УДК 621.3 DOI: 10.34759/trd-2021-118-10

Применение нейронной сети прямого распространения для локализации места удара микрочастиц о поверхность космического аппарата

Воронов К.Е.*, Григорьев Д.П., Телегин А.М.*****

Самарский национальный исследовательский университет имени академика С.П. Королева» Московское шоссе, 34, Самара, 443086, Россия

**e-mail: voronov.ke@ssau.ru*

***e-mail: dan-22225@yandex.ru*

****e-mail: talex85@mail.ru*

Статья поступила 04.05.2021

Аннотация

В данной статье кратко описаны основные виды архитектур нейронных сетей, и теория их работы. Составлен план решения задачи по обнаружению места удара с помощью нейронной сети. С использованием стандартных библиотек Keras и TensorFlow, на языке программирования python составлена модель нейронной сети для эксперимента по определению места удара. В конце статьи подведены итоги проделанной работы, достоинства, недостатки и перспективы рассмотренного метода.

Ключевые слова: космический мусор, космический аппарат, нейронная сеть прямого распространения, keras, tensorflow, локализация места удара.

Введение

Во время полёта космического аппарата, поверхность его корпуса постоянно находится под воздействием микрочастиц, представляющих собой микрометеороиды, и частицы космического мусора. Под космическим мусором понимаются отработавшие ресурс или вышедшие из строя космические аппараты, ступени ракет-носителей и их обломки [1]. При малых размерах частиц, космический аппарат получает повреждения, связанные с разгерметизацией блока, проникновением радиации и т.д. Своевременное и точное определение места удара, может позволить космонавтам предотвратить выход из строя аппарата, вовремя залатав место пробоя. Для локализации места удара, предлагается применить нейронную сеть, т.к. с совершенствованием компьютерных технологий, нейронные сети показывают себя с наилучшей стороны, в плане обработки и предсказывания информации. К примеру, в работе [2] нейронная сеть показала хорошие результаты в плане снижения вычислительных ресурсов по оценке прихода сигнала в радиолокационной системе. В нашей задаче требуется определить область поверхности космического аппарата, испытавшего удар от микрочастицы.

Для проведения эксперимента, вначале был изготовлен испытательный макет, представляющий собой алюминиевую пластину размерами 123х102 см, и толщиной 5 мм (рисунок 1) [3]. Пластина была поделена на 9 областей (A, B … I). К ней крепились четыре пьезодатчика, которые через зарядовые предусилители подключались к четырёхканальному осциллографу RIGOL с полосой частот 100 МГц, и частотой дискретизации 8Гвыб/с. Данные записывались на USB накопитель.

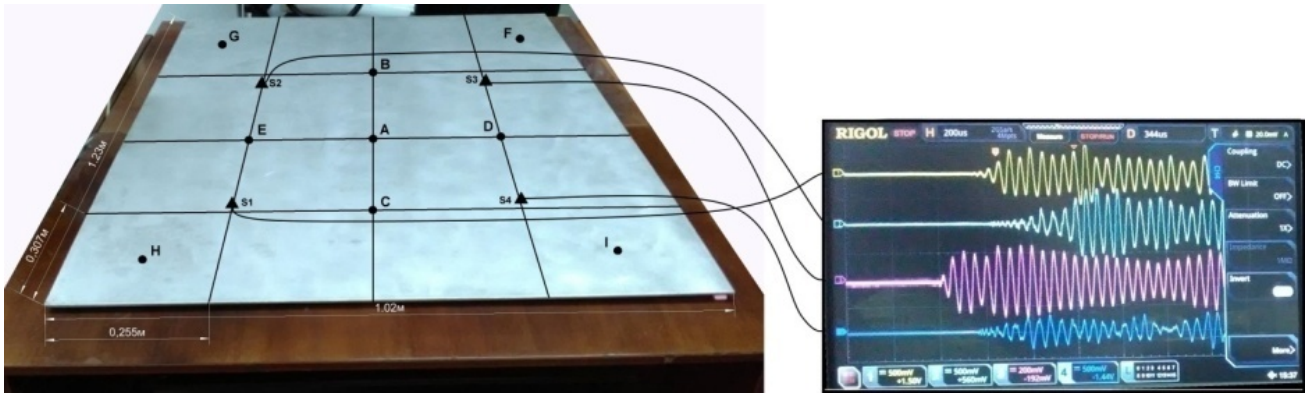

Рисунок 1 – Экспериментальный макет контроля поверхности космического аппарата

По полученным данным, с помощью предложенного в нашей работе [3] двукратного корреляционного метода, были получены времена прихода импульсов от каждого датчика, по которым были рассчитаны задержки сигналов (относительно первого сработанного датчика). Задача состоит в реализации нейронной сети, которая будет определять область пластины по полученным временным задержкам.

Основные положения нейронных сетей

Нейронная сеть - это виртуальная модель настоящей нейронной системы, которая является неотъемлемой частью головного мозга любого живого существа.

Любой нейрон можно представить по общей схеме на рисунке 2 [4]:

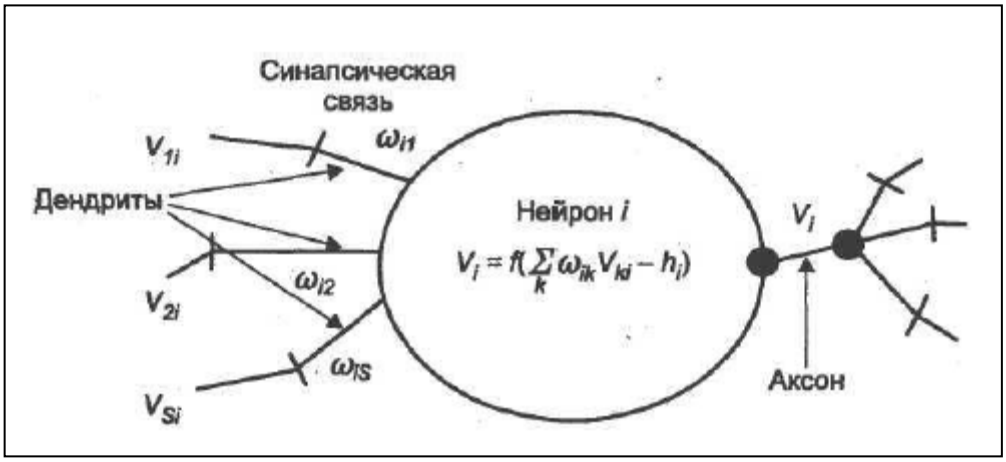

Рисунок 2 – Общая модель нейрона

V_{ki} - это энергетические доли импульсов V_k, выработанных другими нейронами, которые поступают на дендриты следующего нейрона i. Эти импульсы умножаются на ω_{ik} - веса дендритов. Внутри нейрона происходит суммирование всех поступивших на его вход импульсов V_{ki}, умноженных на соответствующий вес дендрита ω_{ik}, по которому они протекали. Произведение импульса и веса может сдвигаться на величину порога h_i. Далее, эта сумма преобразуется по некоторой функциональной зависимости $f(x)$ (функцией активации), формируя на своём выходе импульс V_i, который распространяется по другим нейронам через ветвления аксона. Значения синоптических весов ω_{ik} и порога h_i подлежат регулировке с помощью обратных связей [4].

Теперь перейдём от биологической модели к искусственной реализации, которая моделируется в компьютерных системах.

Искусственная нейронная сеть - это по сути своей инструмент, позволяющий с максимальной вероятностью установить желаемый результат на своём выходе, при загрузке в неё определённых входных данных. Она состоит из следующих частей [5]:

1. входного слоя;

2. скрытого слоя;

3. выходного слоя.

Входной и выходной слой представляет собой набор входных и выходных нейронов, которые являются по сути входными и выходными выборками каких-нибудь данных (имена людей, выборки сигнала или спектра и т.д.). Скрытый слой вмещает в себя несколько нейронов, по количеству, оптимальным образом подобранному для наилучшей аппроксимации верного ответа нейронной сети. Все нейроны соединяются между слоями через связующие связи синапсов, каждые из которых имеют свой весовой коэффициент.

Скрытых слоев может быть несколько, в зависимости от статистики получения верного ответа нейронной сети. Нейронная сеть - вещь не предсказуемая. Для неё нет какого-либо алгоритма или метода оптимального построения. Для каждой отдельной задачи есть своя структура нейронной сети. Один вопрос можно решить с минимальным количеством слоёв, а другой - с максимально большим. Также и с количеством нейронов. Всё это необходимо редактировать в процессе обучения модели, ссылаясь на точность предсказанного ответа.

Самой простой архитектурой нейронной сети является – нейронная сеть прямого распространения (Feed Forward Neural Networks) или однослойный персептрон (Single-Layer Perceptron) [6]. Такая архитектура является прямолинейной, и передаёт информацию от входа к выходу (рисунок 3). Кружками отмечены нейроны, а линиями – связующие их синапсические связи. Нейроны одного слоя

связаны с нейронами следующего слоя, однако сами нейроны между собой в рамках одного слоя не взаимодействуют. Такие сети обучаются, как правило, с помощью метода обратного распространения ошибки.

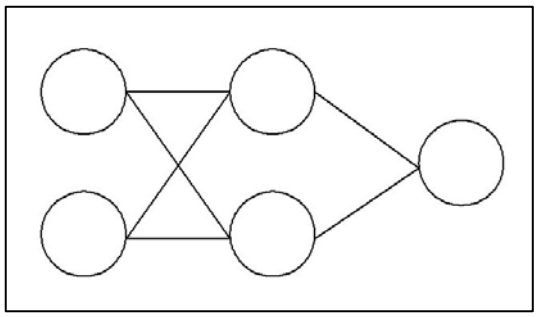

Рисунок 3 – Архитектура нейронной сети прямого распространения

Архитектура автокодировщика (Autoencoder) похожа по строению на нейронную сеть прямого распространения, но немного в другом исполнении. Архитектура производит кодирование, в смысле «сжатия» информации, восполняя её признаки на выходе. Скрытый слой меньше по количеству нейронов относительно входного и выходного слоя (рисунок 4). Сеть можно обучить методом обратного распространения ошибки, подавая входные данные и задавая ошибку, как разницу между выходом и входом (обучение без учителя).

Основная цель обучения автокодировщика – добиться того, чтобы входной вектор признаков вызывал отклик сети, равный входному вектору. Другими словами, находится аппроксимация такой функции, чтобы отклик нейронной сети равнялся значению входных признаков с точностью до заданного значения ошибки [7].

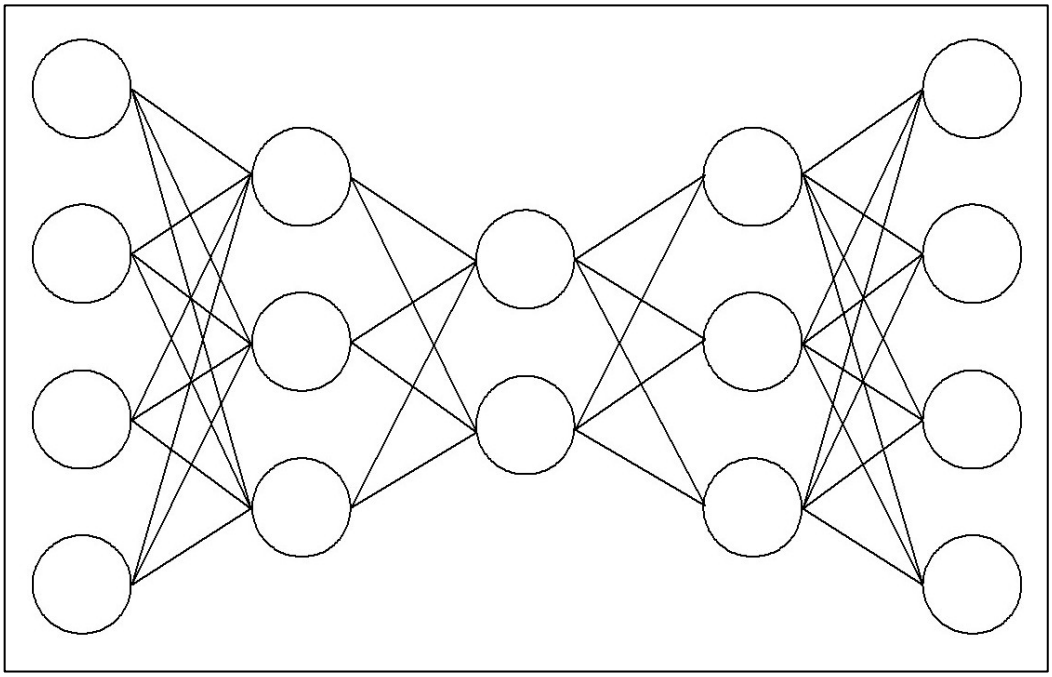

Рисунок 4 – Архитектура автокодировщика

Свёрточные нейронные сети (Convolutional Neural Networks) сильно отличаются от других видов сетей. Такие сети используются в основном для обработки изображений, а именно: распознавание людей, применение фильтров, реставрация, и т.д. Обучение происходит методом обратного распространения ошибки. Суть свёрточной сети заключается в том, что по входному 2D слою (матрице) скользит ядро меньшего размера, чем искомый слой [8]. Входных слоёв может быть несколько, формируя 3D матрицу - тензор (к примеру, три матрицы R, G и B цветного изображения). В ядре происходит суммирование произведений выделяемых им нейронов и весов. Такая операция есть не что иное, как свёртка, откуда и название архитектуры нейронной сети. На рисунке 5 слева показано двумерное представление свёрточной сети, а справа – одномерное.

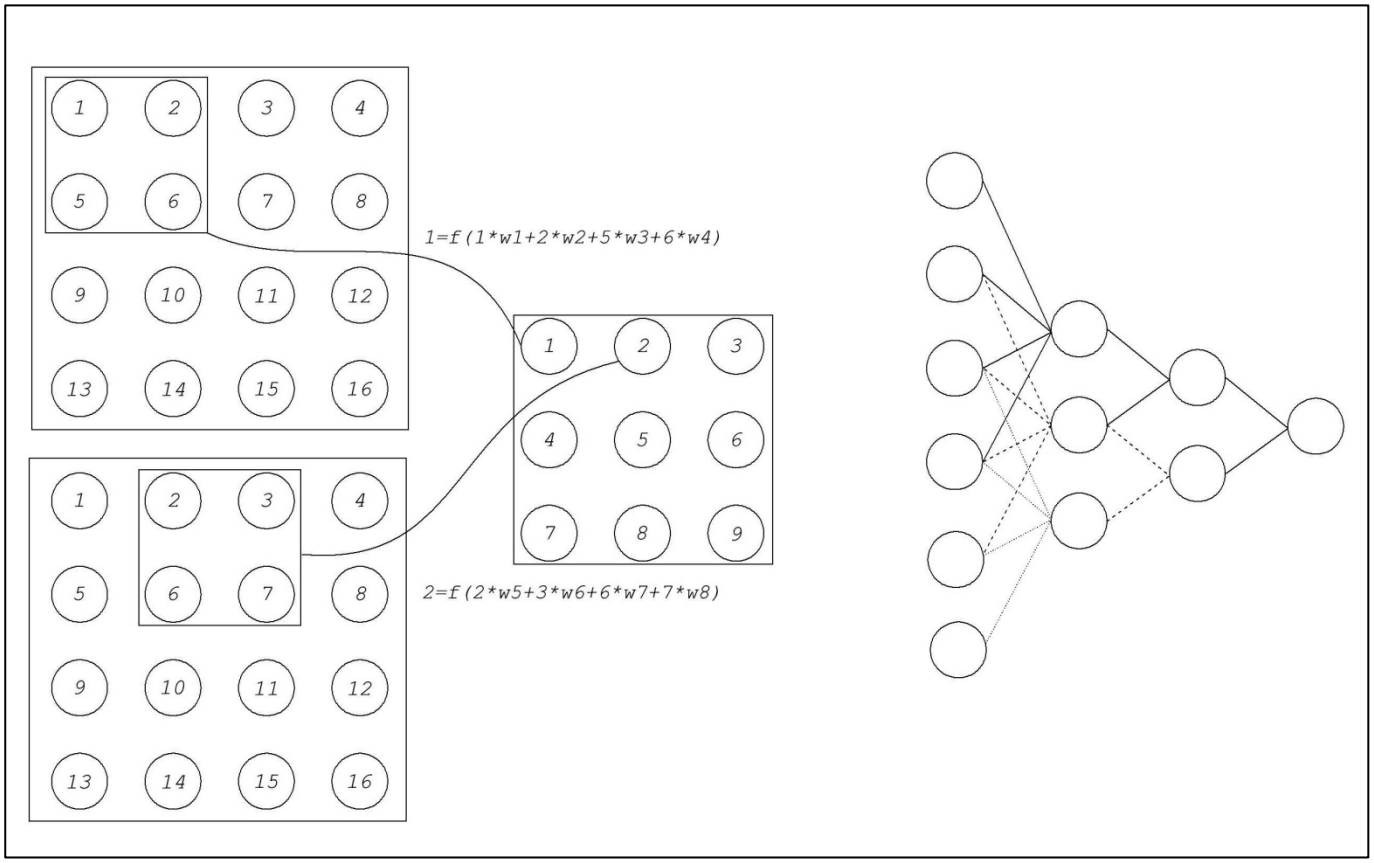

Рисунок 5 – Двумерное (слева) и одномерное (справа) представление свёрточной нейронной сети

После операции свёртки, как правило, применяется метод субдискретизации или подвыборки (subsampling / pooling). На этом этапе уменьшается размерность полученной матрицы, например, с размера 8x8 на 4x4. Графически, методика расчёта такая же, как и при свёртке, за исключением того, что ядро не проходит повторно по пройденным ранее нейронам, т.е. не пересекается (рисунок 6) [9]. В расчёте используется либо субдискретизация «meanpooling» (по среднему значению), либо «maxpooling» (по максимальному значению) выделенных ядром нейронов.

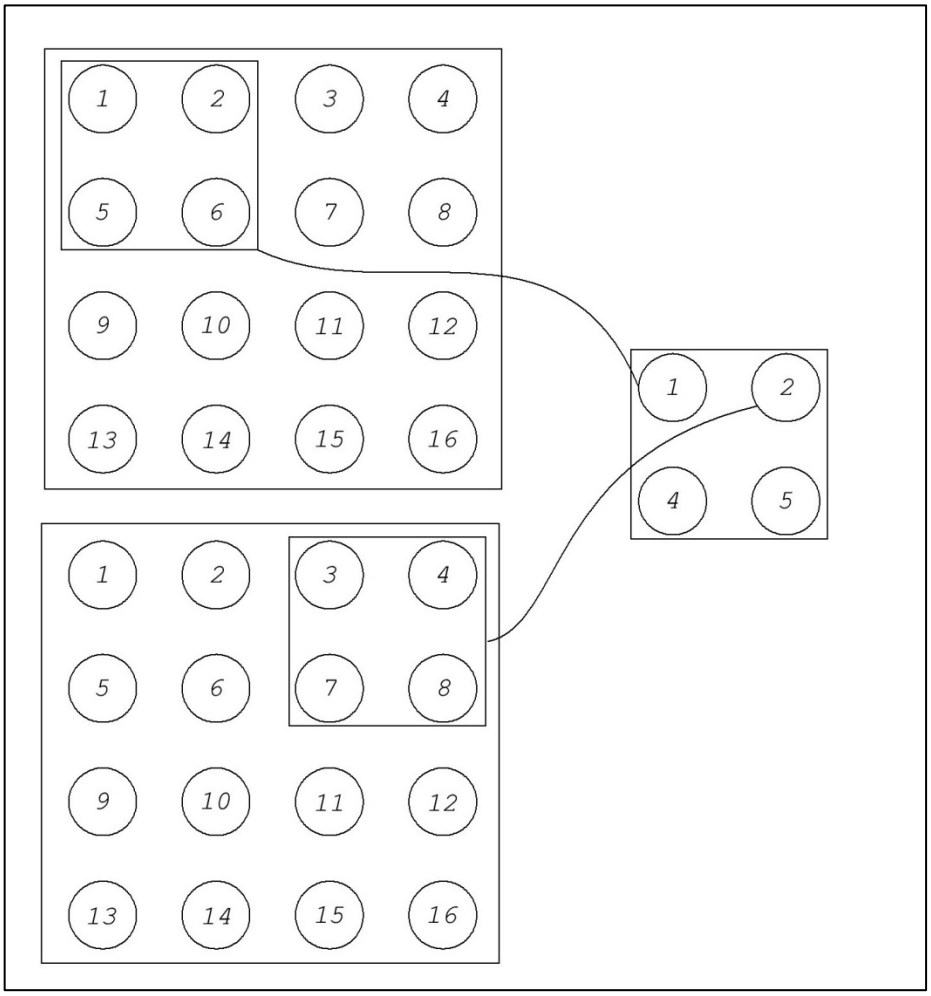

Рисунок 6 – Субдискретизация слоя свёрточной сети

Наконец, при достижении минимальной матрицы 2x2, необходимо преобразовать двумерную информацию в одномерный набор данных (рисунок 7). Происходит это обычным разложением матриц в одномерные массивы [9]. Поэтому количество нейронов в первом полносвязном слое будет равно (1):

$$N = i \times j \times k, \quad (1)$$

где i, j - длина и ширина одной конечной матрицы соответственно (в нейронах);

k - количество конечных матриц.

Далее, полносвязный слой (по принципу прямого распространения) формирует выходные нейроны, которые несут в себе какой-либо признак обработанной исходной матрицы (например, виды опознаваемого животного).

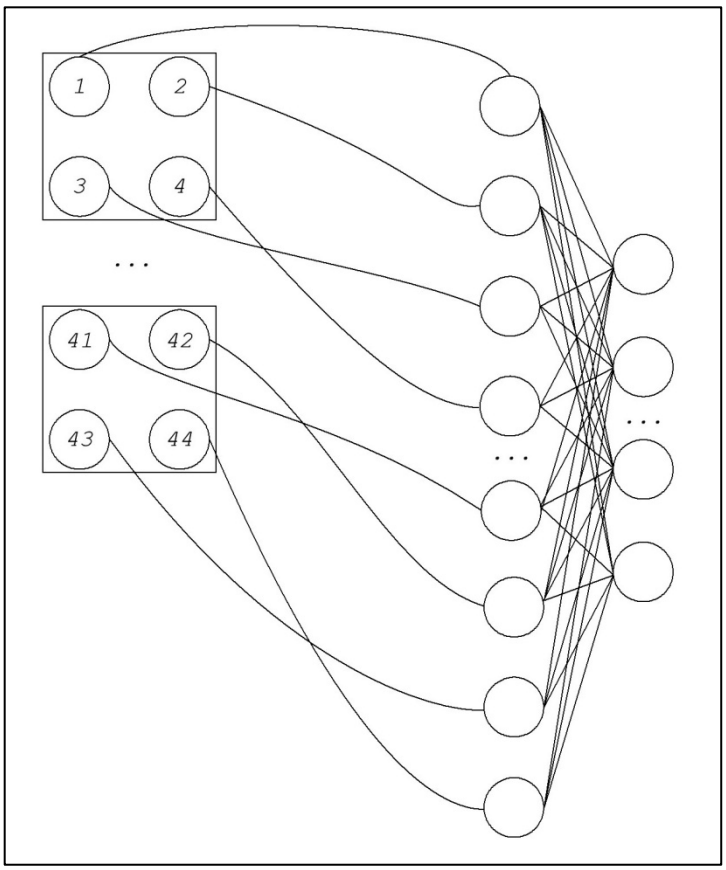

Рисунок 7 – Полносвязный слой

Принцип обучения нейронной сети методом обратного распространения ошибки

Схема нейронной сети прямого распространения показана на рисунке 8.

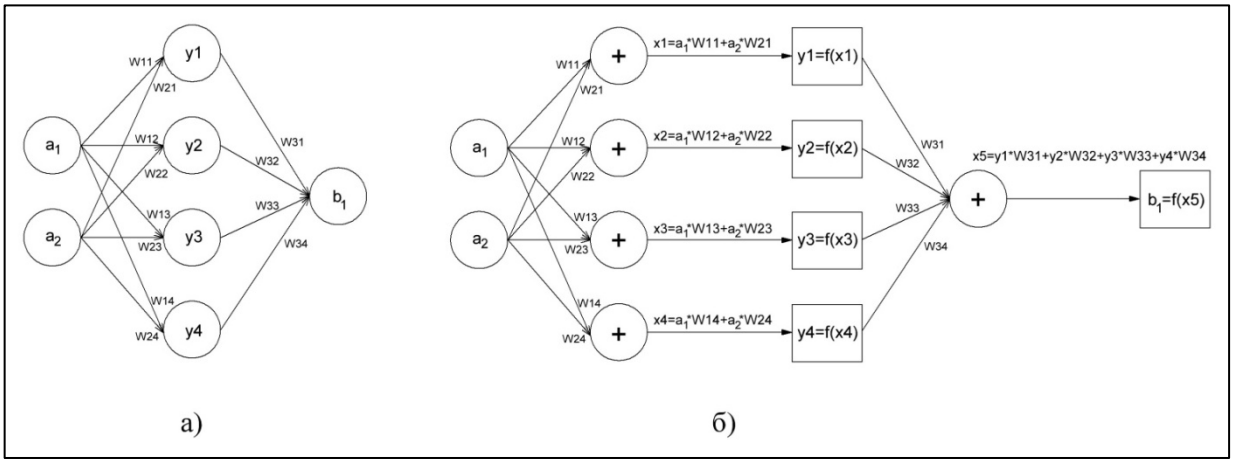

Рисунок 8- Нейронная сеть прямого распространения:

а) каноничная форма; б) развёрнутая форма

Нейронная сеть на рисунке 8 имеет входной слой с входными данными «a1» и «a2», один скрытый слой с промежуточными величинами y1 –y4, и выходной слой с одним выходным значением b1. Для примера допустим, что во входные нейроны поступают числа «8» и «2», а на выходе формируется значение в долях от 0,0 до 1,0, которое бы отвечало на вопрос «имеют ли эти два числа наибольший общий делитель?». Естественно, чем ближе значение к 1,0 (100%), то вероятнее ответ - «имеют», а чем ближе к 0,0 (0%), то ответ - «не имеют».

Нейроны скрытого слоя исполняют роль сумматоров величин каждого нейрона предыдущего слоя, умноженных на весовой коэффициент соответствующего синапса w_{ij}. Так, из рисунка 8, для каждого нейрона скрытого слоя $j = \overline{0,4}$ можно записать (2):

$$y_j = \sum_{i=1}^{N} (a_i \times w_{ij}), \qquad (2)$$

где N - количество нейронов во входном слое (в нашем случае N=2).

Однако при обычной операции суммирования, значения в нейронах были бы больших порядков, и с ними было бы сложно работать. Поэтому результат подвергают прохождению через функцию активации, которая «сжимает» результат, например, в диапазоне от 0,0 до 1,0. Таким образом, получается перенос больших чисел из бесконечного диапазона в процентное (долевое) соотношение. Чем больше число, тем сильнее оно стремится к единице, а чем меньше - тем сильнее стремится к нулю. Самые распространённые функции активации показаны на рисунке 9 и в таблице 1.

Таблица 1 - Виды функций активации

Название	Формула
Логистический сигмоид	$f(x) = \frac{1}{1+e^{-x}}$
Гиперболический тангенс	$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
Relu	$f(x) = 0, x < 0$ $f(x) = x, x > 0$
Гауссова	$f(x) = exp\left(-\frac{x^2}{2}\right)$
Линейная	$f(x) = x$
Пороговая (Хэвисайда)	$f(x) = 0, x < 0$ $f(x) = 1, x > 0$

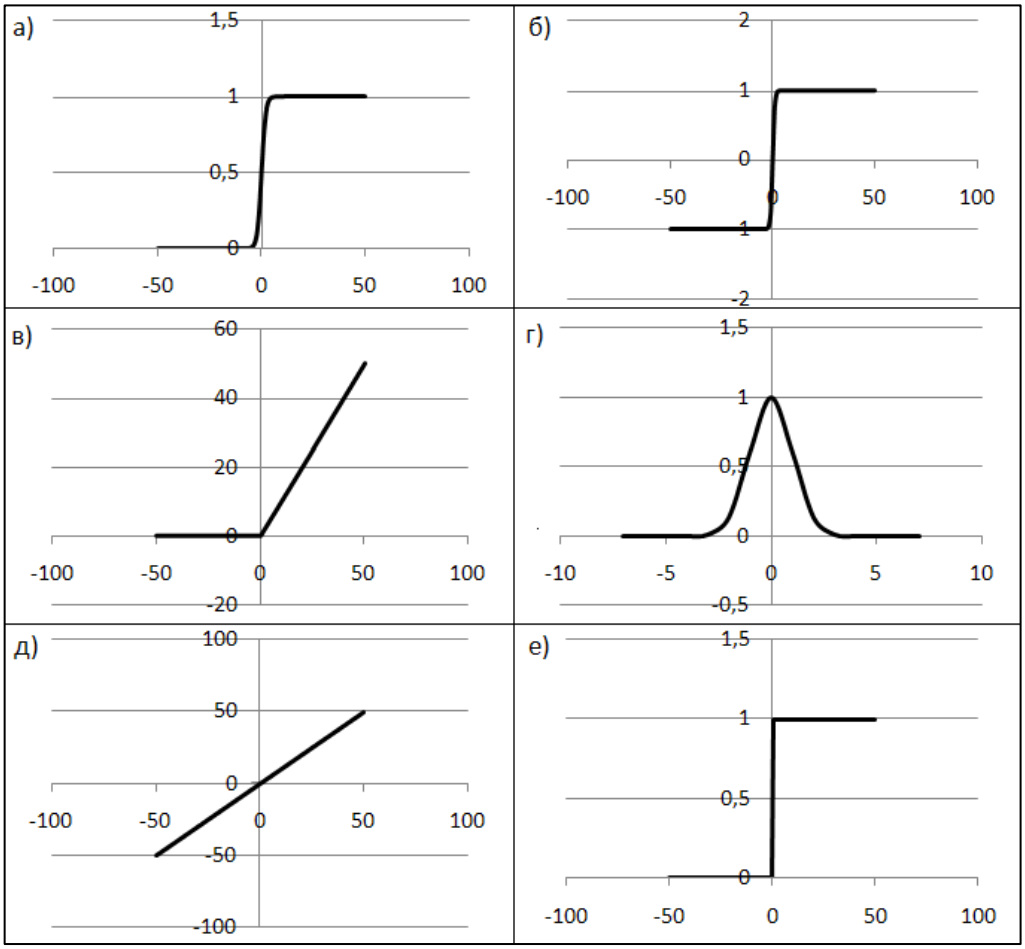

Рисунок 9 - Функции активации: а) «логистический сигмоид»; б) гиперболический тангенс; в) relu; г) гауссова функция; д) линейная функция; е) пороговая функция (Хэвисайда)

Каждая функция активации предназначается для определённого выделения данных, получаемых в нейронах. Так, например: сигмоида применяется для нелинейного преобразования данных в диапазоне от 0 до 1; гиперболический тангенс используется для диапазона данных от -1 до 1; функция Relu отбрасывает отрицательные данные, пропуская линейно только положительные числа; гауссова функция выделяет числа в диапазоне от 0 до 1 по нормальному распределению; линейная функция не оказывает «преобразования» данных и описывает процесс

обычного суммирования в нейронах; пороговая функция работает в режиме обнаружения, когда результат в нейроне достиг требуемого порога.

Поэтому выражение (2), при прохождении через функцию сигмоиды, запишется в следующем виде (3):

$$y_j = f_{sigm}\left(\sum_{i=1}^{2}(a_i \times w_{ij})\right), \quad (3)$$

Наконец, выходной нейрон запишется как (4):

$$b_1 = f_{sigm}\left(\sum_{j=1}^{4}(y_j \times w_{3j})\right), \quad (4)$$

Когда происходит расчёт по функции активации, результирующие значения нейрона перемещаются строго по линии функции, как она задана. Иногда бывают случаи, что решения скрываются за пределами функции активации, т.е. слева, или справа. Чтобы получить эти решения, необходимо сдвигать функцию активации вдоль оси абсцисс. Покажем на примере логистической сигмоиды (рисунок 10).

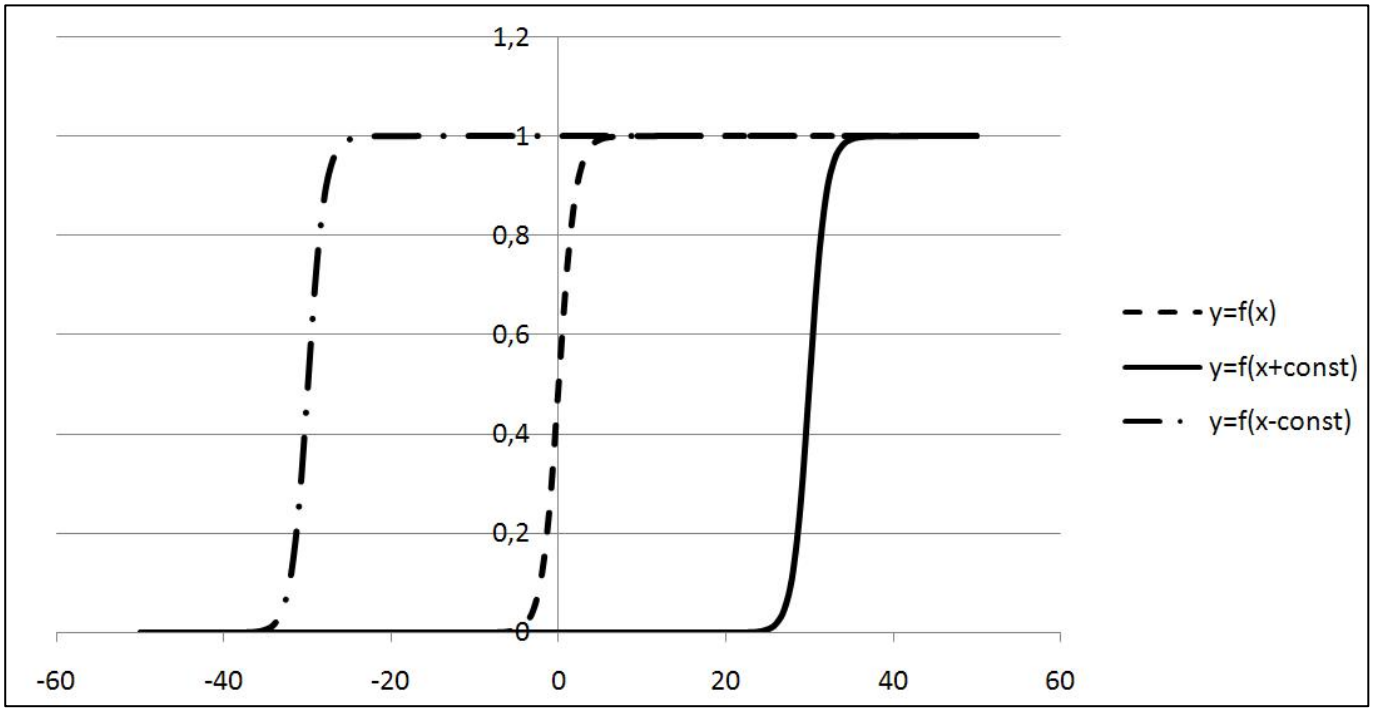

Рисунок 10 - Смещение функции активации логистический сигмоид

Для такой операции используется отдельный нейрон смещения, который не имеет входа, а только выход. Соответственно, выражение (4) примет вид (5):

$$b_1 = f_{sigm}\left(\sum_{j=1}^{4}\left[(y_j \times w_{3j}) + z_1\right]\right), \quad (5)$$

Все значения весов задаются произвольно при первом запуске нейронной сети. На выходе её будут также произвольные значения, имеющие большие расхождения с требуемым результатом. На данном этапе начинается подготовка к обучению нейронной сети.

Так как у нас есть требуемый выходной ответ (тренировочный образец), выполняется расчёт ошибки (6):

$$\Delta b = b_{real} - b_{ideal}, \quad (6)$$

где b_{real} - полученное значение на выходе нейронной сети;

b_{ideal} - требуемое значение на выходе нейронной сети.

Далее начинается процесс обратного распространения ошибки, который включает в себя запись ошибок в каждом нейроне, двигаясь от выходного слоя к входному слою.

В каждом нейроне последнего скрытого слоя записывается сумма ошибок каждого выходного нейрона, умноженная на вес связующих их синапсов. В нашем случае, имеется только один выходной нейрон, поэтому для каждого нейрона скрытого слоя справедлива формула (7):

$$y_j = \sum_{j=1}^{4} \Delta b \times w_{3j}, \quad (7)$$

Такая операция распространяется на все последующие скрытые слои. Входной слой не меняется, т.к. значениями нейронов являются входные данные сети.

После обратного распространения ошибки, обучение переходит на финальный этап - корректировка весов. В момент корректировки весов, необходимо рассчитать величину изменения веса по методу градиентного спуска ошибки.

Градиентный спуск - это способ обучения сети, который представляет собой постепенное уменьшение ошибки, путём приближения к локальному минимуму производной функции ошибки, распространяясь по её градиенту (8):

$$f(x) \to \min_{x \in R} f(x); \quad (8)$$

Градиент функции ошибки обозначается следующим образом (9) [10]:

$$\nabla f_{MSE}(x) = \begin{pmatrix} \frac{\partial f_{MSE}}{\partial x_1} \\ \frac{\partial f_{MSE}}{\partial x_2} \\ \dots \\ \frac{\partial f_{MSE}}{\partial x_n} \end{pmatrix}; \quad (9)$$

В качестве функции ошибки может являться среднеквадратичное отклонение (Mean Square Error), равное (10) [11]:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2, \quad (10)$$

где Y_i - полученный результат;

$\hat{Y}_i$ - требуемый результат;

n - количество итераций (эпох).

Функция ошибки имеет, как правило, нелинейный характер (рисунок 11), т.е. непрерывно возрастает, убывает, или не изменяется. Градиент $\nabla f_{MSE}(x)$ указывает направление, в котором функция ошибки быстрее всего возрастает. Отрицательный градиент $-\nabla f_{MSE}(x)$ укажет на обратное направление, которое как раз будет соответствовать минимуму функции потери [10].

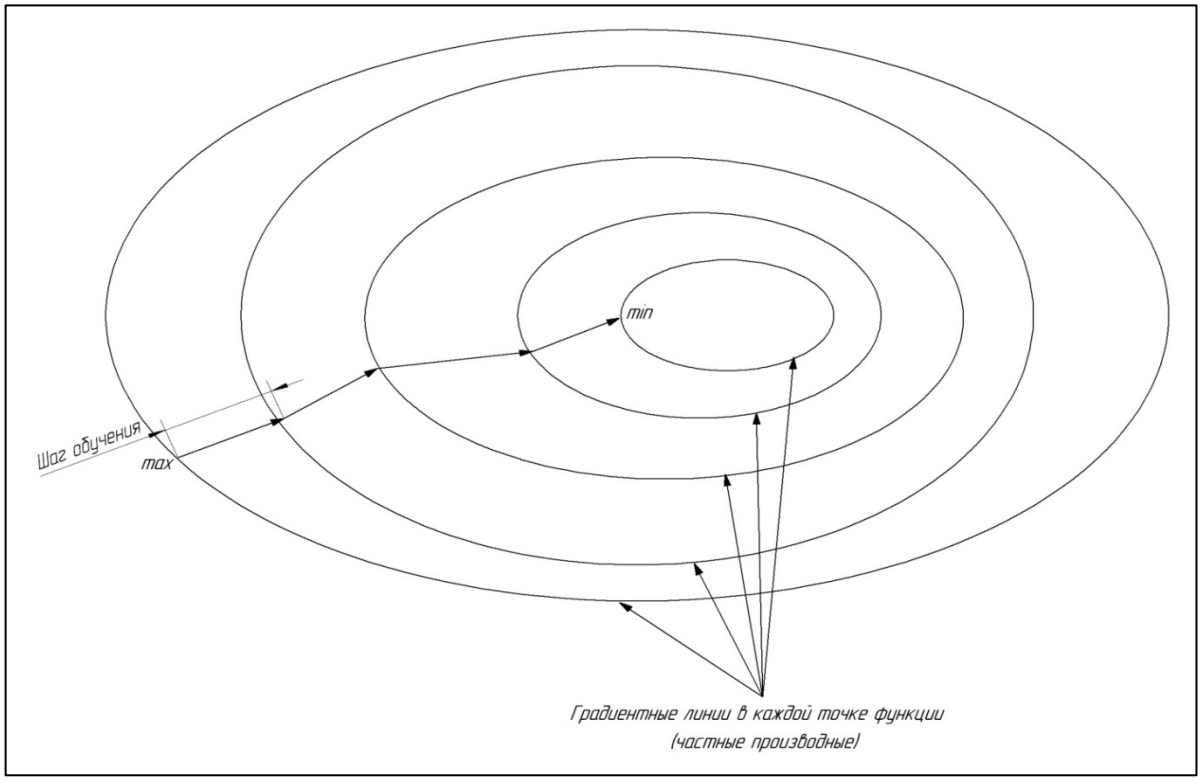

Рисунок 11- Иллюстрация градиентного спуска по функции потерь (MSE)

Основным параметром является коэффициент (шаг) обучения η, которым регулируется скорость обучения. Однако при неправильном обращении с ним, возможны два неблагоприятных исхода ситуации:

- локальный минимум функции потерь никогда не будет достигнут из-за большого шага перемещения (рисунок 12-б);
- локальный минимум функции потерь будет достигаться спустя большое количество эпох (рисунок 12-а), и останется в минимуме.

Первый случай происходит обычно из-за большого коэффициента обучения, а второй случай - из-за малого коэффициента обучения. Поэтому, необходимо с каждой итерации подбирать коэффициент обучения по некоторой функциональной зависимости. При первом запуске данный коэффициент подбирается произвольно.

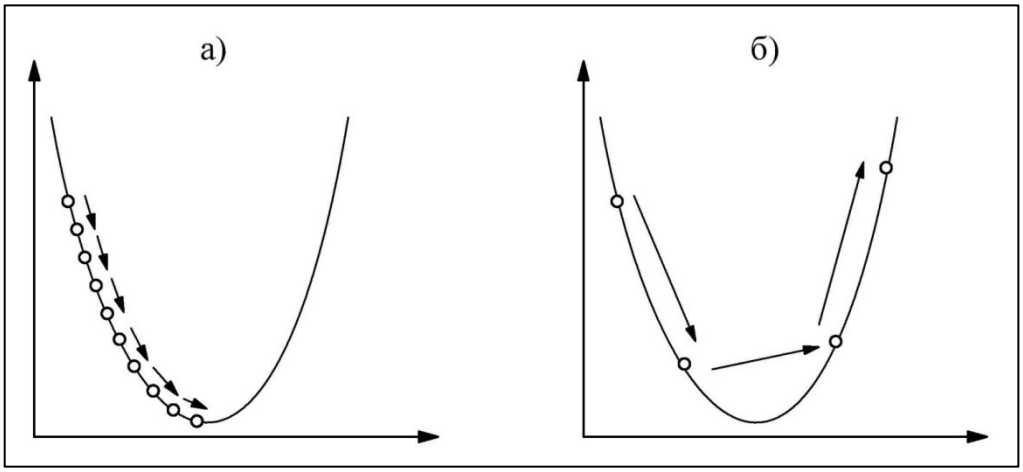

Рисунок 12 - Графический пример скорости градиентного спуска при разных η

Если коэффициент обучения не изменять, то нейронная сеть будет обучаться с постоянной скоростью, которая может повлечь за собой выше описанные проблемы. Поэтому в качестве функции коэффициента обучения применяют линейную (11) или экспоненциальную (12) зависимости [10] (или оптимизационные моменты):

$$\eta = \eta_0\left(1-\frac{t}{T}\right), \tag{111}$$

$$\eta = \eta_0 e^{-\frac{t}{T}}, \tag{122}$$

где t - это прошедшее время с начала обучения (число эпох);

T - параметр, определяющий, как быстро будет меняться коэффициент η.

Метод градиентного спуска сложен в плане вычислений, т.к. требуется пройти все тренировочные данные для корректировки весов (13). Поэтому на практике применяют его аналог – стохастический градиентный спуск (14), позволяющий корректировать весы после каждой входной выборки. Корректировка весов, обычно, представляет собой разность предыдущего значения веса с произведением η, и градиентного спуска функции потерь [10]:

$$\theta_t = \theta_{t-1} - \eta \sum_{(x,y)\in D} \nabla E(f(x,\theta_{t-1}), y), \quad (133)$$

$$\theta_t = \theta_{t-1} - \eta \nabla E(f(x_t,\theta_{t-1}), y_t), \quad (144)$$

где θ_t, θ_{t-1} – новый и старый вес соответственно;

x – входные тренировочные данные;

y - выходные желаемые данные;

$E(f(x,\theta_{t-1}), y)$ – функция ошибки от предсказанного ответа и y.

Моделирование нейронной сети прямого распространения на языке python

В качестве архитектуры выберем нейронную сеть прямого распространения, так как она проста в реализации, и представляет собой самый распространённый тип нейронной сети, применяющейся в оценке или прогнозировании данных, как например, в работе [2] или [12].

Нейронная сеть в настоящем эксперименте была написана на python с использованием библиотеки Keras в связке с TensorFlow.

TensorFlow – это библиотека машинного обучения от Google с открытым исходным кодом [13], предназначенная для автоматического моделирования нейронных сетей, а именно: построения, тренировки и тестирования.

Keras – это высокоуровневая API, предоставляющая удобные инструменты по работе с библиотеками нейронных сетей TensorFlow, Theano или CNTK [14].

Приступим к реализации нейронной сети в виде программы. Количество нейронов входного слоя будет определяться количеством времён запаздывания

сигнала, которые будут указаны в отдельном текстовом файле (тренировочном сете): 0; 0,0004; 0,0005; 0,0007 (в секундах). Т.е. на вход будет подаваться 4 числа.

Количество нейронов выходного слоя, будет равно 9 областям пластины (A - I), которые будут заданы в выходном файле как вероятности мест удара для каждого случая (файла) временных задержек: 1; 0; 0; 0; 0; 0; 0; 0; 0. Подробно покажем на рисунке 13. Входные нейроны помечены временными задержками t_{1-4} , а выходные нейроны – местами удара P_{1-9} (place). Чем ближе выходной нейрон к 1,0, тем выше вероятность того, что в этом месте произошёл удар.

Нейронам P1, P2 … P9 будут соответствовать области пластины A, B … I соответственно.

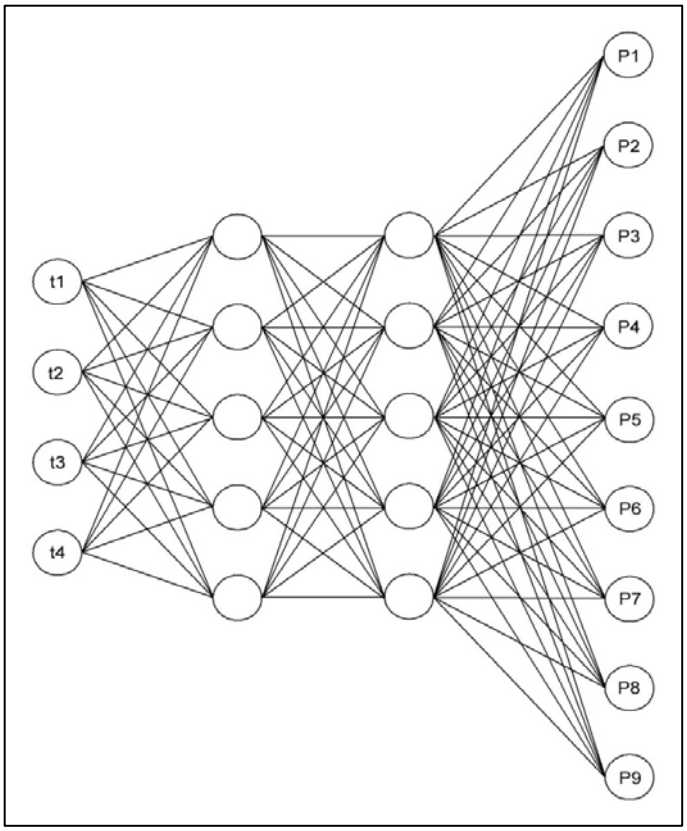

Рисунок 13 – Графический пример реализации нейронной сети прямого распространения для решения задачи локализации места удара

Перед началом работы, установим в среду разработки Visual Studio интерпретатор Python 3.7. После, установим следующие пакеты, которые пригодятся для работы с нейросетью (установка происходит в окне управления пакетами командой «pip install “namepackage”»): Keras; Numpy (для формирования кортежей); Pandas (для формирования DataFrame).

Следующий шаг – подготовка входных и выходных данных для обучения. Нейронная сеть сначала должна тренироваться, а потом пройти тестирование. Поэтому нам нужно иметь 2 набора данных - тренировочный и тестовый набор. Каждый набор включает в себя входные и выходные данные. Все эти данные у нас подготовлены в текстовых файлах с расширением *.txt. Файлы сгенерированы конструктором, который дополнительно был написан на языке C#. Он формирует 900 файлов, с временными задержками, по 100 файлов на каждую область пластины. Количество файлов увеличено, благодаря некоторому разбросу значений по оси времени, моделируя тем самым неоднородность скорости распространения волн по поверхности пластины. Таким образом, будут созданы четыре массива (в дальнейшем мы преобразуем их в кортежи, а те в DataFrame):

```
time = []  # Тренировочные времена
time_ = []  # Тестовые времена
place = []  # Тренировочные области
place_ = []  # Тестовые области
```

Теперь зададим входные параметры нейронной сети:

```
batch_size = 100        # количество поступающих входных сетов (файлов)
num_epochs = 800     # количество эпох
```

```
hidden_size = 512      # количество нейронов в скрытом слое
num_train = 900        # общее количество входных трен. сетов (файлов)
num_test = 45          # общее количество входных тест. сетов (файлов)
num_in = 4             # размер входного вектора (4 временные задержки)
num_out = 9            # размер выходного вектора (9 областей пластины)
```

Параметр batch_size формирует блоки данных для помещения их на вход нейронной сети. Другими словами, вместо 900 наших сетов, на вход нейронной сети (в рамках одной эпохи) будет поступать по очереди batch_size сетов. Т.к. мы указали батч равный 100, то за одну эпоху нейронная сеть будет обучаться 9 раз по 100 сетам, что в сумме даст общее обучение по 900 сетам. Если количество батчей не будет кратно общему количеству сетов (например, batch_size = 99), то сначала будет подано на вход 9 батчей по 99 файлов (891 сет), а последний батч, будет меньшего размера, т.е. 9, что в сумме даст 900 [14].

Количество выходных файлов не задаётся, т.к. подразумевается, что на каждый входной тренировочный (или тестовый) сет есть свой выходной файл.

Теперь задаём тип модели нейронной сети (последовательный): model = Sequential(), и заложим в неё 4 слоя:

```
model.add(Dense(hidden_size, activation='tanh', input_dim = num_in))
model.add(Dense(hidden_size,activation='tanh'))
model.add(Dense(hidden_size,activation='tanh'))
model.add(Dense(num_out, activation='tanh'))
```

Слои задаются методом «Dense», в аргументах которого выступают [14]:

- размер слоя (количество нейронов);

- тип функции активации слоя;
- для первого скрытого слоя необходимо указать размер входного вектора «input_dim = 4».
- Последний слой описывает выходной слой, с размером равным выходному вектору«num_out = 9».

Теперь соберём модель нейросети методом Compile:

```
model.compile(loss = 'mean_squared_error', optimizer = 'adam',
metrics = ['accuracy'])
```

Аргументами этого метода служат [14]:

- наименование выбранной функции потерь (MSE - среднеквадратичное отклонение);
- вид оптимизатора расчёта стохастического градиентного спуска, (Adam – Adaptive Moment Estimation, метод накопления движения) [15];
- метрики, по которой сопровождается статистическое отображение процесса обучения нейронной сети в консоли (accuracy - расчёт точности ответа нейронной сети).

Наконец, запустим тренировку методом: history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = num_epochs). В этот метод передаются [14]:

- входные тренировочные выборки (временные задержки);
- выходные тренировочные выборки (области пластины);

- количество батчей;
- количество эпох.

Тестирование нейронной сети осуществим через следующий метод: score = model.evaluate(X_test, Y_test). Как можно видеть, аргументами служат входные и выходные тестовые выборки. Можно дополнительно задать batch_size, однако, без его указания, он по умолчанию будет выбран batch_size = 32 [14] (справедливо и для метода model.fit).

Используем уже обученную модель нейронной сети для выведения результатов областей пластины, используя метод: answer = model.predict() (в целых числах и в десятичных) при 512 нейронах:

```
[[ 0. -0. -0.  0. -0.  0.  0. -0.  0.]
 [ 0.  1.  0.  0. -0.  0. -0. -0.  0.]
 [-0. -0.  1.  0. -0.  0. -0.  0.  0.]
 [ 0.  0. -0.  0.  0.  0. -0. -0.  0.]
 [-0. -0. -0.  0.  1. -0.  0.  0.  0.]
 [ 0. -0. -0.  0.  0.  0.  0.  0. -0.]
 [ 0. -0.  0. -0.  0.  0.  1. -0.  0.]
 [-0.  0.  0. -0.  0.  0.  0.  1.  0.]
[ 0. -0.  0.  0.  0. -0.  0. -0.  1.]]
```

```
[[ 0.436 -0.017 -0.002  0.257 -0.145  0.33   0.157 -0.129  0.073]
 [ 0.04   0.975  0.004  0.035 -0.019  0.027 -0.007 -0.011  0.003]
 [-0.032 -0.073  0.964  0.057 -0.057  0.051 -0.026  0.01   0.049]
 [ 0.182  0.015 -0.008  0.466  0.14   0.152 -0.267 -0.093  0.355]
 [-0.169 -0.016 -0.014  0.159  0.653 -0.039  0.112  0.044  0.163]
 [ 0.32  -0.015 -0.005  0.213  0.001  0.365  0.028  0.16  -0.071]
 [ 0.179 -0.014  0.002 -0.156  0.166  0.062  0.622 -0.003  0.067]
 [-0.131  0.01   0.006 -0.027  0.065  0.107  0.009  0.705  0.124]
[ 0.009 -0.048  0.006  0.318  0.077 -0.142  0.026 -0.027  0.608]]
```

при 700 нейронах:

```
[[ 1. -0. -0.  0. -0.  0. -0. -0. -0.]
 [-0.  1. -0. -0. -0.  0.  0. -0.  0.]
 [ 0. -0.  1. -0. -0.  0.  0. -0. -0.]
 [-0. -0. -0.  1. -0.  0.  0.  0. -0.]
```

```
[-0. -0.  0. -0.  1.  0. -0.  0. -0.]
[-0. -0. -0.  0.  0.  1. -0.  0. -0.]
[-0. -0. -0. -0.  0.  0.  1.  0. -0.]
[-0. -0. -0.  0.  0.  0. -0.  1.  0.]
[-0. -0.  0.  0.  0.  0.  0.  0.  1.]]

[[ 0.788 -0.018 -0.006  0.116 -0.034  0.298 -0.011 -0.005 -0.016]
 [-0.015  0.985 -0.01  -0.007 -0.027  0.09   0.022 -0.012  0.018]
 [ 0.01  -0.013  0.986 -0.013 -0.002  0.008  0.007 -0.012 -0.001]
 [-0.207 -0.026 -0.009  0.958 -0.021  0.148  0.029  0.017 -0.018]
 [-0.005 -0.031  0.008 -0.011  0.917  0.048 -0.075  0.069 -0.05 ]
 [-0.212 -0.006 -0.004  0.071  0.039  0.895 -0.064  0.008 -0.028]
 [-0.032 -0.018 -0.004 -0.024  0.09   0.051  0.937  0.037 -0.018]
 [-0.076 -0.015 -0.015  0.011  0.068  0.042 -0.079  0.943  0.017]
[-0.032 -0.043  0.011  0.002  0.057  0.013  0.041  0.052  0.955]]
```

Как видно из результатов, увеличением количества нейронов получилось добиться большей точности прогнозирования результата. При 700 нейронах практически все области определились верно. Однако при 512 нейронах области также верно были определены (видно по наибольшим значениям в строках матрицы). Ошибка возникала в данном случае из-за округления по 0,5. Ниже приведём графики потерь и точности ответа нейронной сети, которые наглядно показывают прогресс её обучения. Потери (представляющие собой результаты функции потерь) в данном случае убывают, а точность желаемого ответа – возрастает (по ординатам приведены процентные соотношения в долях).

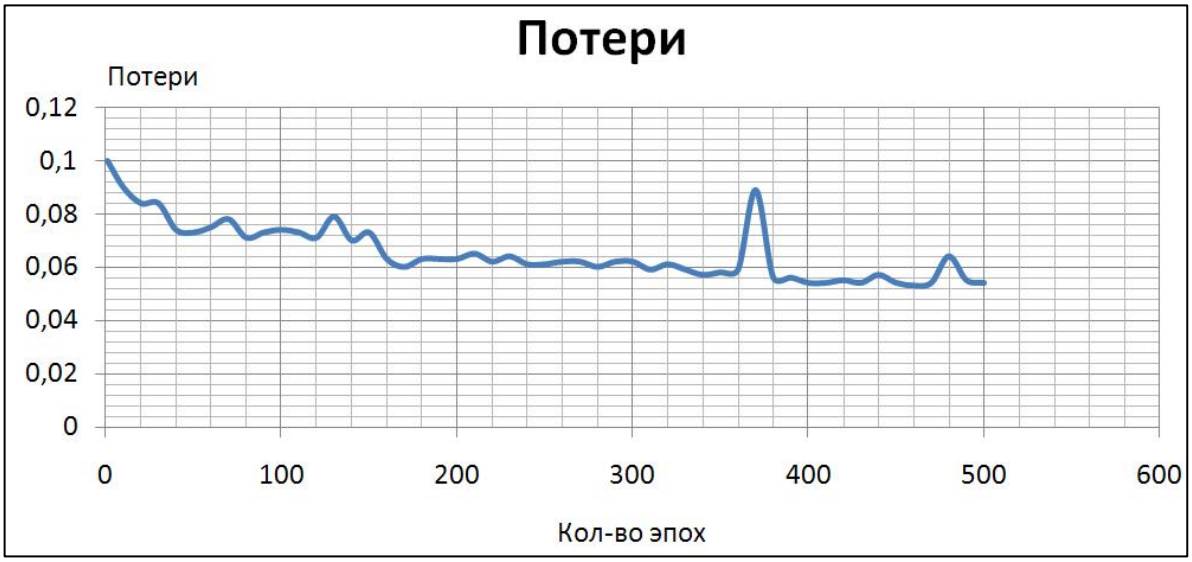

Рисунок 14 – График потерь ответа нейронной сети в зависимости от числа эпох

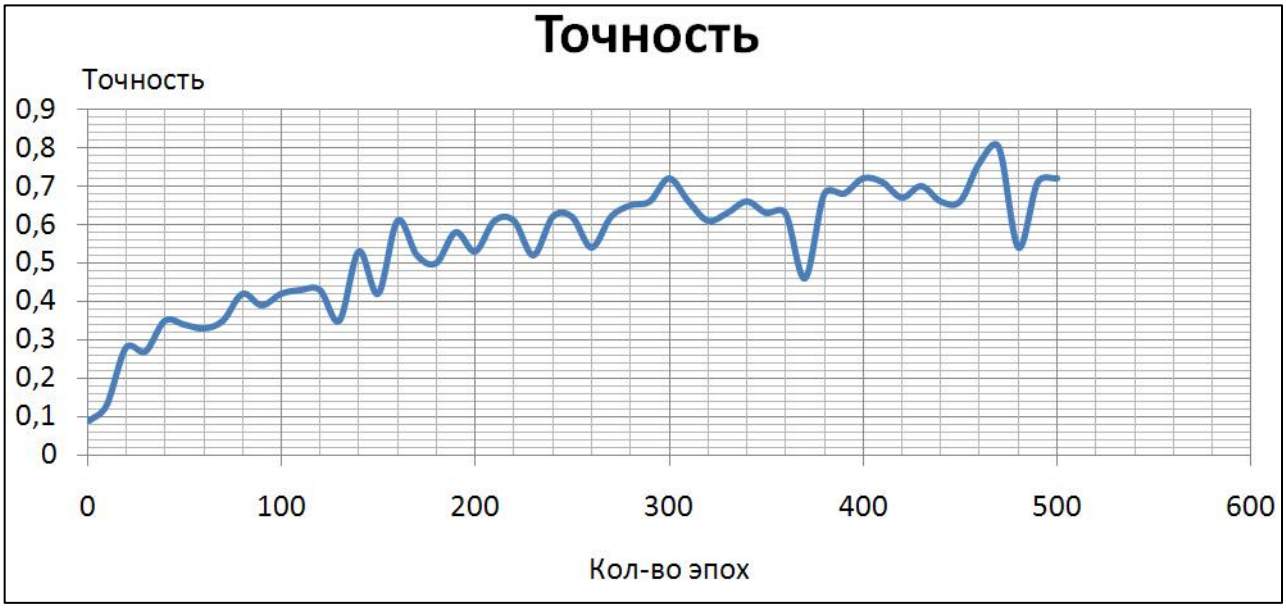

Рисунок 15 – График точности ответа нейронной сети в зависимости от числа эпох

Отметим тот факт, что дополнительно повышения точности можно достичь изменением количества датчиков. Так в работе [16] используется 8 датчиков, а в работах [17] и [18] применяется 6 и 3 датчика соответственно. Когда приходится задумываться о габаритных и экономических показателях устройства, то выгоднее применять меньшее количество датчиков. Например, в работе [19] проводился опыт

с одним датчиком и статистической моделью, а в [20] применялись нейронные сети, с архитектурами автокодировщика и свёртки (также для одного датчика). Таким образом, решать данную задачу необходимо путём нахождения компромисса, между описанными выше ограничениями.

Заключение

В данной статье продемонстрирована работа нейронной сети прямого распространения по обнаружению областей удара микрометеороидов на поверхности космического корабля.

Достоинством данного метода здесь является повышенная точность прогнозирования места удара. В качестве обучающих данных были использованы экспериментальные замеры из нашего опыта [13], полученные с помощью испытательного макета и корреляционной методики определения времени появления импульсов. При увеличении экспериментальных данных, и характеристик модели нейронной сети, можно повысить точность и для других областей пластины, приближаясь к более малым областям, или к их большему количеству.

Недостатком данного метода является ограниченность обучения нейронной сети в рамках лабораторных условий. Обучение необходимо провести с дополнительным воздействием шумовых сигналов, и желательно на реальных конструкциях космического аппарата. Также для данного метода необходимо вычислять дополнительно временные задержки для нейронной сети, что усложняет задачу в плане быстродействия и математического аппарата.

Перспективы данного метода заключаются в обучении нейронных сетей на более простых видах сигналов, полученных с датчиков контроля поверхности, и их ускорения, за счёт аппаратной реализации нейронной сети (например, на ПЛИС).

Библиографический список

1. Баркова М.Е. Космический аппарат для утилизации космического мусора в околоземном пространстве // Труды МАИ. 2018. № 103. URL: http://trudymai.ru/published.php?ID=100712

2. Ефимов Е.Н., Шевгунов Т.Я. Формирование оценки направления прихода сигнала с использованием искусственных нейронных сетей // Труды МАИ. 2015. № 82. URL: http://trudymai.ru/published.php?ID=58786

3. Воронов К.Е., Григорьев Д.П., Телегин А.М. Исследование алгоритмов для системы контроля поверхности космического аппарата на основе пьезодатчиков // Авиакосмическое приборостроение. 2021. № 1. С. 40 - 50. DOI: 10.25791/aviakosmos.1.2021.1200

4. Барский А.Б. Нейронные сети: распознавание, управление, принятие решений. – М.: Финансы и статистика, 2004. – 176 с.

5. Рашид Т. Создаём нейронную сеть. - СПб.: Альфа-книга, 2017. - 272 с.

6. Аникеев М.В., Бабенко Л.К., Макаревич О.Б. Обзор современных типов нейронных сетей // Радіоелектроніка, інформатика, управління. 2001. № 1. С. 48 - 56.

7. Акинина Н.В., Акинин М.В., Соколова А.В., Никифоров М.Б., Таганов А.И. Автоэнкодер: подход к понижению размерности векторного пространства с

контролируемой потерей информации // Известия ТулГУ. Технические науки. 2016. № 9. С. 3 - 12.

8. Землевский А.Д. Исследование архитектуры сверточных нейронных сетей для задачи распознавания образов // Вестник науки и образования. 2017. Т. 2. № 6 (30). С. 36 – 43.

9. Бредихин А.И. Алгоритмы обучения свёрточных нейронных сетей // Вестник Югорского государственного университета. 2019. № 1 (52). С. 41 - 54. DOI: 10.17816/byusu20190141-54

10. Николенко С.И. Глубокое обучение. – СПб.: Питер, 2018. – 480 с.

11. Mean Square Error. URL: https://en.wikipedia.org/wiki/Mean_squared_error.

12. Шумских И.Ю., Пиганов М.Н. Использование нейроимитатора для прогнозирования показателей надёжности космической аппаратуры // Региональная научно-практическая конференция, посвященная 50-летию первого полета человека в космос: тезисы докладов (Самара, 14 – 15 апреля 2011). – Самара: СГАУ им. С.П. Королева. 2011. С. 205 - 207.

13. A complex open-source machine learning platform. URL: https://www.tensorflow.org/

14. Keras: the python deep learning API. URL: https://keras.io/

15. Diederik Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization, Cornell University, 2017. URL: https://arxiv.org/abs/1412.6980

16. Lei Qi et al. Research on Leakage Location of Spacecraft in orbit Based on Frequency Weighting Matrix Beam forming Algorithm by Lamb Waves // Applied sciences, 2020, no. 10 (4), pp 1201. URL: https://doi.org/10.3390/app10041201

17. Ciampa F., Michele Meo. A new algorithm for acoustic emission localization and flexural group velocity determination in anisotropic structures // Journal Composites Part A Applied Science and Manufacturing, 2010, no. 41, pp. 1777 - 1786.

18. Tobias A. Acoustic-emission source location in two dimensions by an array of three sensors // Non-destructive testing, 1976, vol. 9, no. 1, pp. 9 - 12.

19. Ebrahimkhanlou A. A probabilistic framework for single-sensor acoustic emission source localization in thin metallic plates // Smart Materials and Structures, 2017, no. 26, pp. 1 - 19. DOI:10.1088/1361-665X/aa78de

20. Ebrahimkhanlou A., Salamone S. Single-Sensor Acoustic Emission Source Localization in Plate-Like Structures Using Deep Learning // Aerospace, 2018, pp. 5 - 50. DOI:10.3390/aerospace5020050

Application of the direct propagation neural network for localization of the impact site of microparticles on the surface of the spacecraft

Voronov K.E.*, Grigoriev D.P., Telegin A.M.*****

Samara National Research University named after Academician S.P. Korolev,

34, Moskovskoye shosse, Samara, 443086, Russia

**e-mail: voronov.ke@ssau.ru*

***e-mail: dan-22225@yandex.ru*

****e-mail: talex85@mail.ru*

Abstract

The purpose of this article is to demonstrate an experimental method for determining the impact region of microparticles in the surface of a spacecraft, through a neural network, with information about time delays in the data set. The article briefly describes the main types of neural network architectures that are widely used in various tasks. The theory of operation of the architecture of the neural network of direct propagation with mathematical explanations is given. The fundamental operations of neural network training, such as the method of error back propagation, gradient descent of the loss function, the training coefficient and its optimization, are also considered. The task of detecting the impact site of microparticles on the surface of the spacecraft body is set. As input data for training and testing the neural network, we used the results of an experiment to measure time delays on an experimental model with four piezo sensors. The output data was the numbers of the areas of the plate that were subjected to a simulated impact with a steel ball. The neural network model itself was written in the python programming language, using the Keras library and TensorFlow. This article also provides a detailed method for constructing a

neural network model in python. The neural network obtained in the course of the study showed good results in terms of predicting the impact area of cosmic particles. The accuracy reached almost 90-100%. These results, as well as the advantages, disadvantages and prospects of the considered method, are given at the end of the article.

Keywords: space debris, spacecraft, feed forward neural network, keras, tensorflow, impact localization.

References

1. Barkova M.E. *Trudy MAI*, 2018, no. 103. URL: http://trudymai.ru/eng/published.php?ID=100712

2. Efimov E.N., Shevgunov T.Ya. *Trudy MAI*, 2015, no. 82. URL: http://trudymai.ru/eng/published.php?ID=58786

3. Voronov K.E., Grigor'ev D.P., Telegin A.M. *Aviakosmicheskoe priborostroenie*, 2021, no. 1, pp. 40 - 50. DOI: 10.25791/aviakosmos.1.2021.1200

4. Barskii A.B. *Neironnye seti: raspoznavanie, upravlenie, prinyatie reshenii* (Neural networks: recognition, management, decision-making), Moscow, Finansy i statistika, 2004, 176 p.

5. Rashid T. *Sozdaem neironnuyu set'* (Creating a neural network), Saint Petersburg, Al'fa-kniga, 2017, 272 p.

6. Anikeev M.V., Babenko L.K., Makarevich O.B. *Radioelektronika, informatika, upravlinnya*, 2001, no. 1, pp. 48 - 56.

7. Akinina N.V., Akinin M.V., Sokolova A.V., Nikiforov M.B., Taganov A.I. *Izvestiya TulGU. Tekhnicheskie nauki*, 2016, no. 9, pp. 3 - 12.

8. Zemlevskii A.D. *Vestnik nauki i obrazovaniya*, 2017, vol. 2, no. 6 (30), pp. 36 – 43.

9. Bredikhin A.I. *Vestnik Yugorskogo gosudarstvennogo universiteta*, 2019, no. 1 (52), pp. 41 - 54. DOI: 10.17816/byusu20190141-54

10. Nikolenko S.I. *Glubokoe obuchenie* (Deep Learning), Saint Petersburg, Piter, 2018, 480 p.

11. *Mean Square Error*. URL: https://en.wikipedia.org/wiki/Mean_squared_error

12. Shumskikh I.Yu., Piganov M.N. *Regional'naya nauchno-prakticheskaya konferentsiya, posvyashchennaya 50-letiyu pervogo poleta cheloveka v kosmos: tezisy dokladov*, Samara, SGAU im. S.P. Koroleva, 2011, pp. 205 - 207.

13. *A complex open-source machine learning platform*. URL: https://www.tensorflow.org/

14. *Keras: the python deep learning API*. URL: https://keras.io/

15. Diederik Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization*, Cornell University, 2017. URL: https://arxiv.org/abs/1412.6980

16. Lei Qi et al. Research on Leakage Location of Spacecraft in orbit Based on Frequency Weighting Matrix Beam forming Algorithm by Lamb Waves, *Applied sciences*, 2020, no. 10 (4), pp 1201. URL: https://doi.org/10.3390/app10041201

17. Ciampa F., Michele Meo. A new algorithm for acoustic emission localization and flexural group velocity determination in anisotropic structures, *Journal Composites Part A Applied Science and Manufacturing*, 2010, no. 41, pp. 1777 - 1786.

18. Tobias A. Acoustic-emission source location in two dimensions by an array of three sensors, *Non-destructive testing*, 1976, vol. 9, no. 1, pp. 9 - 12.

19. Ebrahimkhanlou A. A probabilistic framework for single-sensor acoustic emission source localization in thin metallic plates, *Smart Materials and Structures*, 2017, no. 26, pp. 1 - 19. DOI: 10.1088/1361-665X/aa78de

20. Ebrahimkhanlou A., Salamone S. Single-Sensor Acoustic Emission Source Localization in Plate-Like Structures Using Deep Learning, *Aerospace*, 2018, pp. 5 - 50. DOI: 10.3390/aerospace5020050