
        self.bn3 = nn.BatchNorm1d(output_nodes)

    def forward(self, x):
        x = self.bn1(x)
        x = self.fc1(x)
        x = self.bn2(x)
        x = torch.sigmoid(x)
         x = self.fc2(x)
        x = self.bn3(x)
        return x
```

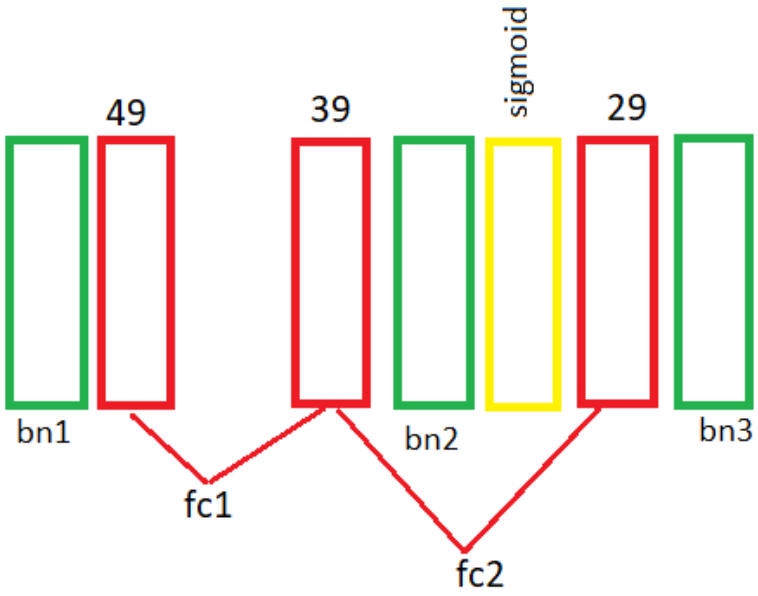

Рисунок 11 – Визуальное представление нейронной сети

В листинге 2 представлен класс, который содержит в себе слои нейронной сети и метод (функция) прохождения сигнала через сеть.

Разберем последовательно то, как сигнал проходит через данную сеть.

На вход поступает мини пакет размером 64x49. Входные данные проходят через пакетную нормализацию (*bn1*). После попадает на первый входной слой и по синапсу проходит на скрытый слой, синапс входной-скрытый обозначен, как *fc1*. Теперь мы имеем матрицу размером 64x39, которая проходит еще одну пакетную нормализацию (*bn2*). После нормализованные данные проходят функцию активации, представленную сигмоидой и по синапсу, проходят на выходной слой (*fc2*). Теперь мы имеем матрицу размером 64x29. Прежде чем отдать данные на выход, они проходят последнюю пакетную нормализацию (*bn3*) и сеть выдает значения.

Таким образом у нас происходит прохождение данных во время тренировки. При получении значений после тренировки, т.е. получении результатов на тестовых данных мы подаем данные по одному, а не пакетом, имеем матрицу 1x49, а принцип прохождения остается прежним.

3.4 Процесс тренировки нейронной сети в проекте

Листинг 5 – Тренировка нейронной сети (*train_net.py*)

```
# импортирование классов для использования функции оптимизации и функции
потерь
from torch import optim, nn
# импортирование конфигураций сети
from configurations import net_config
# импортирование тренировочного дата сета
from data_set import train_data
# импортирование класса для образования дата сета для тренировки
from torch.utils.data import DataLoader
# импортирование функции отрисовки графиков тренировки
from draw_learning_graphics import draw_learnig_graphics


def train_net(net: nn.Module) -> None:
    # создание объекта для оптимизации нейронной сети (используемый метод
стохастический градиентный
    # спуск) передаем параметры сети и коэффициент обучения
    optimizer = optim.SGD(net.parameters(), lr=net_config.learning_rate)
    # создаем объект для изменения коэффициента обучения
    sc = optim.lr_scheduler.CyclicLR(optimizer, base_lr=0.001, max_lr=0.9,
                                     step_size_up=5, mode="triangular2")
    # создаем объект функции потерь
    criterion = nn.L1Loss()
    # массив для хранения значений потерь
    loss_array = []
    # массив для хранения значений коэффициентов обучения
    lr_array = []
    # цикл тренировки в эпохах
    for epoh in range(net_config.epohs):
        # создание объекта тренировочного дата сета, в котором данные
объеденены в пакеты по 64 и
        # перемешаны
        train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
        # проход по мини пакетам
        for batch_idx, (data, target) in enumerate(train_loader):
            # сброс значения градиента
            optimizer.zero_grad()
            # прямой проход данных через сеть
            net_out = net(data)
            # вычисление значения функции потерь
            loss = criterion(net_out, target)
            # обратное распространение
            loss.backward()
            # шаг оптимизации сети
            optimizer.step()
        # добавление весового коэффициента
        lr_array.append(float(optimizer.param_groups[0]['lr']))
        # шаг по изменению весового коэффициента
        sc.step(epoh)
        # добавление значений функции потерь в массив
        loss_array.append(float(loss.item()))
        # вывод эпохи, коэффициента обучения и функции потерь в текущем
состоянии
        print(epoh, optimizer.param_groups[0]['lr'], loss.item())
    # отрисовка графиков обучения
    draw_learnig_graphics(lr_array, loss_array, net_config.epohs)
```

Как уже говорилось ранее, в настоящее время существует большое количество фреймворков для реализации нейронных сетей. В данном проекте используется PyTorch, разработанный компанией Facebook.

optimizer = optim.SGD(net.parameters(), lr=net_config.learning_rate) – для обучения нейронной сети используется метод стохастического градиентного спуска.

criterion = nn.L1Loss() – в качестве функции потерь выбрана L1 loss.

sc = optim.lr_scheduler.CyclicLR(optimizer, base_lr=0.001, max_lr=0.9, step_size_up=5, mode="triangular2") – так же для лучшего обучения коэффициент обучения изменяется динамически, для чего используется подобный класс.

Рисунок 12 – График изменения коэффициента обучения

Подобное изменение коэффициента обучения дало лучшие результаты на тестовых данных.

В каждом цикле эпох происходит прогон всего дата сета через нейронную сеть, на каждом цикле подачи мини пакета нейронной сети на вход вычисляется значение функции потерь, после чего исходя из потерь происходит уравновешивание весовых коэффициентов. Каждые 5 эпох происходит обновление коэффициента обучения. За 400 эпох нейронная сеть проходит обучение.

Многие конфигурации примененные в тренировке сети носят характер аналитического подбора. Количество эпох, размер мини пакета, в каких слоях использовать пакетную нормализацию, динамический коэффициент обучения – все эти параметры подбирались аналитическим путем.

3.5 Результат обучения нейронной сети

В качестве тестовых данных были выбраны следующие параметры структур: a=3e-10, Ef=0.1, m=0.25*9.1e-31 и 4 максимальных значения барьеров UB = 0.5, 0.4, 0.3, 0.2.

Рисунок 13 – ВАХ при барьере 0.2 эВ

Рисунок 14 – ВАХ при барьере 0.3 эВ

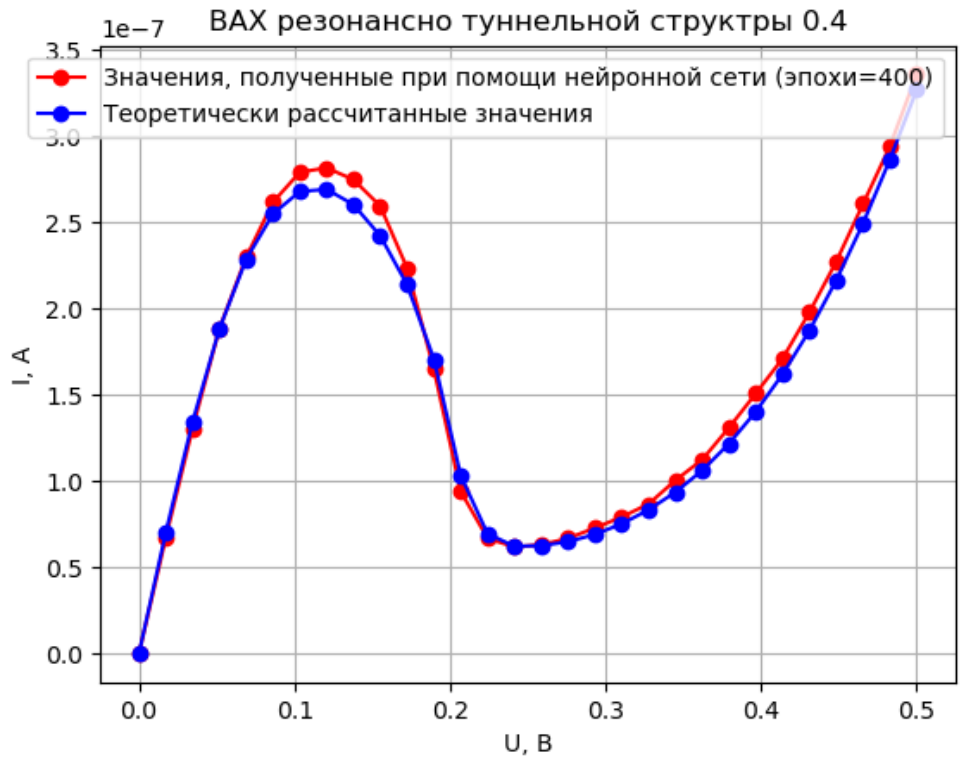

Рисунок 15 – ВАХ при барьере 0.4 эВ

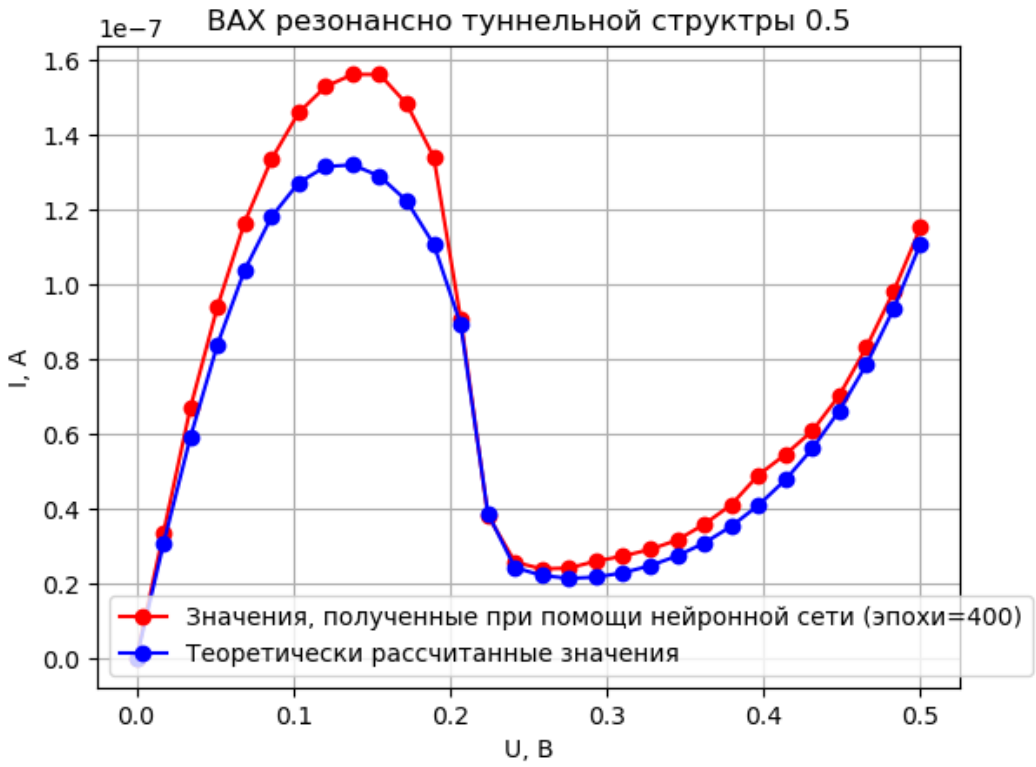

Рисунок 16 – ВАХ при барьере 0.5 эВ

Как видите, на большинстве тестовых данных, значения тока либо совпадают, либо очень близки к реальным. Больше всего выбивается последний график. Но в целом результаты, которые выдает нейронная сеть, говорят о ее хорошем обучении.

Литература

- Тарик Рашид Создаем Нейронную сеть: Пер. с англ. – СПб.: ООО «Альфа-книга», 2017г.
- Supriyo Datta Quantum Transport: Atom to Transistor – 2005г.
- Образцов Олег, Кузин Михаил Метод функций Грина для моделирования токопереноса в резонансно-туннельной структуре – 2020г.
- Козлов С. М. Deep Learning на пальцах [Электронный ресурс] // Видеокурс Deep Learning на пальцах 2019. URL: https://dlcourse.ai

Приложение А

Листинг 6 – Функция тестирования сети (*test_net.py*)

```
# импортирование библиотеки для работы с векторами
import numpy as np

# импортирование класса для образования дата сета для тестов
from torch.utils.data import DataLoader

# импортирование конфигураций сети
from configurations import net_config

# импортирование тестового дата сета
from data_set import test_data

#  импортирование функции отрисовки графиков ВАХ
from draw_graphic import draw_graphic


def test_net(net):
    test_loader = DataLoader(test_data)
    # перевод сети в режим проверки (тестирования)
    net.eval()
    # проход по дата сету
    for i, (data, target) in enumerate(test_loader):
        # получение выходных значений сети
        out = net(data) * 1e-5
        # создание массива со значениями ток и добавление 0 в начало
        current = np.array([0])
        # добавление значений полученных нейронной сетью
        current = np.append(current, out.detach().numpy())
        # создание массива с реальными значениями тока и добавление 0 в
начало
        target_current = np.array([0])
        # добавление реальных значений тока
        target_current = np.append(target_current, target.detach().numpy() *
1e-5)
        # отрисовка графиков ВАХ
        draw_graphic(current, target_current, net_config.epohs,
round(float(data[0][18]), 1))
```

Листинг 7 – Главная функция для старта программы (*main.py*)

```
from configurations import net_config  # импортирование конфигураций
нейронной сети
from neural_network import NeuralNetwork  # импортирование нейронной сети
from test_net import test_net  # импортирование функции тестирования
from train_net import train_net  # импортирование функции тренировки

# создание объекта нейронной сети
net = NeuralNetwork(net_config.input_nodes, net_config.hidden_nodes,
net_config.output_nodes)
# вызов функции тренировки сети
train_net(net)
# вызов функции тестирования сети
test_net(net)
```

Листинг 8 – Функция для построения графиков тренировки ИНС

(*draw_learning_graphics.py*)

```
import matplotlib.pyplot as plt  # импортирование библиотеки для построения
графиков
```

```
def draw_learnig_graphics(lr, loss_1, epohs):
    """Функция по отрисовке графиков связанных с обучением"""
    # создание массива эпох
    epoh = [i for i in range(epohs)]
    plt.title(f"Зависимость функции потерь от коэффициента обучения")
    # построение графика зависимости функции потерь от коэффициента обучения
    plt.plot(loss_1, lr, color='r')
    plt.xlabel('loss')  # подпись оси x
    plt.ylabel('lr')  # подпись оси y
    plt.grid()  # включаем сетку
    plt.show()  # вывод графика
    plt.title(f"Зависимость функции потерь от эпох")
    # построение графика зависимости функции потерь от эпох
    plt.plot(epoh, loss_1, color='r')
    plt.xlabel('epohs')  # подпись оси x
    plt.ylabel('loss')  # подпись оси y
    plt.grid()  # включаем сетку
    plt.show()  # вывод графика
    plt.title(f"Зависимость функции потерь от коэффициента обучения")
    # построение графика зависимости функции потерь от коэффициента обучения
    plt.plot(epoh, lr, color='r')
    plt.xlabel('epoh')  # подпись оси x
    plt.ylabel('lr')  # подпись оси y
    plt.grid()  # включаем сетку
    plt.show()  # вывод графика
```

Листинг 9 – Функция построения графика ВАХ (*draw_graphic.py*)

```
# импортирование библиотеки типизации данных
from typing import List, NoReturn

# импортирование библиотеки для построения графиков
import matplotlib.pyplot as plt

# импортирование библиотеки для работы с матрицами, векторами
import numpy as np

# создание значений напряжения
V = np.linspace(0, 0.5, 30)


def draw_graphic(
    current: List[np.array], target_current: List[np.array], epohs: int, u:
float
) -> NoReturn:
    """ "Функция для отрисовки 2-ух графиков: реального и того, что получаем
    с помощью нейронной сети"""
    plt.title(f"ВАХ резонансно туннельной структры {u}")
    # построение графика ВАХ с полученными значениями Тока из нейронной сети
    plt.plot(
        V,
        current,
        'r-o',
        color='r',
        label=f'Значения, полученные при помощи нейронной сети
(эпохи={epohs})',
    )
    # построение графика реальных значений ВАХ
    plt.plot(V, target_current, 'r-o', color='b', label='Теоретически
рассчитанные значения')
    plt.xlabel('U, В')  # подпись оси x
    plt.ylabel('I, А')  # подпись оси y
    plt.legend()  # легенда
```

```
    plt.grid()  # включаем сетку
    plt.show()  # вывод графика
```

Листинг 10 – Класс дата сета (*data_set.py*)

```
# импортирование библиотеки для считывания данных
import pandas as pd

# импортирование библиотеки для создания тензора
import torch

# импортирование класса дата сета
from torch.utils.data import Dataset


# Создание класса по созданию дата сета, необходим для дальнейшей работы с
ним и правильной
# обработки
class DataSet(Dataset):
    def __init__(self, input_file, output_file):
        # считывание данных их фалов
        input_file_data = pd.read_csv(input_file)
        output_file_data = pd.read_csv(output_file)
        # перезапись данных
        input_data = input_file_data.iloc[0:, 0:49].values
        output_data = output_file_data.iloc[0:, 1:30].values
        # создание переменных для дата сета
        self.x = torch.tensor(input_data, dtype=torch.float)
        self.y = torch.tensor(output_data, dtype=torch.float)

    # создание методов необходимых для работы других инструментов с классом
    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]


# создание объекта дата сета тренировочных данных
train_data = DataSet('../data_sets/input_200k.csv',
'../data_sets/output_200k.csv')
# создание объекта дата сета тестовых данных
test_data = DataSet('../data_sets/input_test.csv',
'../data_sets/output_test.csv')
```

Листинг 11 – Класс с конфигурациями нейронной сети (*configurations.py*)

```
from pydantic import BaseSettings


# Создание класса, который будет хранить в себе конфигурации сети
class Configurations(BaseSettings):

    input_nodes: int = 49
    hidden_nodes: int = 39
    output_nodes: int = 29
    learning_rate: float = 0.9
    epohs: int = 200


# Создание объекта
net_config = Configurations()
```