МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева (Самарский университет)»

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

ОТЧЕТ

к лабораторной работе №2 по дисциплине

«Нейронные сети глубокого обучения»

Студент Е. Г. Плешаков

Студент О.В. Ширяева

Преподаватель А. Н. Жданова

Самара 2021

СОДЕРЖАНИЕ

1 Постановка задачи

Цель лабораторной работы: реализовать нейронную сеть для прогнозирования временных рядов.

В общем виде задача прогнозирования временных рядов может быть сформулирована следующим образом. Пусть имеется некоторый источник, порождающий последовательность элементов x_1, x_2, . . . из некоторого множества A, называемого алфавитом. Алфавит может быть как конечным, так и бесконечным (т. е. представлять собой некоторый ограниченный непрерывный интервал). Пусть при этом на момент времени t мы имеем конечную порождённую источником последовательность x_1, x_2, . . . , x_t . Задача прогнозирования сводится к предсказанию элемента, следующего в момент времени $(t+1)$, т. е. элемента x_t+1. Когда алфавит A является дискретным и конечным, любой алгоритм прогнозирования может быть применён к данному случаю естественным образом, так как будет оперировать с конечным множеством алфавита A и с конечной выборкой x_1, x_2, . . . , x_t. Если алфавит A представляет собой непрерывный конечный интервал, то поступим следующим образом. Разобьём заданный интервал на фиксированное количество непересекающихся подмножеств (в общем случае подмножества могут быть произвольного неравного размера), сопоставим им целочисленные номера в соответствии с их порядком в исходном интервале. Количество возможных номеров будет совпадать с числом интервалов. При этом множество всех номеров будет представлять собой уже новый конечный дискретный алфавит A'. Далее преобразуем исходный временной ряд из терминов в алфавите A в ряд, записанный в терминах нового алфавита A'. Таким образом, получим некоторую конечную выборку (ряд) из конечного алфавита и будем работать с ним, как с конечным дискретным алфавитом. При этом после прогнозирования очередного значения такого ряда ему сопоставляются соответствующий его номеру непрерывный интервал или точка из него (например, центр интервала). Количество букв алфавита обозначим через N. Предполагается, что процесс, или источник

информации, является стационарным и эргодическим, т. е. неформально распределение вероятностей символов этого источника не изменяется со временем и не зависит от конкретной реализации процесса. Пусть источник порождает сообщение $x_1, \ldots, x_t-1, x_t, x_i \in A, i = 1, 2, \ldots, t$, и требуется произвести прогнозирование n следующих элементов (в простейшем случае — одного элемента). Ошибкой прогноза называется (апостериорная) величина отклонения прогноза от действительного состояния объекта (т. е. величина $|x_i - x_i^*|$, где x_i^* — прогнозное значение, x_i — реальное значение). Под ошибкой прогнозирования *n* элементов будем понимать среднюю ошибку прогноза каждого из *n* элементов в отдельности. Ошибка прогноза характеризует качество прогнозирования. Очевидно, если распределение вероятностей исходов процесса известно заранее, то задача прогнозирования следующих значений решается достаточно просто (строится прогнозная функция в соответствии с известной закономерностью либо прогнозируются значения исходя из удовлетворения плотности распределения вероятностей ряда, полученного после вставки прогнозных элементов). Однако в большинстве практических задач описанные априорные данные отсутствуют, да и не всегда заданное распределение явно существует. В настоящей лабораторной работе будет рассматриваться именно такой случай: прогнозирование заболеваемости Covid-19.

2 Архитектуры сети, метода обучения

Для решения поставленной задачи была построена рекуррентная нейронная сеть, а точнее, её разновидность LSTM (Long short-term memory). Архитектура LSTM является наиболее подходящей для моделирования временных связей в глубоких нейронных сетях. Она преодолевает проблему исчезающего градиента в рекуррентной нейронной сети для долгосрочного обучения зависимости в данных с использованием ячеек памяти и вентилей.

LSTM-сеть — это искусственная нейронная сеть, содержащая LSTM-модули вместо или в дополнение к другим сетевым модулям. LSTM-модуль — это рекуррентный модуль сети, способный запоминать значения как на короткие, так и на длинные промежутки времени. Ключом к данной возможности является то, что LSTM-модуль не использует функцию активации внутри своих рекуррентных компонентов. Таким образом, хранимое значение не размывается во времени, и градиент или штраф не исчезает при использовании метода обратного распространения ошибки во времени при обучении искусственной нейронной сети.

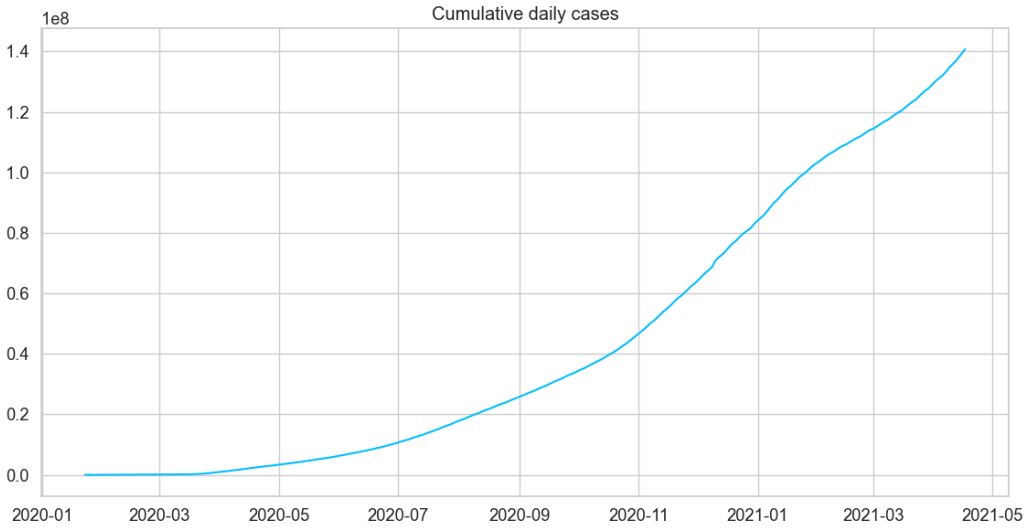
LSTM-модули часто группируются в «блоки», содержащие различные LSTM-модули. Подобное устройство характерно для «глубоких» многослойных нейронных сетей и способствует выполнению параллельных вычислений с применением соответствующего оборудования. В формулах ниже каждая переменная, записанная строчным курсивом, обозначает вектор размерности равной числу LSTM-модулей в блоке.

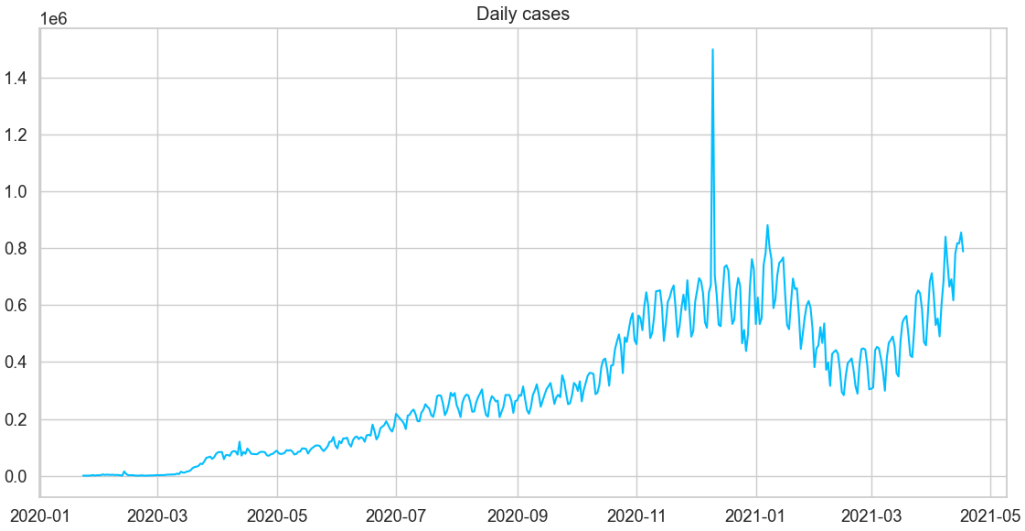
LSTM-блоки содержат три или четыре «вентиля», которые используются для контроля потоков информации на входах и на выходах памяти данных блоков. Эти вентили реализованы в виде логистической функции для вычисления значения в диапазоне [0; 1]. Умножение на это значение используется для частичного допуска или запрещения потока информации внутрь и наружу памяти. Например, «входной вентиль» контролирует меру вхождения нового значения в память, а «вентиль забывания» контролирует меру сохранения значения в памяти. «Выходной

вентиль» контролирует меру того, в какой степени значение, находящееся в памяти, используется при расчёте выходной функции активации для блока. (В некоторых реализациях входной вентиль и вентиль забывания воплощаются в виде единого вентиля. Идея заключается в том, что старое значение следует забывать тогда, когда появится новое значение, достойное запоминания).

Веса в LSTM-блоке (***W*** и ***U***) используются для задания направления оперирования вентилей. Эти веса определены для значений, которые подаются в блок (включая x_t и выход с предыдущего временного шага h_t-*1*) для каждого из вентилей. Таким образом, LSTM-блок определяет, как распоряжаться своей памятью как функцией этих значений, и тренировка весов позволяет LSTM-блоку выучить функцию, минимизирующую потери. LSTM-блоки обычно тренируют при помощи метода обратного распространения ошибки во времени.

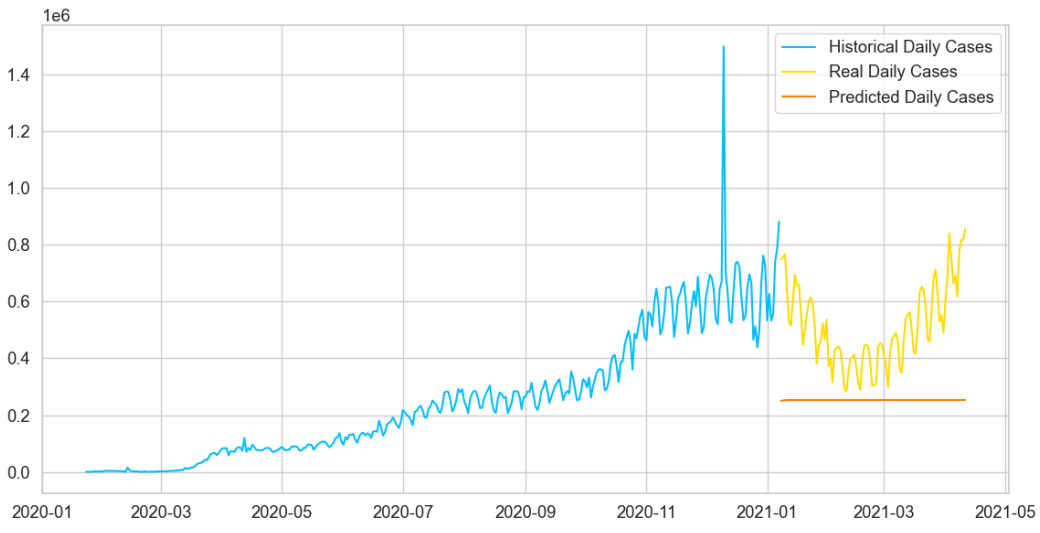
Для обучения модели используются оптимизатор Adam и функция потерь MSEloss. Adam — один из самых эффективных алгоритмов оптимизации в обучении нейронных сетей. Он сочетает в себе идеи RMSProp и оптимизатора импульса. Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), как в RMSProp, Adam также использует среднее значение вторых моментов градиентов. В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент, а параметры beta1 и beta2 управляют скоростью затухания этих скользящих средних. MSELoss – простая функция потери, которая вычисляет среднеквадратичную ошибку между вводом и целью.





3 Резул

Пря
мая
лин
ия в
про



гнозе на самом деле не является прямой, прогноз сети — не константа: [0.1667642742395401, 0.16777415573596954, 0.16824670135974884, 0.16850991547107697, 0.16865192353725433, …]

К сожалению, наших знаний не хватило на то, чтобы денормализовать полученные «спрямлённые» данные. Pytorch – удобный инструмент для создания LSTM-сетей, однако необходимо уделять большое внимание подготовке данных и их трактовке, чтобы не получить прогноз, которым невозможно воспользоваться.

Приложение А

Код программы с комментариями

Файл model.py

```
import torch
from torch import nn

# Класс создания модели, наследующий torch.nn.Module:
class CoronaVirusPredictor(nn.Module):
  # В конструкторе инициализируем поля и создаем слои модели
  def __init__(self, n_features, n_hidden, seq_len, n_layers=2):
    super(CoronaVirusPredictor, self).__init__()
    # количество блоков LSTM на слой
    self.n_hidden = n_hidden
    # количество временных шагов в каждом входном потоке
    self.seq_len = seq_len
    # количество скрытых слоев (всего получается n_hidden * n_layers LSTM блока.
    self.n_layers = n_layers
    self.lstm = nn.LSTM(
      # ожидаемое количество признаков во входном потоке
      input_size=n_features,
      # ожидаемое количество features в скрытом слое
      hidden_size=n_hidden,
      # количество рекуррентных слоев
      num_layers=n_layers,
      # прореживание выхода каждого рекуррентного слоя, кроме последнего
      dropout=0.5
    )
```

```
        self.linear = nn.Linear(in_features=n_hidden, out_features=1)

    # LSTM требует сброса состояния после каждого примера
    def reset_hidden_state(self):
        self.hidden = (
            torch.zeros(self.n_layers, self.seq_len, self.n_hidden),
            torch.zeros(self.n_layers, self.seq_len, self.n_hidden)
        )

    # Получаем все последовательности и пропускаем через слой LSTM вместе за раз.
    # Берём вывод последнего временного шага и пропускаем его через линейный слой для получения прогноза.
    def forward(self, sequences):
        lstm_out, self.hidden = self.lstm(
            sequences.view(len(sequences), self.seq_len, -1),
            self.hidden
        )
        last_time_step = \
            lstm_out.view(self.seq_len, len(sequences), self.n_hidden)[-1]
        y_pred = self.linear(last_time_step)
        return y_pred
```

Файл lab.py

```
# Tutorial: https://curiousily.com/posts/time-series-forecasting-with-lstm-for-daily-coronavirus-cases/
# Data: https://github.com/CSSEGISandData/COVID-19

from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import torch
from pandas.plotting import register_matplotlib_converters
from pylab import rcParams
from sklearn.preprocessing import MinMaxScaler

from model import CoronaVirusPredictor
```

```
sns.set(style='whitegrid', palette='muted', font_scale=1.2)
HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#93D30C", "#8F00FF"]
sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))

rcParams['figure.figsize'] = 14, 10
register_matplotlib_converters()

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)

# Данные о количестве зарегистрированных случаев covid-19 по странам в день. Датасет обновляется ежедневно.
url = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv'
df = pd.read_csv(url, index_col=0)
print(df.head(5))

# Удаляем столбцы province, country, latitude и longitude за ненадобностью:
df = df.iloc[:, 4:]
# Проверяем, есть ли пустые значения:
df.isnull().sum().sum()

daily_cases = df.sum(axis=0)
daily_cases.index = pd.to_datetime(daily_cases.index)
print(daily_cases.head())
plt.plot(daily_cases)
plt.title("Cumulative daily cases")
plt.show()

# Убираем накопление, вычитая текущее значение из предыдущего.
# Первое значение последовательности сохраняем.
daily_cases = daily_cases.diff().fillna(daily_cases[0]).astype(np.int64)
daily_cases.head()
plt.plot(daily_cases)
plt.title("Daily cases")
plt.show()

# Смотрим, за сколько дней у нас данные
```

```
print("Days in dataset:")
print(daily_cases.shape)


# ~3/4 рядов возьмём для обучения, ~1/4 для проверки
test_data_size = 100
train_data = daily_cases[:-test_data_size]
test_data = daily_cases[-test_data_size:]
print(train_data.shape)


# Нормализуем данные (приведём их к значениям между 0 и 1) для повышения точности и скорости обучения
# Для нормализации возьмем MinMaxScaler из scikit-learn:
scaler = MinMaxScaler()
scaler = scaler.fit(np.expand_dims(train_data, axis=1))
train_data = scaler.transform(np.expand_dims(train_data, axis=1))
test_data = scaler.transform(np.expand_dims(test_data, axis=1))
all_data = scaler.transform(np.expand_dims(daily_cases, axis=1))



# Разобьем данные на меньшие последовательности случаев за день:
def create_sequences(data, seq_length):
  xs = []
  ys = []
  for i in range(len(data) - seq_length - 1):
    x = data[i:(i + seq_length)]
    y = data[i + seq_length]
    xs.append(x)
    ys.append(y)
  return np.array(xs), np.array(ys)



seq_length = 5
X_train, Y_train = create_sequences(train_data, seq_length)
X_test, Y_test = create_sequences(test_data, seq_length)


X_all, Y_all = create_sequences(all_data, seq_length)
X_all = torch.from_numpy(X_all).float()
Y_all = torch.from_numpy(Y_all).float()
```

```
# Преобразование массивов NumPy в тензоры PyTorch
X_train = torch.from_numpy(X_train).float()
Y_train = torch.from_numpy(Y_train).float()
X_test = torch.from_numpy(X_test).float()
Y_test = torch.from_numpy(Y_test).float()


# Каждый пример данных, используемый для тренировки, содержит последовательность из
# 5 точек данных и метки с реальным значением, которое должна уметь предсказывать модель
print(X_train.shape)
print(X_train[:2])
print(Y_train.shape)
print(Y_train[:2])
print(train_data[:10])


print("X_train.shape")
print(X_train.shape)
print("Y_train.shape")
print(Y_train.shape)



# Функция тренировки модели
def train_model(
        model,
        train_data,
        train_labels,
        test_data=None,
        test_labels=None
):
    loss_fn = torch.nn.MSELoss(reduction='sum')
    optimiser = torch.optim.Adam(model.parameters(), lr=1e-3)
    num_epochs = 500
    train_hist = np.zeros(num_epochs)
    test_hist = np.zeros(num_epochs)
    for t in range(num_epochs):
        model.reset_hidden_state()
        y_pred = model(X_train)
        loss = loss_fn(y_pred.float(), Y_train)
        if test_data is not None:
```

```
            with torch.no_grad():
                y_test_pred = model(X_test)
                test_loss = loss_fn(y_test_pred.float(), Y_test)
            test_hist[t] = test_loss.item()
            print(f'Epoch {t} train loss: {loss.item()} test loss: {test_loss.item()}')
        else:
            print(f'Epoch {t} train loss: {loss.item()}')
        train_hist[t] = loss.item()
        optimiser.zero_grad()
        loss.backward()
        optimiser.step()
    return model.eval(), train_hist, test_hist


model_ready_to_use = Path("model.pt")
if model_ready_to_use.is_file():
    model = torch.load("model.pt")
    model.eval()
    print(model)
else:
    model = CoronaVirusPredictor(
        n_features=1,
        n_hidden=512,
        seq_len=seq_length,
        n_layers=3
    )
    model, train_hist, test_hist = train_model(
        model,
        X_train,
        Y_train,
        X_test,
        Y_test
    )
    torch.save(model, "model.pt")
    # Смотрим потери
    plt.plot(train_hist, label="Training loss")
    plt.plot(test_hist, label="Test loss")
    plt.ylim((0, 15))
    plt.legend()
    plt.show()


DAYS_TO_PREDICT = 5
```

```
with torch.no_grad():
    test_seq = X_test[:1]
    preds = []
    for _ in range(len(X_test)):
        y_test_pred = model(test_seq)
        pred = torch.flatten(y_test_pred).item()
        preds.append(pred)
        new_seq = test_seq.numpy().flatten()
        new_seq = np.append(new_seq, [pred])
        new_seq = new_seq[1:]
        test_seq = torch.as_tensor(new_seq).view(1, seq_length, 1).float()


true_cases = scaler.inverse_transform(
    np.expand_dims(Y_test.flatten().numpy(), axis=0)
).flatten()
predicted_cases = scaler.inverse_transform(
    np.expand_dims(preds, axis=0)
).flatten()


plt.plot(
    daily_cases.index[:len(train_data)],
    scaler.inverse_transform(train_data).flatten(),
    label='Historical Daily Cases'
)


plt.plot(
    daily_cases.index[len(train_data):len(train_data) + len(true_cases)],
    true_cases,
    label='Real Daily Cases'
)
plt.plot(
    daily_cases.index[len(train_data):len(train_data) + len(true_cases)],
    predicted_cases,
    label='Predicted Daily Cases'
)


plt.legend()
plt.show()
```

Приложение В
Вывод программы в консоль

```
C:\Users\Admin\AppData\Local\Programs\Python\Python38\interpreter\Scripts\python.exe
C:\Users\Admin\Desktop\magistr\AI\6138_time_series\lab.py
        Country/Region      Lat      Long  ...  4/19/21  4/20/21  4/21/21
Province/State                               ...
NaN           Afghanistan  33.93911  67.709953  ...    57898    58037    58214
NaN               Albania  41.15330  20.168300  ...   129694   129842   129980
NaN               Algeria  28.03390   1.659600  ...   119805   119992   120174
NaN               Andorra  42.50630   1.521800  ...    12805    12874    12917
NaN                Angola -11.20270  17.873900  ...    24518    24661    24883


[5 rows x 459 columns]
2020-01-23    655
2020-01-24    941
2020-01-25   1433
2020-01-26   2118
2020-01-27   2927
dtype: int64
Days in dataset:
(455,)
(355,)
torch.Size([349, 5, 1])
torch.Size([349, 1])
X_train.shape
torch.Size([349, 5, 1])
Y_train.shape
torch.Size([349, 1])
CoronaVirusPredictor(
  (lstm): LSTM(1, 512, num_layers=3, dropout=0.5)
  (linear): Linear(in_features=512, out_features=1, bias=True)
)
preds
[0.1667642742395401, 0.16777415573596954, 0.16824670135974884, 0.16850991547107697,
0.16865192353725433, 0.16873125731945038, 0.16877703368663788, 0.16880428791046143,
0.16882078349590302, 0.1688309758901596, 0.16883735358715057, 0.1688414067029953,
0.16884401440620422, 0.16884571313858032, 0.16884683072566986, 0.16884757578372955,
0.16884806752204895, 0.1688484102487564, 0.1688486486673355, 0.16884882748126984,
0.168848916888237, 0.16884900629520416, 0.16884908080101013, 0.16884909570217133,
0.1688491404056549, 0.1688491702079773, 0.1688491851091385, 0.16884920001029968,
0.16884920001029968, 0.16884921491146088, 0.16884922981262207, 0.16884921491146088,
0.16884921491146088, 0.16884921491146088, 0.16884921491146088, 0.16884921491146088,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884924471378326, 0.16884922981262207, 0.16884924471378326, 0.16884924471378326,
0.16884924471378326, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884924471378326,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884924471378326, 0.16884924471378326, 0.16884924471378326,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884924471378326, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884924471378326, 0.16884922981262207, 0.16884922981262207,
0.16884922981262207, 0.16884922981262207]
```