Механико-математический факультет
Кафедра вычислительной механики

Курсовая работа

Родионов Данила, 326 группа
Научный руководитель:
Николай Альбертович Зайцев

2022

Содержание

1 Введение **3**

2 Постановка задачи, необходимая часть формализации получения конечной функции. **4**
2.1 Постановка задачи. 4
2.2 Получение точных граничных условий 5
2.3 Аппроксимация ядра свертки 6

3 Использование аппарата нейронных сетей для реализации аппроксимации Паде полученной функции ядра свертки **7**
3.1 Вопрос существования аппроксимации 7
3.1.1 Теорема Цыбенко 7

4 Постановка задачи. Выбор архитектуры нейронной сети **9**
4.1 Метод scipy.interpolate.Pade 9
4.2 Инициализация нейронной сети 10

5 Реализация нейронной сети. **13**

6 Решение задачи переобучения **16**

1 Введение

Существует большое количество методов построения искусственных граничных условий на открытых границах. Среди них особое место занимают так называемые прозрачные граничные условия (ПГУ), основанные на преобразовании Лапласа по времени и аппроксимация ядра свертки полученного точного граничного условия суммой экспонент.

В настоящей работе предложена численная аппроксимация Паде изображения ядра свертки ПГУ как функции двойственной переменной s с помощью применения нейронной сети. Возможность реализации подобной конструкции обусловлена теоремой Цыбенко(или универсальной теоремой об аппроксимации). Эта теорема, доказанная Джорджем Цыбенко в 1989 году, подтверждает существование аппроксимации функции, заданной на некотором компактном в R^n множестве, которое может быть реализовано с помощью полносвязной нейронной сети прямого распространения с достаточно малым количеством скрытых слоёв.

В работе предложена постановка задачи, алгоритм вывода функции ядра свертки, который полностью описан в работе Н.А. Зайцева "Прозрачные граничные условия для волнового уравнения в канале кругового сечения"([1]), полученная в пространстве изображения путем применения преобразований Фурье и Лапласа. Описан алгоритм реализации аппроксимации полученной функции $F(s)$ с использованием библиотек Keras, Tensorflow и scipy для непосредственной аппроксимации и создания архитектуры нейронной сети прямого распространения с сигмоидальными функциями активации, описан подход к решению классической проблемы переобучения нейронных сетей для данной задачи с помощью L_2-регуляризации и частичного "отсева"узлов (Dropout).

Все выкладки, требующие пояснения к результату, сопровождаются иллюстрациями и примерами кода.

2 Постановка задачи, необходимая часть формализации получения конечной функции.

2.1 Постановка задачи.

Рассмотрим задачу о распространении волн в бесконечном канале кругового сечения радиуса a. Будем считать, что ось канала совпадает с осью z цилиндрической системы координат. Будем искать численное решение задачи в конечной подобласти Ω этого канала, которая ограничена искусственными границами - плоскостями $z = z_L$ и $z = z_R$. Положим, что во всей трубе выполнены следующие условия:

а) поведение неизвестной функции w как функции цилиндрических координат (r, φ, z) и времени t $(w = w(r, \varphi, z))$ описывается волновым уравнением :

$$\frac{1}{c^2}\frac{\partial^2 w}{\partial t^2} = \Delta w$$

где c - скорость звука среды, t - время, Δ - оператор Лапласа

б) при неположительных $t \leq 0$ доопределим w=0.

в) при $r = a$ выполнено граничное условие условие

$$\alpha\frac{\partial w}{\partial r} + \beta w = 0, |\alpha| + |\beta| \neq 0$$

Заметим, что при равенстве нулю одного из параметров, получаются однородные условия Неймана и Дирихле.

На границах $z = z_L$ и $z = z_R$ требуется поставить такие граничные условия, чтобы решение задачи в Ω совпадало с решением во всей неограниченной области.

2.2 Получение точных граничных условий

Рассмотрим разложение решения w по собственным функциям оператора Лапласа по собственным функциям оператора Лапласа в круге радиуса a:

$$w(r,\varphi,z,t) = \sum_{n=0}^{\infty}\sum_{k=1}^{\infty}\sum_{m=1}^{2}\psi_{k,n,m}(r,\varphi)w_{k,n,m}(z,t)$$

Для собственной функции $\psi_{k,n,m}$ собственным значением является

$$\lambda_{k,n,m} = (\frac{\mu_{k,n}}{a})^2$$

. При этом выполняется соотношение $\Delta\psi_{k,n,m} + \lambda_{k,n}\psi_{k,n,m} = 0$
Подставим разложение по собственным функциям Лапласа функции w в исходное волновое уравнение. Тогда получим следующее соотношение для коэффициентов Фурье:

$$\frac{1}{c^2}\frac{\partial^2 w_{k,m,n}}{\partial t^2} = \frac{\partial^2 w_{k,m,n}}{\partial z^2} - \lambda_{k,n}w_{k,m,n}(a)$$

Таким образом получение ПГУ для каждой гармоники автоматически даст ПГУ для исходного волнового уравнения. Введем замену координат:

$$\bar{t} = c\lambda_{k,n}t$$
$$\bar{z} = \lambda_{k,n}z$$

Тогда уравнение (а) перейдет в уравнение вида:

$$\ddot{w_{k,m,n}} = w_{k,m,n}'' - w_{k,m,n}$$

Исходя из этого будем строить ПГУ для уравнения: $\ddot{u} = u'' - u$
Буквой u здесь обозначена амплитуда гармоники $\psi_{k,n,m}$
После применения преобразования Лапласа это уравнение перейдет в уравнение:

$$U'' = (s^2 + 1)U$$

В силу ограниченности решения это уравнение имеет в качестве последнего функцию $U = const * e^{d\bar{z}}$ и для функции U будет выполнено соотношение:

$$\frac{U'}{U} = d$$

Его можно переписать в виде $U' = PU$, так как это и есть искомое граничное условие. Теперь необходимо определить P.
Подставим полученный вид функции U в уравнение второго порядка, полученное после преобразования Лапласа.

Для ограниченности решения на правой и левой границах получим
$$P = \pm\sqrt{s^2+1}.$$

Выделим наибольшие степени по переменной s функции P перед тем как перейти к оригиналу:

$$P(s) = p_1 s + p_0 + F(s)$$

При этом $p_1 p_0$ - константы, а функция $F(s) \to 0$ при $s \to 0$.
Далее необходимо определить константы $p_1 p_0$. Подставим полученное разложение в уравнение $U'' = (s^2+1)U$ и, приравняв коэффициенты при одинаковых степенях s, получим соотношения:

$$p_1^2 = 1$$
$$2\mathrm{p}_1 p_0 = 0$$

откуда $p_1 = 1, p_0 = 0$.
Возвращаясь к оригиналам изображений, получим граничное условие

$$u' = p_1 \dot{u} + f(\bar{t}) * u,$$

$f(\bar{t})$ - оригинал ядра свертки. Для задачи с постоянными коэффициентами получим: $F(s) = \sqrt{s^2+1} - s$.

2.3 Аппроксимация ядра свертки

Заметим, что полученное граничное условие является точным. Однако при этом оно является малопригодным для численных расчетов, так как объем операция свертки является вычислительно затратной.
Для решения этой проблемы воспользуемся аппроксимацией ядра свертки для оператора ПГУ суммой экспонент:

$$\tilde{f}(t) = \sum_{p=1}^{M} a_p e^{b_p t}$$

Рассмотрим сумму экспонент в пространстве изображений:

$$\tilde{F}(s) = \sum_{p=1}^{M} \frac{a_p}{s - b_p}$$

Как видно, сумме экспонент соответствует сумма полюсов(первого порядка). Приведем сумму дробей к рациональной функции. После приведения дробей к общему знаменателю получим дробь:

$$\tilde{F}(s) = \frac{Q_{M-1}(s)}{P_M(s)}$$

где $Q_{M-1}(s), P_M(s)$ полиномы степеней соответствующих индексов. Поэтому для аппроксимации ядра свертки $F(s)$ вполне закономерно использовать аппроксимацию Паде.

3 Использование аппарата нейронных сетей для реализации аппроксимации Паде полученной функции ядра свертки

3.1 Вопрос существования аппроксимации

Использование аппарата нейронных сетей для аппроксимирования той или иной вещественнозначной функции зародилось еще в середине 1950-х годов. Первоначальной элементарной моделью являлся однослойный перцептрон. С помощью скалярного произведения вектора весов скрытого слоя и вектора входных данных на выходе получалось некоторое значение, которое сравнивалось с некоторым порогом. Задача обучения такой сети состояла в подборе значений вектора весов. На данный момент несмотря на значительный прогресс в архитектурах нейронных сетей общая идея обучения сетей осталась прежней.

3.1.1 Теорема Цыбенко

Пусть $\psi(x)$- некоторая вещественнозначная функция. Предположим, существует некоторая архитектура полносвязной сети прямого распространения N с заданными гиперпараметрами(количество скрытых слоев, семейство функций активации, стратегия обучения).
Тогда вполне закономерен следующий вопрос: можно ли с некоторой заданной точностью аппроксимировать данную функцию ψ с помощью этой архитектуры N.
Исчерпывающий формальный ответ на этот вопрос даёт теорема, сформулированная и доказанная Джоржем Цыбенко в 1989 году. Для начала

введем

Определение

Пусть $\sigma(x)$ - некоторая функция вещественной переменной x, такая, что:

$$\sigma(x) \to 1, x \to +\infty$$
$$\sigma(x) \to 0, x \to -\infty$$

Такую функцию функцию будем называть *сигмоидой*

Пример

В качестве функции активации в определенных архитектурах нейронных сетей используется функция

$$sigmoid = \frac{1}{1+e^{-x}}$$

Так, например, в алгоритме логистической регрессии эта функция активации интерпретирует вероятность того, что данное событие относится к тому или иному классу.

Итак, теперь мы можем сформулировать следующую

Теорема Цыбенко(универсальная теорема об аппроксимации)

Пусть φ - некоторая непрерывная сигмоидальная функция, $f(x)$ - произвольная непрерывная функция, заданная на некотором компактном в R^N множестве K. Тогда $\forall \epsilon > 0$ $\exists$ набор векторов $w_1, ..., w_N, \theta, \alpha$ и функция $G = \sum_{i=1}^{N} \alpha_i \varphi(w_i x + \theta_i)$, такая, что $\forall x \in R^N, |G - f(x)| < \epsilon$.

С доказательством этой теоремы можно ознакомиться в []. Заметим лишь, что оно опирается на тот факт, что множество сигмоидальных функций всюду плотно в $C(X)$

Таким образом, эта теорема гарантирует нам существование решения задачи аппроксимации с помощью нейронной сети с достаточным количеством гиперпараметров.

4 Постановка задачи. Выбор архитектуры нейронной сети

Задача формулируется следующим образом: *для полученной функции $F(s)$ ядра свертки решения волнового уравнения в канале сечения радиуса a необходимо получить аппроксимацию Паде с помощью нейронной сети.*
Для решения этой задачи воспользуемся API scipy и Tensorflow, для реализации последней будем использовать библиотеку Keras.

4.1 Метод scipy.interpolate.Pade

Первоначально обратимся к библиотеке scipy. Она имеет в своем арсенале метод scipy.interpolate.Pade, который представляет разложение Паде для заданной функции. Отметим при этом, что точность разложения, вообще говоря, имеет значительный спектр вариации, так как на вход метод получает коэффициенты ряда Тейлора разложения исходной функции. Поэтому очевидно, что если использовать, например, разложение в ряд Тейлора с остаточным членом в форме Пеано с точностью до $\bar{o}(x^4)$ и $\bar{o}(x^5)$, то точность полученного разложения будет соответственно ниже и выше.

Эту проблему можно решить эвристически, то есть подобрать достаточный порядок непосредственно, так как точного формального обоснования выбора последнего пока что не получено. Это первое препятствие, которое возникает при реализации данного метода.

Пример реализации метода

Рассмотрим функцию $f(x) = e^x$. Реализуем для неё рассмотренный метод:
» from scipy.interpolate import pade
e-exp = *[1.0, 1.0, 1.0/2.0, 1.0/6.0, 1.0/24.0, 1.0/120.0]*
p, q = pade(e-exp, 2)

На выходе метод выдает коэффициенты многочленов p, q,
$deg(p) = 1, deg(q) = 2$.
Заметим, что в данном случае порядок равен 6.

4.2 Инициализация нейронной сети

Библиотека Tensorflow позволяет непосредственно реализовывать архитектуры нейронных сетей с возможностью их дальнейшей компиляции и обучения.

Выбор архитектуры и гиперпараметров

Так как реализация нейронной сети опирается на формализм теоремы Цыбенко, вполне обоснованно использовать полносвязную нейронную сеть прямого распространения с сигоидальными функциями активации. Вопрос заключается в том, как подобрать количество скрытых слоёв. На данный момент полного обоснования для точного значения количества скрытых слоев нет, поэтому оно выбирается эвристически и обосновывается спецификой задачи. Однако так как задача аппроксимации функций с помощью нейронных сетей изучена в достаточной мере, на практике достаточно двух-трех слоёв.

Теперь необходимо выбрать метрику оценки качества нашей модели, то есть функцию потерь или лосс-функцию. Здесь необходимо заметить, что выбор метрики качества модели также зависит от решаемой задачи. Так, например, для задач классификации используются метрики точности (accuracy, precision, roc-auc-score), в то время как для задач восстановления регрессии используются метрики, основанные на методе наименьших квадратов(r2-отклонение, среднеквадратичная ошибка). Очевидно, что задача восстановления нелинейной регрессии по существу совпадает с задачей аппроксимации функции, поэтому реализованной модели в качестве функции потерь используется функция среднеквадратичной ошибки:

$$L(y, \bar{y}) = \frac{\sqrt{\sum_{i=1}^{m}(y_i - \bar{y_i})^2}}{m}$$

, где m - количество значений выборки.

Выбор метода обучения

Как было сказано ранее, задача обучения нейронной сети сводится к оптимизационной задаче, а именно - задаче минимизации некоторой функции(выбранной функции потерь). Классическим методом является *метод градиетного спуска.*

Он представляет собой метод поиска локального минимума функции путем сдвига на антиградиент функции потерь L (градиент, вычисляемый по коэффициентам весов нейронной сети):

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

Эта стратегия хороша тем, что в качестве параметра выступает лишь скорость обучения α, что должно облегчать реализацию модели.
Однако вместе с этим возникает и ряд существенных проблем, таких как:
1) как выбрать само значение для скорости обучения α?
2) при слишком больших α алгоритм может проскочить искомый минимум
3) при слишком малых α значительно увеличивается время работы алгоритма, при этом вычислительная нагрузка увеличивается
Для решения этих проблем стандартный алгоритм градиентного спуска был многократно модифицирован до следующих алгоритмов:
Стохастический градиентный спуск.
В данном случае градиент вычисляется не последовательно в точках, а в случайных подвыборках заданных данных, что позволяет усреднить общую ошибку обучения. Заметим, что при этом качество обучения на тренировочных данных ниже, чем при стандартном градиентном спуске, однако при конечной валидации качество выше. Пусть P_j - некоторое разбиение множества полученных тренировочных данных X

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \sum_{x_i \in P_j} \frac{\partial L}{\partial \mathbf{W}}|_{x=x_i}$$

Импульсный градиентный спуск(GD with momentum)
Как было отмечено ранее, при стандартном градиентном спуске есть риск "проскочить"локальный минимум или же выбрать неправильное "глобальное"направление спуска. Эту проблему успешно решает импульсный градиентный спуск, который опускает минимумы м*е*ньшего порядка и не позволяет миновать глобальные минимумы функции потерь. Для в обновление градиента добавляется параметр затухания β, то есть обновление градиента имеет вид:

$$\mathbf{w}_{i+1} = \beta \frac{\partial L}{\partial \mathbf{W}}|_{x=x_{i-1}} - \alpha \frac{\partial L}{\partial \mathbf{W}}|_{x=x_i} + \mathbf{w}_i$$

Импульсный градиентный спуск Нестерова
Этот метод отличается от предыдущего тем, что градиент, вычисленный на i-ом шаге в качестве аргумента имеет "сглаженный"вектор весов, т.е. $L = L(\mathbf{W} + \beta\mathbf{V})$. Заметим однако, что этот метод хорош только для небольших множеств, т.к. при значительном увеличении выборки он снижает ошибку до $O(\frac{1}{t^2})$, что сравнимо с обыкновенным методом импульсов.

В качестве дальнейших модификаций можно рассматривать алгоритмы AdaGrad(нормированное обновление агрегации), RMSProp(выпуклое обновление весов для избегания неопределенности при малой агрегации(значения i-ой компоненты градиента)) и Adam, который представляет собой выпуклое обновление значений градиента и весов.

Все эти алгоритмы нужны в тех случаях, когда эффективность стандартного градиентного спуска очевидно слишком мала(это определяется спецификой задачи и входными данными). Однако в случае, когда задача при предельной линеаризации представляет собой линейную модель(линейная классификация, линейная регрессия), эффективность каждого из этих алгоритмов достаточно близка к стратегии обыкновенного градиентного спуска. Поэтому стратегией минимизации лосс-функции в данной задаче будет являться обыкновенный градиентный спуск.
С учетом теоремы Цыбенко в качестве функции активации будем использовать функцию, указанную в качестве примера несколько ранее:

$$sigmoid = \frac{1}{1+e^{-x}}$$

Тренировочные и тестовые данные
Функция $F(s)$ определена, вообще говоря, в $\mathbf{C}$, эффективно вычислить ее можно на луче $s \geq -1$, поэтому логично рассмотреть некоторое компактное в $\mathbf{R}^1$ множество (отрезок) $\mathbf{I} = [-1+\delta; a+\delta]$, где число a возьмем не очень большим(чуть позже продемонстрируем некоторое ограничение в произволе выбора этого значения). Далее рассмотрим разбиение этого множества на два(тренировочные и тестовые данные). Далее рассмотрим образ $M = Im(F)|_I = [F(s)|s \in I]$ и рассмотрим его разбиение в соответствии с разбиением множества I. Таким образом получим пары $(I_1, M_1), (I_2, M_2)$ - тренировочные и валидационные данные соответственно, т.е. наша модель задана на сетке I_1xM_1, а проверяется на сетке I_2xM_2.

5 Реализация нейронной сети.

Первым шагом подключим необходимые модули:
import math
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow.compat.v1 as tf
import scipy
from scipy.interpolate import pade
tf.disable-v2-behavior()

Далее зададим архитектуру сети и метод аппроксимации:
x0, x1 = 0, 8 диапазон аргумента функции
test-data-size = 2000 количество данных для итерации обучения
iterations = 20000 количество итераций обучения
learn-rate = 0.01 скорость спуска
hiddenSize = 10 размер скрытого слоя
args = [1.0, -1.0/2.0, -1.0/8.0, 1.0/16.0, -5.0/128.0] коэффициенты ряда Тейлора
p, q = pade(args, 2)

Создание данных для обучения:

```
# функция генерации тестовых величин
def generate_test_values():
    train_x = []
    train_y = []

    for _ in range(test_data_size):
        x = x0+(x1-x0)*np.random.rand()
        y = math.sqrt(x+1)-x # исследуемая функция
        train_x.append([x])
        train_y.append([y])

    return np.array(train_x), np.array(train_y)
```

Инициализация архитектуры сети:

```
# узел на который будем подавать аргументы функции
x = tf.placeholder(tf.float32, [None, 1], name="x")

# узел на который будем подавать значения функции
y = tf.placeholder(tf.float32, [None, 1], name="y")

# скрытый слой
nn = tf.layers.dense(x, hiddenSize,
                     activation=tf.nn.sigmoid,
                     kernel_initializer=tf.initializers.ones(),
                     bias_initializer=tf.initializers.random_uniform(minval=-x1, maxval=-x0),
                     name="hidden")

# выходной слой
model = tf.layers.dense(nn, 1,
                        activation=None,
                        name="output")

# функция подсчёта ошибки
cost = tf.losses.mean_squared_error(y, model)

train = tf.train.GradientDescentOptimizer(learn_rate).minimize(cost)

init = tf.initializers.global_variables()
```

Компиляция и обучение:

```
with tf.Session() as session:
    session.run(init)

    for _ in range(iterations):

        train_dataset, train_values = generate_test_values()

        session.run(train, feed_dict={
            x: train_dataset,
            y: train_values
        })

        if(_ % 1000 == 999):
            print("cost = {}".format(session.run(cost, feed_dict={
                x: train_dataset,
                y: train_values
            })))

    train_dataset, train_values = generate_test_values()

    train_values1 = session.run(model, feed_dict={
        x: train_dataset,
    })
```

Отчетность о результате работы нейронной сети: график функции и её аппроксимации, значения весов и порогов скрытого и выходного слоёв, значения функции потерь в зависимости от эпохи обучения:

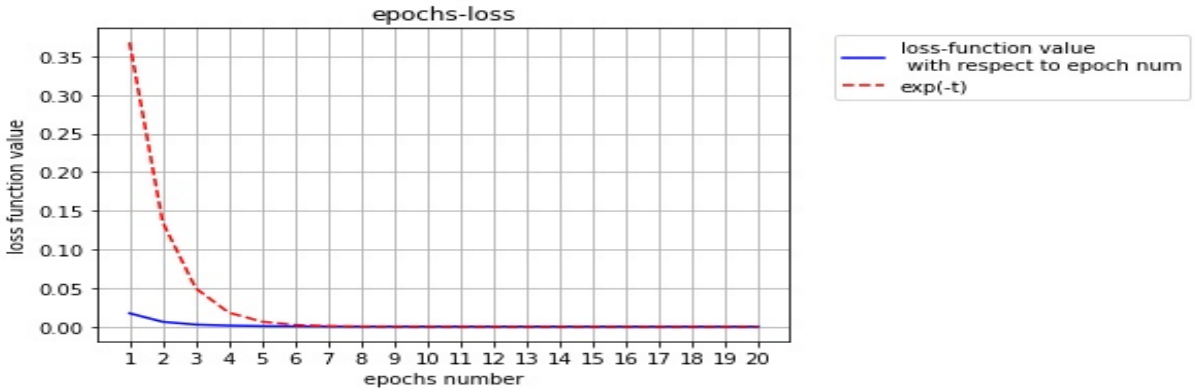
```
    plt.plot(train_dataset, train_values, "bo",
             train_dataset, p(train_dataset)/q(train_dataset), "ro")
    plt.show()

    with tf.variable_scope("hidden", reuse=True):
        w = tf.get_variable("kernel")
        b = tf.get_variable("bias")
        print("hidden:")
        print("kernel=", w.eval())
        print("bias = ", b.eval())

    with tf.variable_scope("output", reuse=True):
        w = tf.get_variable("kernel")
        b = tf.get_variable("bias")
        print("output:")
        print("kernel=", w.eval())
        print("bias = ", b.eval())
```

Асимптотика функции потерь

Построим график зависимости значения лосс-функции от номера эпохи обучения. Нетрудно заметить, что она имеет асимптотику e^{-t}, при $t \to +\infty$.



Значения весов и порогов скрытого и выходного слоя соответственно:

```
hidden:
kernel= [[0.8015867  0.9133082  0.98849505 0.6308246  1.0914394  0.6193903
  1.0853077  0.67437196 1.0363431  1.0754035 ]]
bias =  [-1.8791709  -0.80699486 -7.3330746  -3.880146   -0.9659601  -2.6865442
 -1.4706309  -2.9524467  -1.0820633  -7.8953533 ]
output:
kernel= [[-1.1665546 ]
 [-0.60778594]
 [-0.93441963]
 [-1.8553236 ]
 [ 0.16977698]
 [-1.0394013 ]
 [-0.5307021 ]
 [-0.90485567]
 [-0.20355643]
 [-0.5797366 ]]
bias =  [1.5447233]
```

6 Решение задачи переобучения

Как можно заметить из графика зависимости лосс-функции от количества эпох, при достаточно больших n асимптотика функции e^{-t}, то есть $\exists N$, такое, что $\forall m > N$ нейронная сеть с количеством эпох обучения $= m$ значения функции потерь будут очень малы.
В случае поставленной задачи необходимо найти аппроксимацию на некотором отрезке I. Что произойдет, если уже обученной сети подать на вход новый отрезок I'? Разумно предположить, что если $I \subset I'$, то $F|_I$, где F-отображение, которое задает нейронная сеть, должно совпадать с полученным ранее результатом, то есть модель работает корректно. Однако какой результат будет получен при вычислении $F|_{I'/I}$? Если выходные данные будут соответствовать действительности с точностью до малых порядка ϵ, то задача будет полностью решена : получена аппроксимация на всем луче.
Тем не менее, классической проблемой задач машинного обучения является проблема, вызванная неверным результатом валидации на новых данных. В нашем случае это означает следующее: точность аппроксимации на новом отрезке будет ниже, чем на отрезке, подданым для обуче-

ния нейронной сети. Существует несколько методов решения проблемы переобучения:

Регуляризация

В терминах параметров нейронной сети озвученная выше проблема означает, что часть весов при изменении данных обновляется некорректно, то есть часть весов просто "заучивается". Для решения этой проблемы применим следующую технику: пусть **w** - вектор весов нейронной сети. Далее будем рассматривать этот вектор как финитный элемент из пространств ℓ_1 или ℓ_2 с наследуемыми нормами:

$$|| \circ ||_{\ell_1} = \sum_{i=1}^{\infty} |w_i|,$$

$$|| \circ ||_{\ell_2} = \sum_{i=1}^{\infty} w_i^2$$

Как было замечено ранее, **w** финитный, следовательно вместо рядов будем иметь конечные суммы.
Теперь введем функцию потерь(рассмотрим случай ℓ_2):

$$\tilde{L} = L + \lambda ||\mathbf{w}||$$

Тогда если **w**= $(w^1, ..., w^n)$, то формула обновления весов будет иметь следующий вид:

$$w_{i+1}^j = w_i^j - \alpha \frac{\partial \tilde{L}}{\partial w^j}$$

$$w_{i+1}^j = w_i^j - \alpha(\frac{\partial L}{\partial w^j} + 2\lambda w_i^j)$$

$$w_{i+1}^j = w_i^j(1 - 2\lambda\alpha) - \alpha \frac{\partial L}{\partial w^j}$$

Как видно из конечного выражения, при компоненте w_i^j возник множитель, который "сглаживает"этот параметр, предотвращая тем самым неограниченный рост и слишком малые изменения. Таким образом, с помощью этого множителя предотвращается "запоминание"этого веса. Параметр λ носит название *параметр регуляризации*
Выше мы рассмотрели только случай нормы пространства ℓ_2, однако для пространства ℓ_1 выкладки аналогичны.
На практике преобладает применение метода L_2-регуляризации, так как при использовании метода L_1 возникает $sgn(w_i^j)$ при дифференцировании модуля, в связи с чем в конечном итоге определенная часть весов

при входном слое обращается в нуль, что способствует избавлению от лишних входных узлов. Этот метод также носит название *лассо* и применяется преимущественно для предобработки данных.
Заметим, что метод регуляризации, безусловно, является далеко не единственным способом решения проблемы переобучения. На практике часто он комбинируется с так называемыми ансамблевыми методами: dropout-слоями(случайное исключение части весов из агрегации), bagging (обучение семейства алгоритмов на подпространствах данных и получение конечного результата путем "голосования").
Заметим, что эти методы в поставленной задаче избыточны в связи с тем, что преимущественно область их применения связана с глубокими сетями(то есть сетями с достаточно большим количеством скрытых слоев) где вероятность ковариации между весами велика.