МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук им. И. И. Воровича

Кафедра теории упругости

Пандов Вячеслав Дмитриевич

ПРИМЕНЕНИЕ НЕЙРОСЕТЕЙ ДЛЯ РЕШЕНИЯ ОБРАТНЫХ ЗАДАЧ

КУРСОВАЯ РАБОТА
по направлению подготовки
01.03.02 – Прикладная математика и информатика

Научный руководитель –
доцент, д. ф.-м. н. Карякин Михаил Игорьевич

_____ _____
оценка (рейтинг) подпись руководителя

Ростов-на-Дону – 2020

Оглавление

1. Постановка задачи 3

2. Введение 4

3. Восстановление зависимостей с помощью нейронной сети 5

3.1. Компоненты размеченной выборки 5

3.2. Устройство нейронной сети 6

3.3. Обучение нейронной сети 8

4. Решение обратной краевой задачи с помощью нейронной сети 10

4.1. Решение прямых краевых задач для сбора обучающих данных 10

4.2. Обучение и тестирование нейронной сети 14

5. Заключение 19

6. Список литературы 20

1. Постановка задачи

1) Рассмотрим прямую и обратную задачу.

2) Сформулируем постановку приблизительного решения обратной задачи.

3) Рассмотрим все необходимые компоненты для обучения искусственных нейронных сетей.

4) Реализуем описанные компоненты и оценим результаты работы.

2. Введение

Для курсовой работы была поставлена задача решения обратных краевых задач для дифференциальных уравнений с начальными краевыми условиями с помощью нейронных сетей. Рассмотрим краевую задачу вида:

$$\begin{cases} \dfrac{d^2y}{dx^2} + f(x)y = 0 \\ \quad y(a) = y_a \\ \quad y(b) = y_b \end{cases}$$

Числа a, b, y_a, y_b – известны. Решением данной краевой задачи является массив точек:

$$\{ (x_i, y_i), i = 1 \dots n \},$$

где $x_1 = a, x_n = b, y_1 = y_a, y_n = y_b$

При решении прямой задачи необходимо найти решение $y(x)$, когда функция $f(x)$ – определена. Иными словами, существует некоторый оператор A отображающий зависимость:

$$A : F \to Y$$

$$F = \{ f_1, \dots, f_n \}, \qquad f_i = f(x_i)$$

$$Y = \{ y_1, \dots, y_n \}, \qquad y_i = y(x_i)$$

$$i = 1 \dots n$$

Обратная задача сводится к нахождению обратного оператора A^{-1}. С помощью некоторой нейронной сети необходимо найти оператор A^* – приближающий оператор A^{-1}:

$$A^* : Y \to F'$$

$$F' = \{ f_1 + \epsilon_1, \dots, f_n + \epsilon_n \}$$

$$\varepsilon = \{ \epsilon_1, \dots, \epsilon_n \}$$

где F' – множество, приближенное к F с некоторой погрешностью ε.

В ходе работы мы попробуем решить поставленную задачу с помощью искусственной нейронной сети, которая будет являться тем самым приближающим A^* оператором.

3. Восстановление зависимостей с помощью нейронной сети

3.1. Компоненты размеченной выборки

Прежде чем рассматривать процесс построения и обучения нейронной сети, рассмотрим понятие размеченной выборки.

Размеченная выборка состоит из N экземпляров. Для каждого экземпляра известны наборы из P нецелевых и Q целевых числовых признаков. Общий вид такой выборки приведен в табл. 1.

Таблица 1. Общий вид размеченной выборки

x_{11}	…	x_{1P}	y_{11}	…	y_{1Q}
x_{12}	…	x_{2P}	y_{12}	…	y_{2Q}
…	…	…	…	…	…
x_{N1}	…	x_{NP}	y_{N1}	…	y_{NQ}

Имея такой набор размеченных данных, можно обучить некоторую нейронную сеть. Однако, чтобы в процессе обучения можно было отслеживать качество работы обучаемой сети, данную выборку необходимо разделить на две подвыборки – тренировочную и проверочную. Тренировочная используется непосредственно для обучения модели, а проверочная для проверки результатов работы сети. Важно, что проверочная подвыборка никогда не используется в процессе обучения.

3.2. Устройство нейронной сети

Перед обсуждением устройства нейронной сети рассмотрим математическую модель простого нейрона[1].

$$y = \varphi(\langle x, w \rangle + b)$$

Нейрон принимает на вход вектор нецелевых признаков x. Здесь $\langle x, w \rangle$ – скалярное произведения вектора x на вектор весов w. Часто к этому произведению прибавляют значение b – вес смещения. Для нормализации выходного сигнала получившееся выражение проходит через некоторую функцию активации φ. Схематично устройство простого нейрона изображено на рисунке 1.

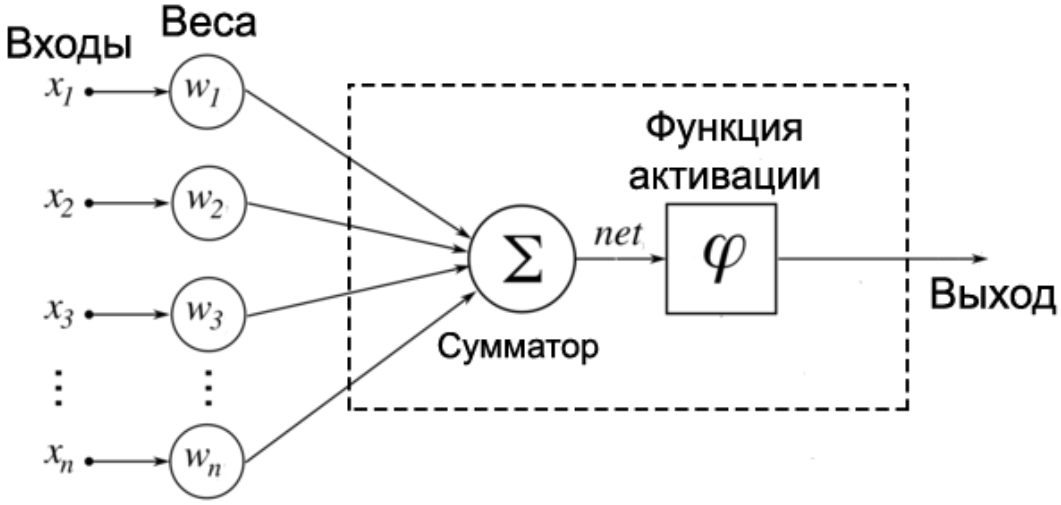

Рис. 1. Схема простого нейрона

В свою очередь, нейронная сеть состоит из нескольких слоев простых нейронов. Устройство нейронных сетей схематично изображено на рисунке 2.

[1] Нейронные сети: полный курс, 2-е издание. : Пер. с англ. – М. : Издательский дом "Вильямс", 2006. – 1104 с. : ил. – Парал. тит. англ.

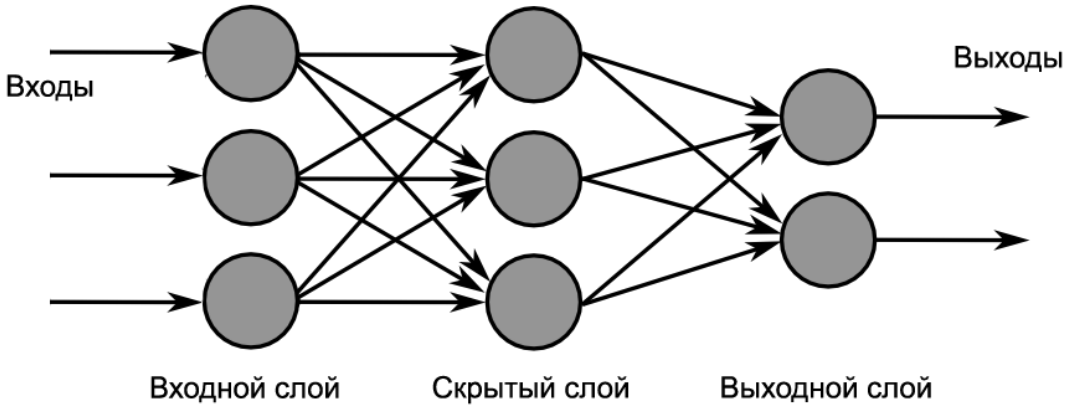

Рис. 2. Схема нейронной сети

Рассмотрим математическую модель нейронной сети[2]:

$$y_{i+1}^{k} = \varphi(\langle y_i, w_k \rangle + b_k)$$

Входы нейронов слоя $i + 1$ являются выходами нейронов слоя i. Выход -го нейрона слоя $i + 1$ рассчитывается как матричное произведение всех его входов со слоя i с весами k слоя, к которому впоследствии применена функция активации φ[3].

[2] Нейронные сети: полный курс, 2-е издание. : Пер. с англ. – М. : Издательский дом "Вильямс", 2006. – 1104 с. : ил. – Парал. тит. англ.

[3] Функция активации: [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Функция_активации (Дата обращения: 30.05.2020).

3.3. Обучение нейронной сети

Важную роль в обучении нейронной сети играет так называемая функция потерь, которая зависит от обучаемых параметров сети, называемых весами. Эта функция рассчитывает ошибку между предсказанным моделью значением и истинным значением из обучающей выборки. Есть важный критерий, которому должна удовлетворять такая функция – она должна быть дифференцируемой, чтобы по ней можно было посчитать градиент. Задача, поставленная в этой работе, является задачей регрессии. При решении таких задач хорошо подходит среднеквадратичная ошибка. Её иначе называют аббревиатурой MSE, что означает Mean Square Error.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2$$

где N – количество экземпляров для обучения, y_i – целевое значение, y_i' – предсказанное нейросетью значение. Все обучение сводиться к подбору параметров функции для поиска локального минимума функции потерь. Рассмотрим самый распространенный метод оптимизации, применяемый в нейронных сетях – градиентный спуск[4].

Для нахождения градиента функции потерь по её аргументам необходимо построить граф вычислений, идея которого заключается в разложении достаточно сложной функции по цепочке на более простые. Данный алгоритм называют методом обратного распространения ошибки.

Нахождения градиента функции потерь по весам сети позволяет найти направление локального максимума. Обновлять веса необходимо в обратном направлении, для достижения локального минимума по следующему правилу:

$$W_{t+1} = W_t - \alpha\nabla L(W)$$

[4] Современные численные методы оптимизации. Метод универсального градиентного спуска : учебное пособие / А. В. Гасников. – М. : МФТИ, 2018. – 286 с. – Изд. 2-е, доп.

где t – номер итерации, W – тензор весов сети, α – величина шага спуска, а ∇L – градиент функции потерь. Значение α ещё называют обучающим шагом, а итерацию t – эпохой. Весь процесс итеративный, и повторяется до тех пор, пока не будет выполнен некоторый критерий, вроде достижения удовлетворительных результатов.

Однако у метода градиентного спуска есть некоторые недостатки. Один из таких недостатков заключается в том, что для того, чтобы посчитать направление, в котором нужно производить спуск, необходимо посчитать значение функции потерь и её градиент по всем обучающих примерам. Обучающая выборка может быть очень большой. В этом случае потратиться очень много времени для того, чтобы сделать всего лишь один небольшой шаг. Поэтому на практике применяют различные модификации этого метода.

4. Решение обратной краевой задачи с помощью нейронной сети

4.1. Решение прямых краевых задач для сбора обучающих данных

Для решения обратной задачи нейронной сетью, необходимо обучить сеть на примерах. Для создания обучающих данных необходимо решить много краевых задач. Для этого напишем программу на языке Python[5] версии 3.7.7. В качестве инструмента для решения дифференциальных уравнений воспользуемся пакетом Sympy[6] версии 1.5.1. Реализуем метод, который принимает на вход начальные краевые условия и функцию $f(x)$, которую мы впоследствии будем предсказывать. И возвращает 2 словаря с переменными и функциями, чтобы впоследствии мы могли совершать подстановку значений. Реализация такого метода приведена в листинге 1.

Листинг 1. Метод на языке Python,
решающий краевую задачу с начальными условиями

```
from sympy import *
def solve_bvp(x_0: int, y_0: float, x_n: int, y_n: float,
              F: Function):
     x = Symbol('x')
     y = Function('y')(x)
     F = F(x)
     ode = Eq(y.diff(x, 2) + F * y, 0)
     ics = {
         y.subs(x, x_0): y_0,
         y.subs(x, x_n): y_n,
      }
      Y = dsolve(ode, y, ics=ics).rhs.evalf()
 return {'x': x, 'y': y}, {'Y': Y, 'F': F}
```

Также реализуем метод, который будет по полученному решению будет создавать и возвращать массивы точек по функциям $y(x)$ и $f(x)$. Для работы с массивами будем использовать пакет Numpy[7] версии 1.18.2. Реализация такого метода приведена в листинге 2.

---

[5] Интерпретатор Python: [Электронный ресурс]. URL: https://www.python.org/downloads/release/python-377/ (Дата обращения: 30.05.2020).
[6] Пакет Sympy: [Электронный ресурс]. URL: https://www.sympy.org/ru/index.html (Дата обращения: 30.05.2020).
[7] Пакет Numpy: [Электронный ресурс]. URL: https://numpy.org/ (Дата обращения: 30.05.2020).

Листинг 2. Метод на языке Python,

создающий массив точек по функциям

```
import numpy as np
def to_arrays(x_0: int, x_n: int,
              x: Symbol, y: Function, Y: Function, F: Function):
    n = (x_n - x_0) * 10
    aX = np.linspace(x_0, x_n, n, dtype=dtype)
    aY = np.array([Y.subs(x, i).evalf() for i in aX])
    aF = np.array([F.subs(x, i).evalf() for i in aX])
    arrays = {'Y(x)': np.array([aX, aY]),
              'F(x)': np.array([aX, aF])}
    return arrays
```

Реализуем ещё один метод, на этот раз для визуализации графиков по полученным массивам точек. Для работы с графиками будем использовать пакет Matplotlib[8] версии 3.2.1. Реализация такого метода приведена в листинге 3.

Листинг 3. Метод на языке Python,

визуализирующий графики по массивам точек

```
import matplotlib.pyplot as plt
def plot(arrays: dict, x_0: int, x_n: int):
    fig = plt.figure(figsize=(17, 7))
    ax = fig.gca()
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_xlim([x_0, x_n])
    ax.grid()
    for label, value in arrays.items():
        ax.plot(*value, label=label)
    ax.legend()
    return fig
```

Итак, весь необходимый инструментарий для решения и визуализации прямых задач готов. Убедимся в правильности его работы. Определим начальные краевые условия. Используя их, решим краевую задачу с

[8] Пакет Matplotlib: [Электронный ресурс]. URL: https://matplotlib.org/ (Дата обращения: 30.05.2020).

помощью реализованного метода. Создадим массив точек и визуализируем графики функций. Пример описанных действий представлен в листинге 4.

Листинг 4. Решение и визуализация краевой задачи на языке Python

```
xlim = {'x_0': 0, 'x_n': 15}
conditions = {
    'x_0': 0, 'x_n': 15,
    'y_0': 1, 'y_n': 1,
    'F': lambda x: -1 + 2 * x,
}
conditions.update(xlim)

variables, functions = solve_bvp(**conditions)
arrays = to_arrays(**xlim, **variables, **functions)
fig = plot(arrays, **xlim)
```

Полученные графики функций изображены на рисунке 3.

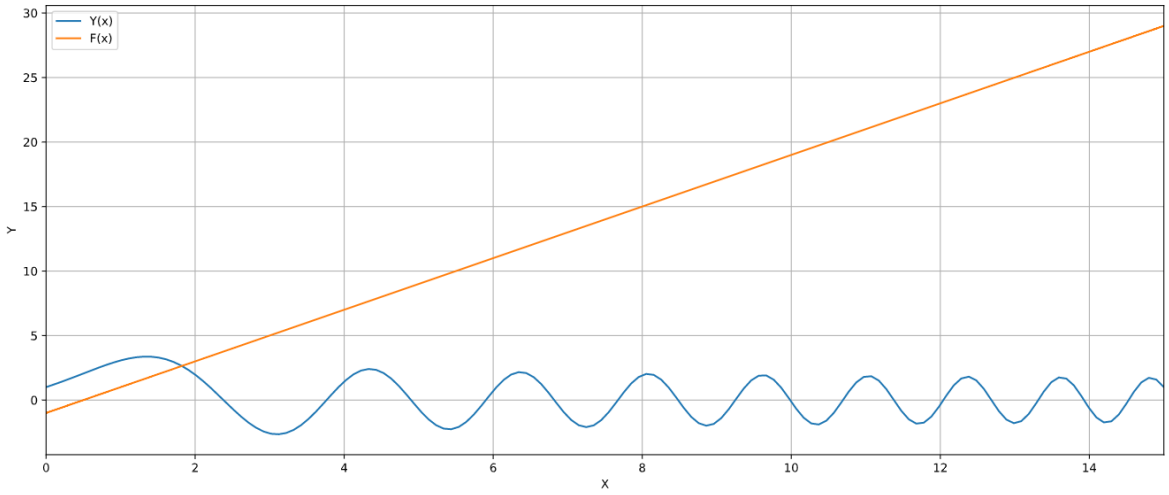

Рис. 3. Графики функций f(x) и $y(x)$

Алгоритм работает исправно, а значит можно переходить к генерации размеченных данных. Таким образом можно решить множество прямых задач и сохранить массивы точек различных функций $f(x)$ и $y(x)$ для создания размеченной выборки. Часть графиков такой выборки изображено на рисунке 4. Желтым графиком обозначена функция $f(x)$, а синим функция $y(x)$.

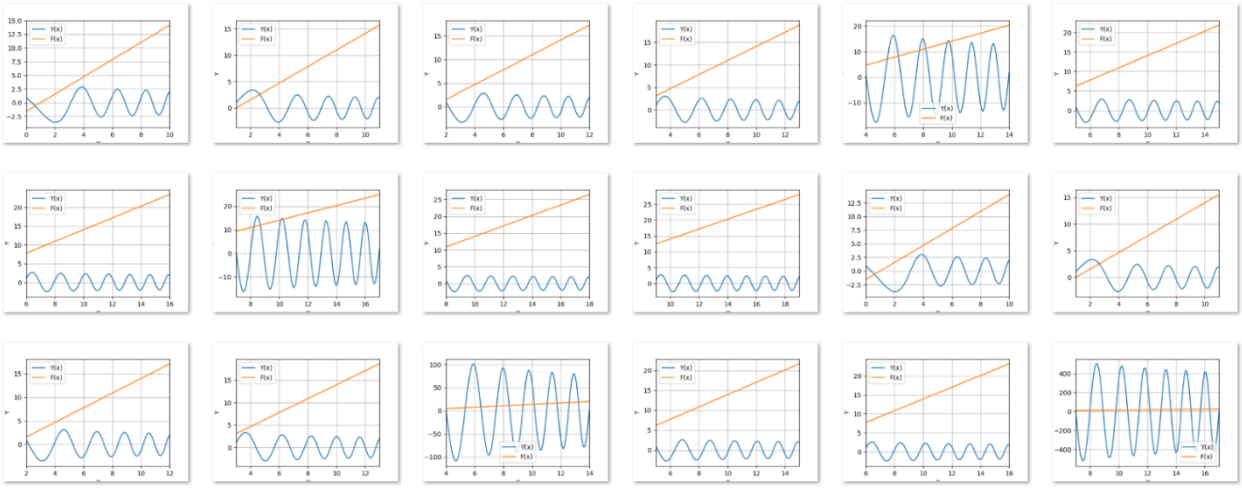

Рис. 4. Графическое представление
нескольких экземпляров созданной выборки

Чтобы использовать созданную выборку при обучении нейронной сети, необходимо сохранить каждый из полученных массивов и графиков. Пример сохранения представлен в листинге 5.

Листинг 5. Сохранение массивов точек и графиков выборки
на языке Python

```
np.save('dataset/array.npy', arrays)
fig.savefig('trains/figure.png')
```

4.2. Обучение и тестирование нейронной сети

Перейдем непосредственно к построению и обучению нейронной сети. Для работы с этими компонентами понадобится сразу несколько библиотек для работы с глубоким обучением:

- PyTorch[9] версии 1.4.0 – библиотека для работы с нейронными сетями.
- TensorBoard[10] версии 2.2.0 – инструмент для интерактивной работы с графиками.
- Catalyst[11] версии 20.5 – библиотека построенная на PyTorch упрощающая процесс обучения и тестирования моделей, с поддержкой TensorBoard.

Для начала опишем класс самой нейросети. В конструкторе определим архитектуру сети, а в методе forward укажем взаимодействие слоев. Модель будет принимать на вход 100 значений – массив точек функции $y(x)$. У сети будет всего 2 скрытых слоя, состоящих из 1024 нейронов в первом слое, и 100 нейронов в последнем – выходном слое. На 100 точек $y(x)$ модель должна будет предсказать 100 точек $f(x)$. Функцией активацией у каждого нейрона выступит гиперболический тангенс. Реализация класса модели представлена в листинге 6.

Листинг 6. Реализация класса нейронной сети

с помощью пакета torch на языке Python

```
import torch
class BVPNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = torch.nn.Linear(100, 1024)
        self.fc2 = torch.nn.Linear(1024, 100)

    def forward(self, x):
```

[9] Библиотека PyTorch: [Электронный ресурс]. URL: https://pytorch.org/ (Дата обращения: 30.05.2020).
[10] Библиотека TensorBoard: [Электронный ресурс]. URL: https://www.tensorflow.org/tensorboard (Дата обращения: 30.05.2020).
[11] Библиотека Catalyst: [Электронный ресурс]. URL: https://catalyst-team.github.io/catalyst/index.html (Дата обращения: 30.05.2020).

```
        x = self.fc1(x)
        x = torch.tanh(x)
        x = self.fc2(x)
        return x
```

Данный вид класса регламентирован библиотекой PyTorch. В родительском классе torch.nn.Module описаны компоненты и механизм работы модели.

Для работы с созданной выборкой реализуем соответствующий класс. Пример реализации представлен в листинге 7.

Листинг 7. Реализация класса выборки

с помощью пакета torch на языке Python

```
import os
class BVPDataset(torch.utils.data.Dataset):
    def __init__(self, is_train: bool):
        filepaths = sorted(os.listdir('dataset'))
        self.filepaths = filepaths[int(is_train)::2]

    def __len__(self):
        return len(self.filepaths)

    def __getitem__(self, index):
        filepath = self.filepaths[index]
        sample = np.load(filepath.as_posix())
        sample = torch.FloatTensor(sample)
        inputs = sample[0]
        targets = sample[1]
        return inputs, targets
```

Для экономии времени и ресурсов компьютера, экземпляры нашей выборки будут подгружаться в процессе обучения. Такой алгоритм позволяет реализовать класс DataLoader пакета torch.utils.data. Также необходимо определиться с функцией потерь и методом оптимизации. Как уже было озвучено ранее, для задач регрессий хорошо подходит MSE. В качестве метода оптимизации будем использовать стохастический градиентный спуск с импульсом. Реализации данной функции потерь и метода оптимизации имеются в библиотеке PyTorch. Инициализация всего описанного выше вместе с моделью представлена в листинге 8.

Листинг 8. Инициализация компонентов для обучения

```
train_dataset = BVPDataset(True)
```

```
valid_dataset = BVPDataset(False)
train_loader = torch.utils.data.DataLoader(train_dataset)
valid_loader = torch.utils.data.DataLoader(valid_dataset)


model = BVPNet()
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e2,
                            momentum=0.9, nesterov=True)
```

Все необходимые компоненты для обучения настроены. Чтобы сравнить результаты после обучения, посмотрим какие значения предсказывает нейронная сеть с необученными параметрами. На рисунке 5 зеленый график – это предсказанный моделью массив точек функции $f(x)$.

Рис. 5. Предсказанные значения необученной модели

Обучить сеть поможет класс SupervisedRunner пакета catalyst.dl. Данный класс имеет метод train, который принимает все компоненты для обучения. Применение этого метода приведено в листинге 9.

Листинг 9. Инициализация компонентов для обучения

```
from catalyst.dl import SupervisedRunner
runner.train(
    model=model,
    criterion=criterion,
```

```
    optimizer=optimizer,
    loaders={'train': train_loader, 'valid': valid_loader},
    num_epochs=100,
)
```

Дождавшись окончания обучения, посмотрим на результаты, изображенные на рисунке 6. Синий график отвечает за значение функции ошибки на тренировочной подвыборке, а красный график соответственно за значение на проверочной подвыборке. На графиках видно, как значение ошибки на обеих подвыборках со временем падает. К 93 эпохе значение ошибки на проверочной выборке достигает оптимального значения на всем обучении – примерно 3.7273.

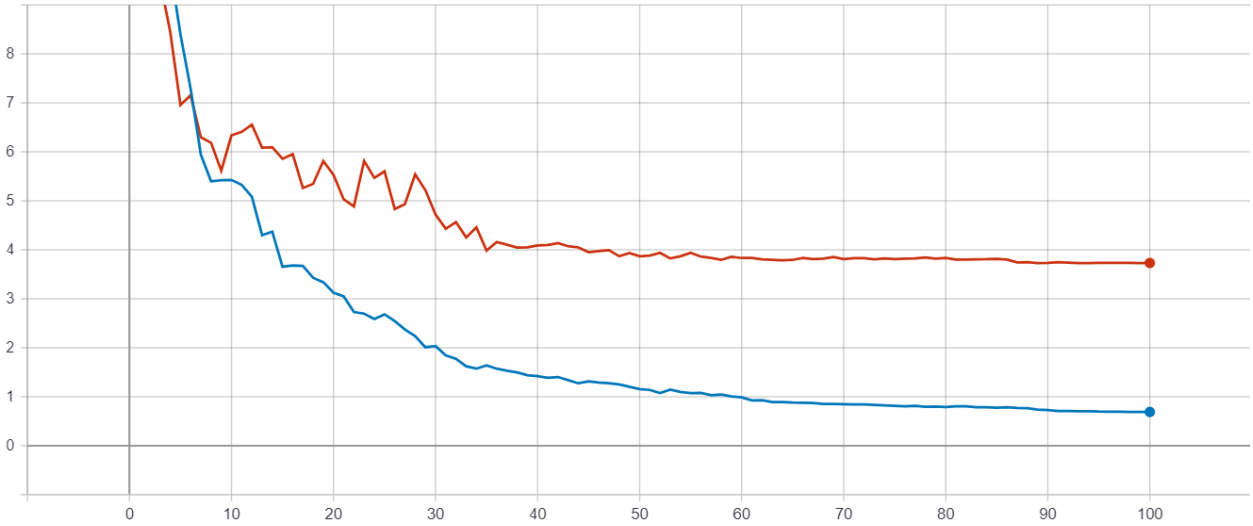

Рис .6. Результаты обучения

Теперь сравним результаты работы необученной модели с только что обученной. Предсказанный обученной моделью массив точек функции f(x) изображен на рисунке 7 зеленым графиком.

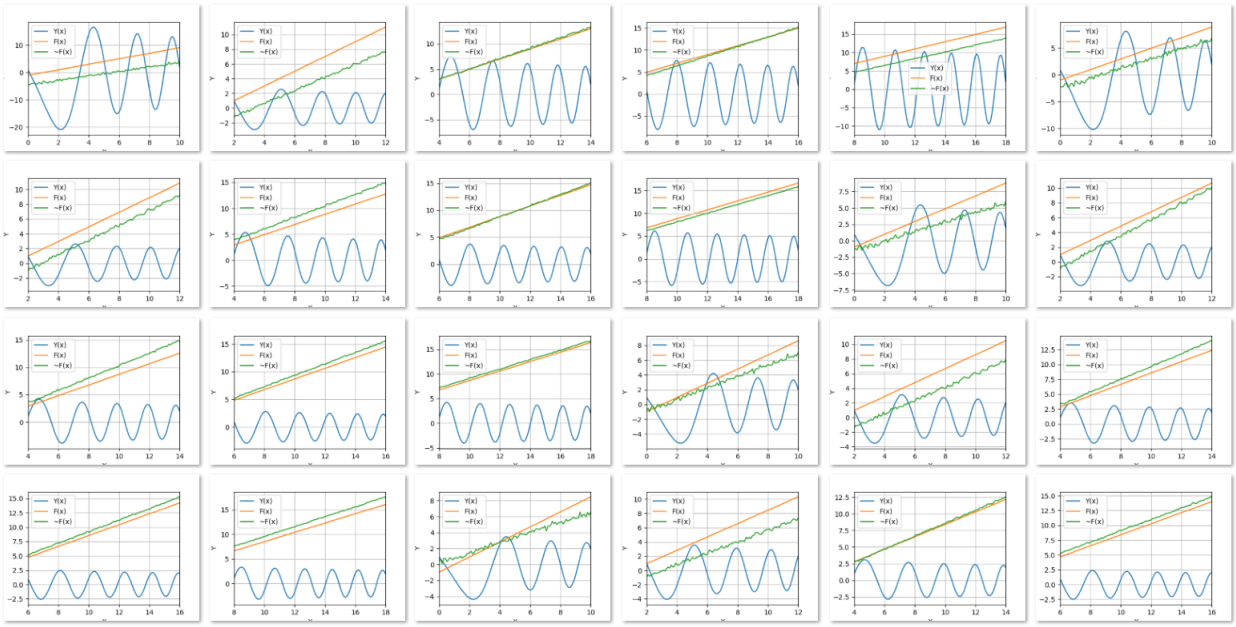

Рис. 7. Предсказанные значения обученной моделью

5. Заключение

Исходя из постановки цели данной курсовой работы, можно сделать вывод, что искусственные нейронные сети справляются с решением обратных задач. Наглядно был продемонстрирован весь процесс от сбора данных до обучения модели.

В работе полученной сети существует некоторая погрешность, которая изменяется от экземпляра к экземпляру. Однако есть ещё множество вариантов для улучшения текущего результата.

6. Список литературы

1. Библиотека Catalyst: [Электронный ресурс]. URL: https://catalyst-team.github.io/catalyst/index.html (Дата обращения: 30.05.2020).
2. Библиотека PyTorch: [Электронный ресурс]. URL: https://pytorch.org/ (Дата обращения: 30.05.2020).
3. Библиотека TensorBoard: [Электронный ресурс]. URL: https://www.tensorflow.org/tensorboard (Дата обращения: 30.05.2020).
4. Интерпретатор Python: [Электронный ресурс]. URL: https://www.python.org/downloads/release/python-377/ (Дата обращения: 30.05.2020).
5. Нейронные сети: полный курс, 2-е издание. : Пер. с англ. – М. : Издательский дом "Вильямс", 2006. – 1104 с. : ил. – Парал. тит. англ.
6. Современные численные методы оптимизации. Метод универсального градиентного спуска : учебное пособие / А. В. Гасников. – М. : МФТИ, 2018. – 286 с. – Изд. 2-е, доп.
7. Функция активации: [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Функция_активации (Дата обращения: 30.05.2020).
8. Пакет Matplotlib: [Электронный ресурс]. URL: https://matplotlib.org/ (Дата обращения: 30.05.2020).
9. Пакет Numpy: [Электронный ресурс]. URL: https://numpy.org/ (Дата обращения: 30.05.2020).
10. Пакет Sympy: [Электронный ресурс]. URL: https://www.sympy.org/ru/index.html (Дата обращения: 30.05.2020).

ОТЗЫВ

научного руководителя на курсовую работу В.Д.Пандова

«Применение нейросетей для решения обратных задач»

Коэффициентные обратные задачи для дифференциальных уравнений образуют широкий класс актуальных задач современного математического моделирования. В области механики твердого деформируемого тела они возникают, например, при определении таких свойств материала, как упругие модули, плотность, пористость и т.п. на основе данных статических или динамических экспериментов. Возможным подходом к преодолению трудностей, связанных с существенной некорректностью таких задач, является использование искусственных нейронных сетей.

Цель курсовой работы В.Д.Пандова состояла в знакомстве с данной предметной областью и разработке тестовой программы, связанной с обучением нейросети для определения функции, представляющей собой коэффициент дифференциального уравнения.

В ходе работы В.Д.Пандов познакомился с большим количеством библиотек для языка Питон, связанных, так или иначе, с построением и обучением искусственных нейронных сетей, а также с численным решением краевых задач для обыкновенных дифференциальных уравнений. Он зарекомендовал себя самостоятельным и активным студентом, способным анализировать сложные задачи математического моделирования и разрабатывать эффективный программный код.

Рассмотренный пример восстановления коэффициентов дифференциального уравнения по имеющейся информации о решении краевой задачи для этого уравнения показал возможность применения нейросетей в этой сфере, хотя открытых вопросов осталось еще много.

Учитывая объем проделанной работы и приобретенный в ходе этой работы опыт, считаю, что курсовая работа В.Д.Пандова «Применение нейросетей для решения обратных задач» соответствует требованиям, предъявляемым к таким работам, и может быть оценена на «отлично» (85 баллов).

Научный руководитель,
доктор физ.-мат. наук, доц. М.И.Карякин

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Южный Федеральный Университет

СПРАВКА
о результатах проверки текстового документа на наличие заимствований

Проверка выполнена в системе Антиплагиат.ВУЗ

Автор работы	**Пандов В.Д.**
Подразделение	ИММ и КН
Тип работы	Курсовая работа
Название работы	Применение нейросетей для решения обратных задач
Название файла	Пандов Курсовая.docx
Процент заимствования	**3.65 %**
Процент самоцитирования	**0.00 %**
Процент цитирования	**8.47 %**
Процент оригинальности	**87.88 %**
Дата проверки	**09:30:21 01 июня 2020г.**
Модули поиска	Модуль поиска ИПС "Адилет"; Модуль выделения библиографических записей; Сводная коллекция ЭБС; Модуль поиска "Интернет Плюс"; Коллекция РГБ; Цитирование; Модуль поиска переводных заимствований; Модуль поиска переводных заимствований по elibrary (EnRu); Модуль поиска переводных заимствований по интернет (EnRu); Модуль поиска переводных заимствований по Wiley (RuEn); Коллекция eLIBRARY.RU; Коллекция ГАРАНТ; Коллекция Медицина; Диссертации и авторефераты НББ; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска перефразирований Интернет; Коллекция Патенты; Модуль поиска "ЮФУ"; Модуль поиска общеупотребительных выражений; Кольцо вузов; Коллекция Wiley
Работу проверил	Карякин Михаил Игорьевич
	ФИО проверяющего
Дата подписи	
	Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.