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](#_heading=h.gjdgxs) Постановка задачи 3

[2](#_heading=h.30j0zll) Описание архитектуры сети, метода обучения 5

[3](#_heading=h.tyjcwt) Результат работы программы 7

4 Выводы по работе 9

[Приложение А Код программы](#_heading=h.3znysh7) 10

[Приложение B Вывод программы](#_heading=h.3znysh7) 31

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 *x1*, *x2*, . . . 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 *x1*, *x2*, *. . . , xt* . Задача прогнозирования сводится к предсказанию элемента, следующего в момент времени (*t+1*), т. е. элемента *xt+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 *x1*, *x2*, *. . . , xt*. 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 *x1, . . . , xt−1, xt , xi ∈ A, i = 1, 2, . . . , 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 |xi−xi∗|, где xi∗ — прогнозное значение, xi — 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). Cвёрточные 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](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82) или штраф не исчезает при использовании [метода обратного распространения ошибки во времени](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8_%D0%B2%D0%BE_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8&action=edit&redlink=1) 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](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) 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 *xt* и выход с предыдущего временного шага *ht-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 МАЕ – средняя абсолютная ошибка.

3 Результат работы программы

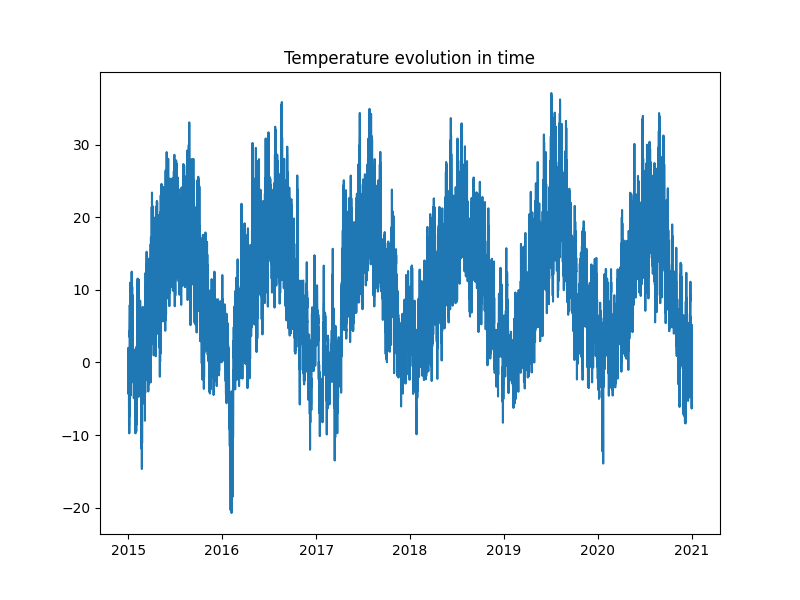


Рисунок 1 - Общий график изменений температуры (один признак) во времени

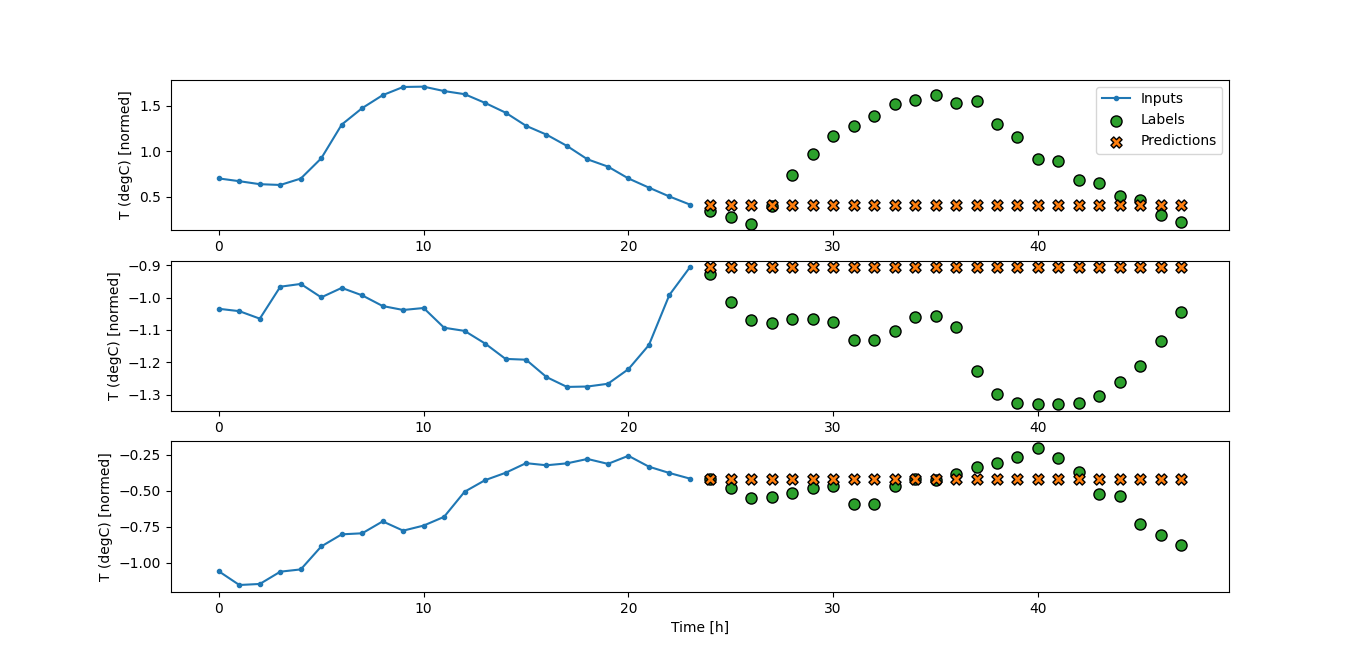


Рисунок 2 - Референсный “прогноз” без изменений для сравнения с прогнозами сетей

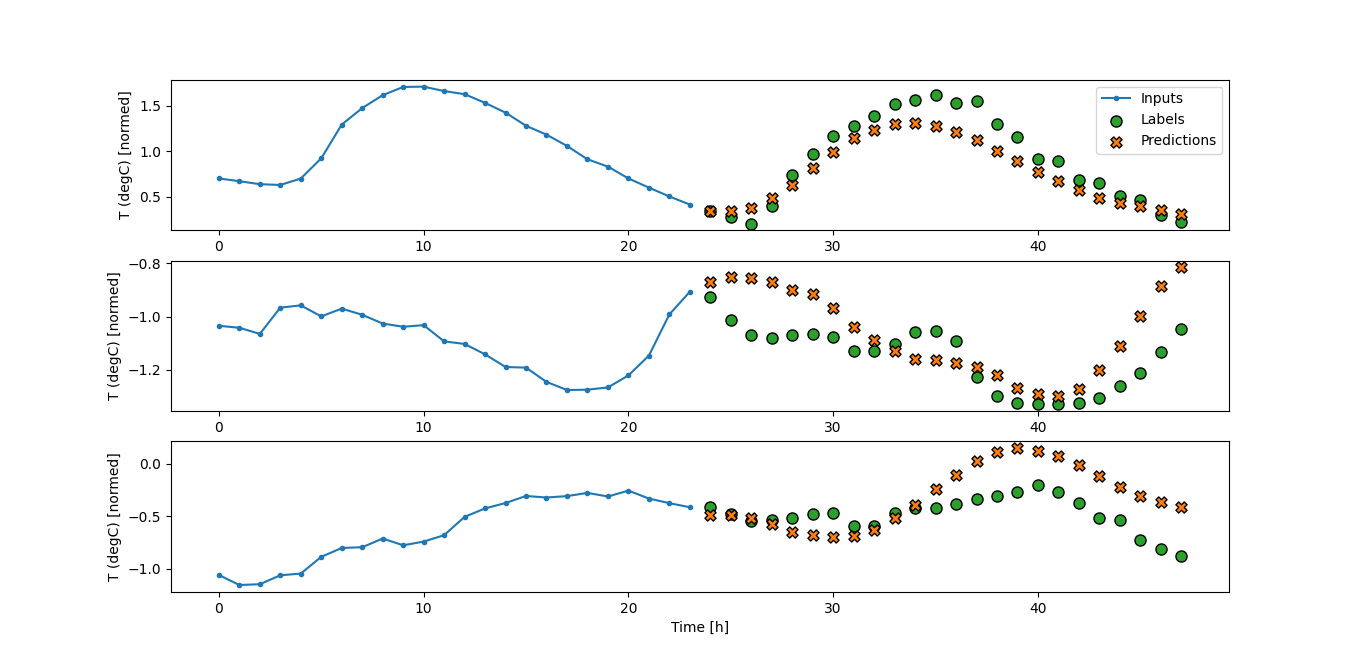


Рисунок 3 - Пример прогноза на 24 часа вперёд (LSTM)

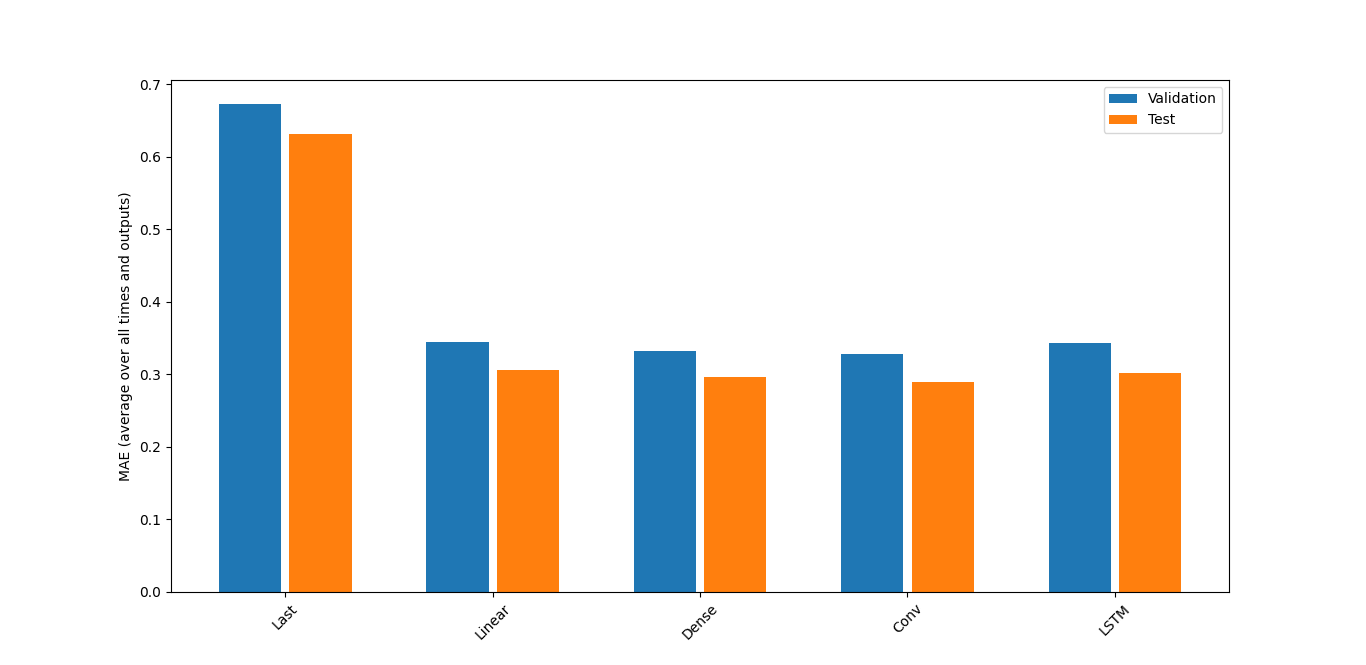


Рисунок 4 - Сравнение эффективности разных моделей (метрика - средняя абсолютная ошибка)

|  |  |  |
| --- | --- | --- |
|  | 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 B  
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