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 $x_1, x_2, \ldots, x_t$. 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 $x_1, x_2, \ldots, x_t$. 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

Лучше всего для решения поставленной задачи подходят рекуррентная нейронная сеть, а точнее, её разновидность LSTM (Long short-term memory), многослойный перцептрон (Dense-слои в Keras) и свёрточные сети (CNN). Свёрточные сети с плотными слоями были подробно рассмотрены нами в лабораторной работе №1 ”Распознавание изображений”, поэтому расскажем подробнее об LSTM. 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.

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 MSE (MeanSquaredError). 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. MSE – простая функция потери, которая вычисляет среднеквадратичную ошибку между вводом и целью. В качестве метрики для сравнения эффективности разных моделей выбрана MAE – средняя абсолютная ошибка.

3 Результат работы программы

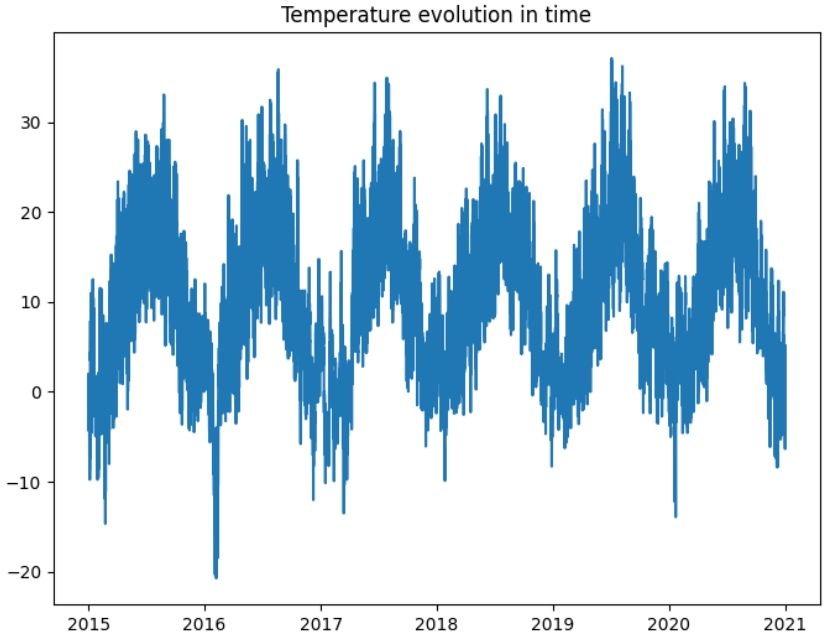

Рисунок 1 - Общий график изменений температуры (один признак) во времени

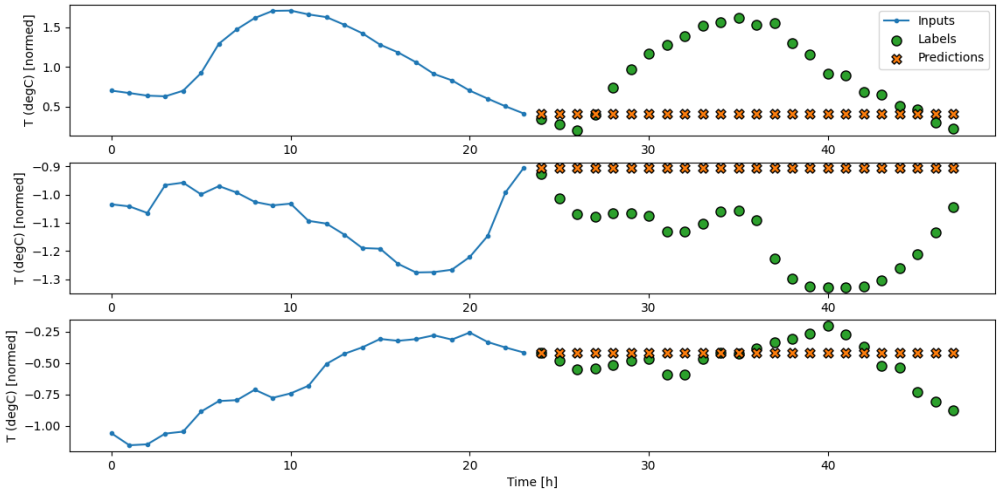

Рисунок 2 - Референсный “прогноз” без изменений для сравнения с прогнозами сетей

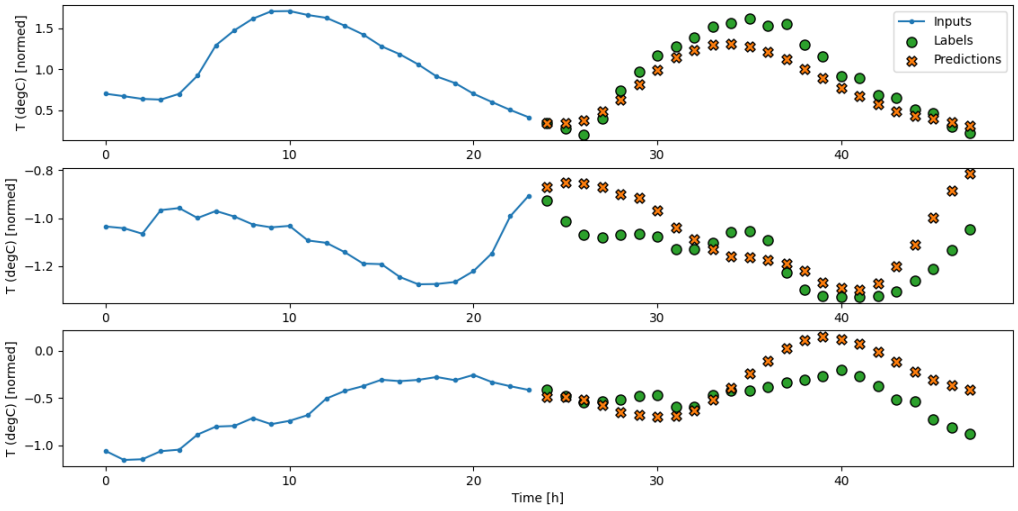

Рисунок 3 - Пример прогноза на 24 часа вперёд (LSTM)

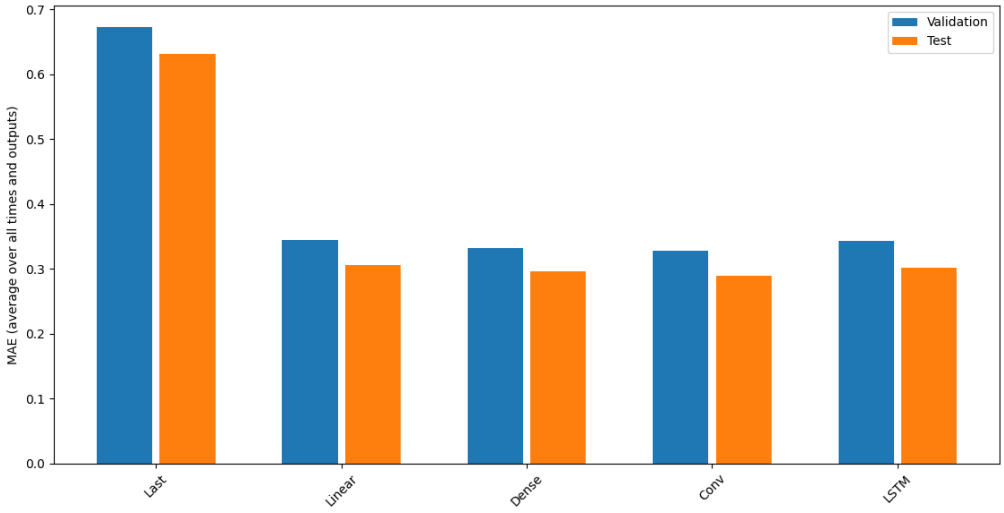

Рисунок 4 - Сравнение эффективности разных моделей (метрика - средняя абсолютная ошибка)

	20 эпох	200 эпох
Прогноз без изменений	0.6313	0.6313
Линейная функция	0.3066	0.3050
Многослойный перцептрон (Dense)	0.2965	0.2919
CNN (свёрточная)	0.2898	0.2885
RNN-LSTM (рекуррентная)	0.3017	0.2998

4 Выводы по работе

Как мы видим, увеличение количества эпох обучения с 20 до 200 не даёт ощутимой прибавки в точности построения прогноза, а потому не имеет смысла. Для задачи построения прогноза по одному признаку одинаково хорошо подходят как Dense-модели, так и CNN и LSTM. CNN опережает другие модели в эффективности распознавания образов на изображениях, а LSTM идеально подходит для обработки естественного языка и построения чат-ботов. Такая простая задача, как предсказание температуры, под силу любому типу нейросети.

Исходный код проекта доступен на GitHub: https://github.com/oshiryaeva/time_series_forecasting

Приложение А

Код программы с комментариями

Файл lab.py

```
# Tutorial:
https://www.tensorflow.org/tutorials/structured_data/time_series?hl=en
import datetime
import IPython
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False

url = "weather.csv"
df = pd.read_csv(url)
df = df[5::6]

print(df.head())

date_time = pd.to_datetime(df.pop('Time'), format='%d.%m.%Y %H:%M:%S')

# График по одному признаку
plot = df['T (degC)']
plot.index = date_time
plt.title('Temperature evolution in time')
plt.plot(plot)
plt.show()

# Статистика по датафрейму для выявления аномалий
print(df.describe())
```

```
# Аномальные значения заменяем нулями
wv = df['wv (m/s)']
bad_wv = wv == -9999.0
wv[bad_wv] = 0.0
max_wv = df['max. wv (m/s)']
bad_max_wv = max_wv == -9999.0
max_wv[bad_max_wv] = 0.0
df['wv (m/s)'].min()

# График скорости и направления ветра
plt.hist2d(df['wd (deg)'], df['wv (m/s)'], bins=(50, 50), vmax=400)
plt.colorbar()
plt.xlabel('Wind Direction [deg]')
plt.ylabel('Wind Velocity [m/s]')
plt.show()

# Превращаем направление и скорость ветра в вектор
wv = df.pop('wv (m/s)')
max_wv = df.pop('max. wv (m/s)')
wd_rad = df.pop('wd (deg)') * np.pi / 180
df['Wx'] = wv * np.cos(wd_rad)
df['Wy'] = wv * np.sin(wd_rad)
df['max Wx'] = max_wv * np.cos(wd_rad)
df['max Wy'] = max_wv * np.sin(wd_rad)

# Смотрим, что получилось в результате преобразования данных по ветру
plt.hist2d(df['Wx'], df['Wy'], bins=(50, 50), vmax=400)
plt.colorbar()
plt.xlabel('Wind X [m/s]')
plt.ylabel('Wind Y [m/s]')
ax = plt.gca()
ax.axis('tight')
plt.show()
```

```
# Конвертируем дату и время в секунды
timestamp_s = date_time.map(datetime.datetime.timestamp)
day = 24 * 60 * 60
year = (365.2425) * day
df['Day sin'] = np.sin(timestamp_s * (2 * np.pi / day))
df['Day cos'] = np.cos(timestamp_s * (2 * np.pi / day))
df['Year sin'] = np.sin(timestamp_s * (2 * np.pi / year))
df['Year cos'] = np.cos(timestamp_s * (2 * np.pi / year))

plt.plot(np.array(df['Day sin'])[:25])
plt.plot(np.array(df['Day cos'])[:25])
plt.xlabel('Time [h]')
plt.title('Time of day signal')
plt.show()

# Разделяем данные на три порции: 70% для тренировки, 20% для валидации и 10%
для тестов.
# Данные предварительно не перемешиваются, чтобы сохранить возможность
разделения их на хронологические отрезки,
# а также чтобы результаты валидации и проверки были более реалистичными.
column_indices = {name: i for i, name in enumerate(df.columns)}
n = len(df)
train_df = df[0:int(n * 0.7)]
val_df = df[int(n * 0.7):int(n * 0.9)]
test_df = df[int(n * 0.9):]

num_features = df.shape[1]

# Нормализация данных
train_mean = train_df.mean()
train_std = train_df.std()

train_df = (train_df - train_mean) / train_std
val_df = (val_df - train_mean) / train_std
test_df = (test_df - train_mean) / train_std
```

```
df_std = (df - train_mean) / train_std
df_std = df_std.melt(var_name='Column', value_name='Normalized')
plt.figure(figsize=(12, 6))
ax = sns.violinplot(x='Column', y='Normalized', data=df_std)
_ = ax.set_xticklabels(df.keys(), rotation=90)
plt.show()


# Класс для нарезки порций ("окон") данных на вхождения для тренировки (пары
feature-label) и проверки
class WindowGenerator():
    def __init__(self, input_width, label_width, shift,
                 train_df=train_df, val_df=val_df, test_df=test_df,
                 label_columns=None):
        self.train_df = train_df
        self.val_df = val_df
        self.test_df = test_df

        self.label_columns = label_columns
        if label_columns is not None:
            self.label_columns_indices = {name: i for i, name in
                                          enumerate(label_columns)}
        self.column_indices = {name: i for i, name in
                               enumerate(train_df.columns)}

        self.input_width = input_width
        self.label_width = label_width
        self.shift = shift

        self.total_window_size = input_width + shift

        self.input_slice = slice(0, input_width)
        self.input_indices =
np.arange(self.total_window_size)[self.input_slice]
```

```
        self.label_start = self.total_window_size - self.label_width
        self.labels_slice = slice(self.label_start, None)
        self.label_indices =
np.arange(self.total_window_size)[self.labels_slice]

    def __repr__(self):
        return '\n'.join([
            f'Total window size: {self.total_window_size}',
            f'Input indices: {self.input_indices}',
            f'Label indices: {self.label_indices}',
            f'Label column name(s): {self.label_columns}'])

w1 = WindowGenerator(input_width=24, label_width=1, shift=24,
                     label_columns=['T (degC)'])
print(w1)

w2 = WindowGenerator(input_width=6, label_width=1, shift=1,
                     label_columns=['T (degC)'])
print(w2)

def split_window(self, features):
    inputs = features[:, self.input_slice, :]
    labels = features[:, self.labels_slice, :]
    if self.label_columns is not None:
        labels = tf.stack(
            [labels[:, :, self.column_indices[name]] for name in
self.label_columns],
            axis=-1)

    inputs.set_shape([None, self.input_width, None])
    labels.set_shape([None, self.label_width, None])

    return inputs, labels
```

```
WindowGenerator.split_window = split_window

example_window = tf.stack([np.array(train_df[:w2.total_window_size]),
                           np.array(train_df[100:100 +
w2.total_window_size]),
                           np.array(train_df[200:200 +
w2.total_window_size])])

example_inputs, example_labels = w2.split_window(example_window)

print('All shapes are: (batch, time, features)')
print(f'Window shape: {example_window.shape}')
print(f'Inputs shape: {example_inputs.shape}')
print(f'labels shape: {example_labels.shape}')

w2.example = example_inputs, example_labels

# Функция для визуализации результатов
def plot(self, model=None, plot_col='T (degC)', max_subplots=3):
    inputs, labels = self.example
    plt.figure(figsize=(12, 8))
    plot_col_index = self.column_indices[plot_col]
    max_n = min(max_subplots, len(inputs))
    for n in range(max_n):
        plt.subplot(max_n, 1, n + 1)
        plt.ylabel(f'{plot_col} [normed]')
        plt.plot(self.input_indices, inputs[n, :, plot_col_index],
                 label='Inputs', marker='.', zorder=-10)

        if self.label_columns:
            label_col_index = self.label_columns_indices.get(plot_col, None)
        else:
            label_col_index = plot_col_index
```

```
        if label_col_index is None:
            continue

        plt.scatter(self.label_indices, labels[n, :, label_col_index],
                    edgecolors='k', label='Labels', c='#2ca02c', s=64)
        if model is not None:
            predictions = model(inputs)
            plt.scatter(self.label_indices, predictions[n, :,
label_col_index],
                        marker='X', edgecolors='k', label='Predictions',
                        c='#ff7f0e', s=64)

        if n == 0:
            plt.legend()

    plt.xlabel('Time [h]')
    plt.show()

WindowGenerator.plot = plot

w2.plot()

# Функция для создания датасета
def make_dataset(self, data):
    data = np.array(data, dtype=np.float32)
    ds = tf.keras.preprocessing.timeseries_dataset_from_array(
        data=data,
        targets=None,
        sequence_length=self.total_window_size,
        sequence_stride=1,
        shuffle=True,
        batch_size=32, )
```

```
    ds = ds.map(self.split_window)

    return ds

WindowGenerator.make_dataset = make_dataset

@property
def train(self):
    return self.make_dataset(self.train_df)

@property
def val(self):
    return self.make_dataset(self.val_df)

@property
def test(self):
    return self.make_dataset(self.test_df)

@property
def example(self):
    result = getattr(self, '_example', None)
    if result is None:
        result = next(iter(self.train))
        self._example = result
    return result

WindowGenerator.train = train
WindowGenerator.val = val
WindowGenerator.test = test
WindowGenerator.example = example

print(w2.train.element_spec)

for example_inputs, example_labels in w2.train.take(1):
```

```
    print(f'Inputs shape (batch, time, features): {example_inputs.shape}')
    print(f'Labels shape (batch, time, features): {example_labels.shape}')

# Самая простая модель - прогнозирует один признак на один шаг вперед
single_step_window = WindowGenerator(
    input_width=1, label_width=1, shift=1,
    label_columns=['T (degC)'])
print("single_step_window")
print(single_step_window)

for example_inputs, example_labels in single_step_window.train.take(1):
    print(f'Inputs shape (batch, time, features): {example_inputs.shape}')
    print(f'Labels shape (batch, time, features): {example_labels.shape}')

# Референсный класс для сравнения производительности моделей. Предсказывает
ту же температуру, что накануне, без изменений
class Baseline(tf.keras.Model):
    def __init__(self, label_index=None):
        super().__init__()
        self.label_index = label_index

    def call(self, inputs):
        if self.label_index is None:
            return inputs
        result = inputs[:, :, self.label_index]
        return result[:, :, tf.newaxis]

baseline = Baseline(label_index=column_indices['T (degC)'])

baseline.compile(loss=tf.losses.MeanSquaredError(),
                 metrics=[tf.metrics.MeanAbsoluteError()])

val_performance = {}
performance = {}
```

```
val_performance['Baseline'] = baseline.evaluate(single_step_window.val)
performance['Baseline'] = baseline.evaluate(single_step_window.test,
verbose=0)

# Модель без обучения, но с бОльшим окном
print('# Модель без обучения, но с бОльшим окном')
wide_window = WindowGenerator(
    input_width=24, label_width=24, shift=1,
    label_columns=['T (degC)'])

print(wide_window)

# Простейшая обучаемая модель: один слой линейной трансформации между входом
и выходом (без функции активации)
linear = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1)
])

print('Linear (tf.keras.Sequential)')
print('Input shape:', single_step_window.example[0].shape)
print('Output shape:', linear(single_step_window.example[0]).shape)

MAX_EPOCHS = 20
def compile_and_fit(model, window, patience=2):
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                      patience=patience,
                                                      mode='min')

    model.compile(loss=tf.losses.MeanSquaredError(),
                  optimizer=tf.optimizers.Adam(),
                  metrics=[tf.metrics.MeanAbsoluteError()])

    history = model.fit(window.train, epochs=MAX_EPOCHS,
                        validation_data=window.val,
```

```
                        callbacks=[early_stopping])
    return history

history = compile_and_fit(linear, single_step_window)

val_performance['Linear'] = linear.evaluate(single_step_window.val)
performance['Linear'] = linear.evaluate(single_step_window.test, verbose=0)

print('Input shape:', wide_window.example[0].shape)
print('Output shape:', baseline(wide_window.example[0]).shape)

wide_window.plot(linear)

# График весов
plt.bar(x=range(len(train_df.columns)),
        height=linear.layers[0].kernel[:, 0].numpy())
axis = plt.gca()
axis.set_xticks(range(len(train_df.columns)))
_ = axis.set_xticklabels(train_df.columns, rotation=90)
plt.show()

# Модель, похожая на линейную, но с добавлением нескольких плотных слоёв с
фукнцией активации relu
dense = tf.keras.Sequential([
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=1)
])

history = compile_and_fit(dense, single_step_window)

val_performance['Dense'] = dense.evaluate(single_step_window.val)
performance['Dense'] = dense.evaluate(single_step_window.test, verbose=0)
```

```
# Плотная модель с бОльшим количеством входов
CONV_WIDTH = 3
conv_window = WindowGenerator(
    input_width=CONV_WIDTH,
    label_width=1,
    shift=1,
    label_columns=['T (degC)'])
print("conv_window")
print(conv_window)

conv_window.plot()
plt.title("Given 3h as input, predict 1h into the future.")

multi_step_dense = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
    tf.keras.layers.Reshape([1, -1]),
])

print('Input shape:', conv_window.example[0].shape)
print('Output shape:', multi_step_dense(conv_window.example[0]).shape)

history = compile_and_fit(multi_step_dense, conv_window)

IPython.display.clear_output()
val_performance['Multi step dense'] =
multi_step_dense.evaluate(conv_window.val)
performance['Multi step dense'] = multi_step_dense.evaluate(conv_window.test,
verbose=0)

conv_window.plot(multi_step_dense)
```

```
print('Input shape:', wide_window.example[0].shape)
try:
    print('Output shape:', multi_step_dense(wide_window.example[0]).shape)
except Exception as e:
    print(f'\n{type(e).__name__}:{e}')

# Свёрточная модель
print('# Свёрточная модель')
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=32,
                           kernel_size=(CONV_WIDTH,),
                           activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
])

print("Conv model on `conv_window`")
print('Input shape:', conv_window.example[0].shape)
print('Output shape:', conv_model(conv_window.example[0]).shape)

history = compile_and_fit(conv_model, conv_window)

IPython.display.clear_output()
val_performance['Conv'] = conv_model.evaluate(conv_window.val)
performance['Conv'] = conv_model.evaluate(conv_window.test, verbose=0)

print("Wide window")
print('Input shape:', wide_window.example[0].shape)
print('Labels shape:', wide_window.example[1].shape)
print('Output shape:', conv_model(wide_window.example[0]).shape)

# Свёрточная модель с дополнительными входами
LABEL_WIDTH = 24
INPUT_WIDTH = LABEL_WIDTH + (CONV_WIDTH - 1)
wide_conv_window = WindowGenerator(
```

```
    input_width=INPUT_WIDTH,
    label_width=LABEL_WIDTH,
    shift=1,
    label_columns=['T (degC)'])
print("wide_conv_window")
print(wide_conv_window)

print("Wide conv window")
print('Input shape:', wide_conv_window.example[0].shape)
print('Labels shape:', wide_conv_window.example[1].shape)
print('Output shape:', conv_model(wide_conv_window.example[0]).shape)

wide_conv_window.plot(conv_model)

# Рекуррентная модель с долгосрочной кратковременной памятью (LSTM)
lstm_model = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, lstm_units]
    tf.keras.layers.LSTM(32, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])

print('Input shape:', wide_window.example[0].shape)
print('Output shape:', lstm_model(wide_window.example[0]).shape)

history = compile_and_fit(lstm_model, wide_window)

IPython.display.clear_output()
val_performance['LSTM'] = lstm_model.evaluate(wide_window.val)
performance['LSTM'] = lstm_model.evaluate(wide_window.test, verbose=0)

wide_window.plot(lstm_model)

# Сравнение эффективности моделей
```

```
print('# Сравнение эффективности моделей')
x = np.arange(len(performance))
width = 0.3
metric_name = 'mean_absolute_error'
metric_index = lstm_model.metrics_names.index('mean_absolute_error')
val_mae = [v[metric_index] for v in val_performance.values()]
test_mae = [v[metric_index] for v in performance.values()]

plt.ylabel('mean_absolute_error [T (degC), normalized]')
plt.bar(x - 0.17, val_mae, width, label='Validation')
plt.bar(x + 0.17, test_mae, width, label='Test')
plt.xticks(ticks=x, labels=performance.keys(),
           rotation=45)
_ = plt.legend()
plt.show()

for name, value in performance.items():
    print(f'{name:12s}: {value[1]:0.4f}')

# Множественный выход
print('# Множественный выход')
single_step_window = WindowGenerator(
    input_width=1, label_width=1, shift=1)

wide_window = WindowGenerator(
    input_width=24, label_width=24, shift=1)

for example_inputs, example_labels in wide_window.train.take(1):
    print(f'Inputs shape (batch, time, features): {example_inputs.shape}')
    print(f'Labels shape (batch, time, features): {example_labels.shape}')

baseline = Baseline()
baseline.compile(loss=tf.losses.MeanSquaredError(),
                 metrics=[tf.metrics.MeanAbsoluteError()])
```

```
val_performance = {}
performance = {}
val_performance['Baseline'] = baseline.evaluate(wide_window.val)
performance['Baseline'] = baseline.evaluate(wide_window.test, verbose=0)

print('tf.keras.Sequential')
dense = tf.keras.Sequential([
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=num_features)
])

history = compile_and_fit(dense, single_step_window)

IPython.display.clear_output()
val_performance['Dense'] = dense.evaluate(single_step_window.val)
performance['Dense'] = dense.evaluate(single_step_window.test, verbose=0)

wide_window = WindowGenerator(
    input_width=24, label_width=24, shift=1)

print('tf.keras.models.Sequential LSTM')
lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(32, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=num_features)
])

history = compile_and_fit(lstm_model, wide_window)

IPython.display.clear_output()
val_performance['LSTM'] = lstm_model.evaluate(wide_window.val)
performance['LSTM'] = lstm_model.evaluate(wide_window.test, verbose=0)
```

```
# Сравнение эффективности моделей с множественным выходом
x = np.arange(len(performance))
width = 0.3

metric_name = 'mean_absolute_error'
metric_index = lstm_model.metrics_names.index('mean_absolute_error')
val_mae = [v[metric_index] for v in val_performance.values()]
test_mae = [v[metric_index] for v in performance.values()]

plt.bar(x - 0.17, val_mae, width, label='Validation')
plt.bar(x + 0.17, test_mae, width, label='Test')
plt.xticks(ticks=x, labels=performance.keys(),
           rotation=45)
plt.ylabel('MAE (average over all outputs)')
_ = plt.legend()
plt.show()

for name, value in performance.items():
    print(f'{name:15s}: {value[1]:0.4f}')

# Прогноз на заданное количество временных шагов вперед
OUT_STEPS = 24
multi_window = WindowGenerator(input_width=24,
                               label_width=OUT_STEPS,
                               shift=OUT_STEPS)

multi_window.plot()

# Референсный класс с прогнозом без изменений
class MultiStepLastBaseline(tf.keras.Model):
    def call(self, inputs):
        return tf.tile(inputs[:, -1:, :], [1, OUT_STEPS, 1])
```

```
last_baseline = MultiStepLastBaseline()
last_baseline.compile(loss=tf.losses.MeanSquaredError(),
                      metrics=[tf.metrics.MeanAbsoluteError()])

multi_val_performance = {}
multi_performance = {}

multi_val_performance['Last'] = last_baseline.evaluate(multi_window.val)
multi_performance['Last'] = last_baseline.evaluate(multi_window.test,
verbose=0)
multi_window.plot(last_baseline)

class RepeatBaseline(tf.keras.Model):
    def call(self, inputs):
        return inputs

repeat_baseline = RepeatBaseline()
repeat_baseline.compile(loss=tf.losses.MeanSquaredError(),
                        metrics=[tf.metrics.MeanAbsoluteError()])

multi_val_performance['Repeat'] = repeat_baseline.evaluate(multi_window.val)
multi_performance['Repeat'] = repeat_baseline.evaluate(multi_window.test,
verbose=0)
multi_window.plot(repeat_baseline)

# Линейная
multi_linear_model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    tf.keras.layers.Dense(OUT_STEPS * num_features,
                          kernel_initializer=tf.initializers.zeros()),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

```
history = compile_and_fit(multi_linear_model, multi_window)

IPython.display.clear_output()
multi_val_performance['Linear'] =
multi_linear_model.evaluate(multi_window.val)
multi_performance['Linear'] = multi_linear_model.evaluate(multi_window.test,
verbose=0)
multi_window.plot(multi_linear_model)

# С плотными слоями
multi_dense_model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(OUT_STEPS * num_features,
                          kernel_initializer=tf.initializers.zeros()),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])

history = compile_and_fit(multi_dense_model, multi_window)

IPython.display.clear_output()
multi_val_performance['Dense'] = multi_dense_model.evaluate(multi_window.val)
multi_performance['Dense'] = multi_dense_model.evaluate(multi_window.test,
verbose=0)
multi_window.plot(multi_dense_model)

# CNN
CONV_WIDTH = 3
multi_conv_model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: x[:, -CONV_WIDTH:, :]),
    tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH)),
    tf.keras.layers.Dense(OUT_STEPS * num_features,
                          kernel_initializer=tf.initializers.zeros()),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

```
history = compile_and_fit(multi_conv_model, multi_window)

IPython.display.clear_output()

multi_val_performance['Conv'] = multi_conv_model.evaluate(multi_window.val)
multi_performance['Conv'] = multi_conv_model.evaluate(multi_window.test,
verbose=0)
multi_window.plot(multi_conv_model)

# RNN
print('RNN')
multi_lstm_model = tf.keras.Sequential([
    tf.keras.layers.LSTM(32, return_sequences=False),
    tf.keras.layers.Dense(OUT_STEPS * num_features,
                          kernel_initializer=tf.initializers.zeros()),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])

history = compile_and_fit(multi_lstm_model, multi_window)

IPython.display.clear_output()

multi_val_performance['LSTM'] = multi_lstm_model.evaluate(multi_window.val)
multi_performance['LSTM'] = multi_lstm_model.evaluate(multi_window.test,
verbose=0)
multi_window.plot(multi_lstm_model)

# Сравнение эффективности моделей с длинным прогнозом
x = np.arange(len(multi_performance))
width = 0.3

metric_name = 'mean_absolute_error'
metric_index = lstm_model.metrics_names.index('mean_absolute_error')
```

```
val_mae = [v[metric_index] for v in multi_val_performance.values()]
test_mae = [v[metric_index] for v in multi_performance.values()]

plt.bar(x - 0.17, val_mae, width, label='Validation')
plt.bar(x + 0.17, test_mae, width, label='Test')
plt.xticks(ticks=x, labels=multi_performance.keys(),
           rotation=45)
plt.ylabel(f'MAE (average over all times and outputs)')
_ = plt.legend()
plt.show()

for name, value in multi_performance.items():
    print(f'{name:8s}: {value[1]:0.4f}')
```

Приложение В
Вывод программы в консоль

```
Для 20 эпох
                    Time  T (degC)  rh (%)  wv (m/s)  max. wv (m/s)  wd (deg)
5   01.01.2015 00:50:00     -3.68    92.2      1.16           2.00     219.5
11  01.01.2015 01:50:00     -4.31    93.5      1.42           2.32     141.4
17  01.01.2015 02:50:00     -3.29    89.1      1.94           2.74     145.4
23  01.01.2015 03:50:00     -3.41    91.3      0.77           1.76     125.2
29  01.01.2015 04:50:00     -1.57    84.1      2.05           4.45     237.1
           T (degC)        rh (%)      wv (m/s)  max. wv (m/s)      wd (deg)
count  52548.000000  52548.000000  52548.000000   52548.000000  52548.000000
mean       9.879613     75.632117      1.555799       2.964220    175.288693
std        8.164463     16.551080     75.581220      75.612902     85.826482
min      -20.740000     13.880000  -9999.000000   -9999.000000      0.000000
25%        3.790000     64.630000      0.980000       1.760000    127.600000
50%        9.700000     78.800000      1.750000       2.960000    198.700000
75%       15.730000     89.000000      2.850000       4.760000    233.700000
max       37.100000    100.000000     12.810000      20.400000    360.000000
2021-04-30 00:17:38.812243: I tensorflow/compiler/jit/xla_cpu_device.cc:41]
Not creating XLA devices, tf_xla_enable_xla_devices not set
2021-04-30 00:17:38.815096: W
tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load
dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll not found
2021-04-30 00:17:38.815885: W
tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit:
UNKNOWN ERROR (303)
Total window size: 48
Input indices: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
20 21 22 23]
Label indices: [47]
Label column name(s): ['T (degC)']
Total window size: 7
Input indices: [0 1 2 3 4 5]
Label indices: [6]
Label column name(s): ['T (degC)']
All shapes are: (batch, time, features)
Window shape: (3, 7, 10)
Inputs shape: (3, 6, 10)
labels shape: (3, 1, 1)

(TensorSpec(shape=(None, 6, 10), dtype=tf.float32, name=None),
TensorSpec(shape=(None, 1, 1), dtype=tf.float32, name=None))
Inputs shape (batch, time, features): (32, 6, 10)
Labels shape (batch, time, features): (32, 1, 1)

# Сравнение эффективности моделей с единственным выходом
Baseline    : 0.0894
Linear      : 0.0700
Dense       : 0.0769
Multi step dense: 0.0612
Conv        : 0.0620
LSTM        : 0.0580
```

```
# Сравнение эффективности моделей с множественным выходом
Baseline       : 0.2347
Dense          : 0.1824
LSTM           : 0.1721

# Сравнение эффективности моделей с длинным прогнозом
Last    : 0.6313
Repeat  : 0.3846
Linear  : 0.3065
Dense   : 0.2887
Conv    : 0.2901
LSTM    : 0.3018

Process finished with exit code 0
```