Generate_neural_network

10 августа 2023 г.

1 Создание нейронных сетей различных конфигураций

1.1 Подготовленные данные

Импорт исходных данных (по умолчанию - файл ***COVID_PSK.csv***)

[69]:
```
import numpy as np
import pandas as pd

from load_csv_silent import QuickLoad


X,l,ts,df1,df = QuickLoad()
```

```
e:\Users\Alex\source\repos\TSRevenko\load_csv_silent.py:37:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1['Inf_day'] = df1['Infections'].diff().fillna(df1['Infections'])
```

Для вывода в PDF

1. В терминале jupyter nbconvert –to latex имя файла блокнота,
2. В полученном tex файле внести изменения
3. Добавить
4. После (35 строка)
   [T2A]fontenc [english, russian]babel
5. Скомпилировать tex в pdf (PdfLatex)
6. Вариант: правленный tex перекинуть в Overleaf (и директорию с иллюстрациями)

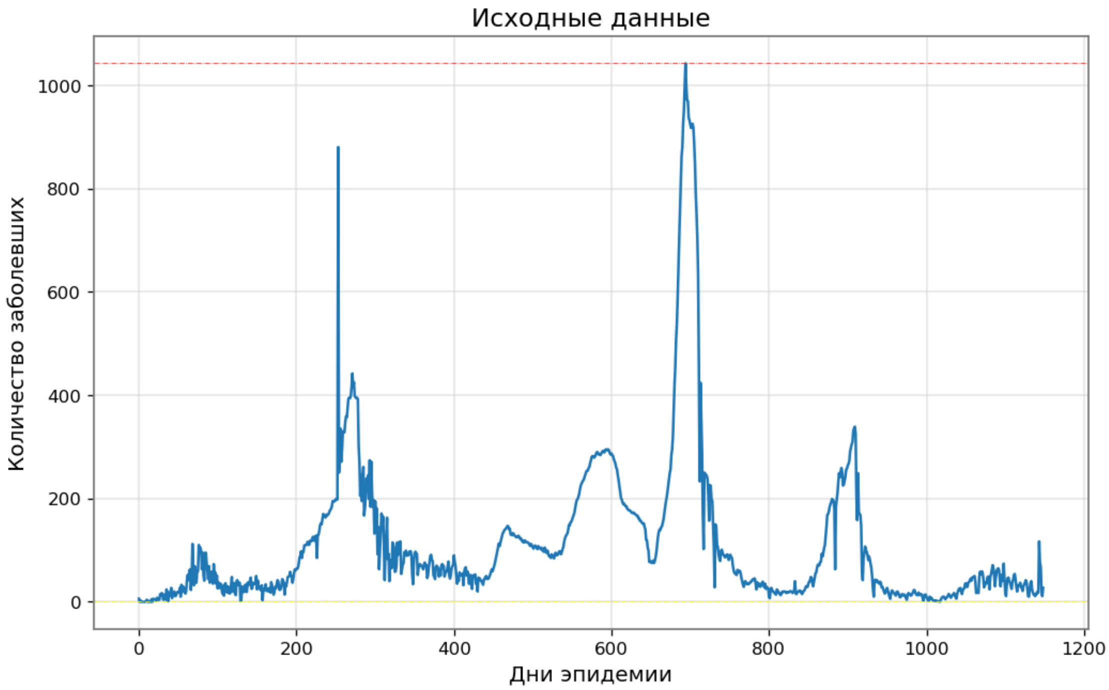
Секция со служебными функциями для вывода графиков

[70]:
```
import pandas as pd
import numpy as np
import scipy as sp

import matplotlib.pyplot as plt
```

```
import seaborn as sns

def set_style(dpi=120,fontsize=12,linewidth=1.
↪5,fontsize_xtick=10,fontsize_ytick=10):
    plt.rcParams['figure.dpi'] = dpi
    # set font size
    plt.rcParams['font.size'] = fontsize
    # set font size on x and y axis
    plt.rcParams['axes.labelsize'] = fontsize-2
    # set font size on thickmark
    plt.rcParams['xtick.labelsize'] = fontsize_xtick
    plt.rcParams['ytick.labelsize'] = fontsize_ytick
    # set grid color
    plt.rcParams['grid.color'] = 'lightgray'
    # set grid line width
    plt.rcParams['grid.linewidth'] = linewidth*0.75
    # set grid alpha
    plt.rcParams['grid.alpha'] = 0.5
    # set border color
    plt.rcParams['axes.edgecolor'] = 'gray'
    # set border width
    plt.rcParams['axes.linewidth'] = linewidth*0.75

def␣
↪lineplot(x,y,title=None,xlabel=None,ylabel=None,figsize=(10,6),dpi=120,fontsize=12,linewidth=
↪5,fontsize_xtick=10,fontsize_ytick=10):
    # set figure size  size
    plt.figure(figsize=figsize)
  ␣
↪set_style(dpi=dpi,fontsize=fontsize,linewidth=linewidth,fontsize_xtick=fontsize_xtick,fontsiz

    plt.plot(x,linewidth=linewidth)
    plt.title(title,fontsize=fontsize+2)
    plt.xlabel(xlabel,fontsize=fontsize)
    plt.ylabel(ylabel,fontsize=fontsize)
    plt.gridcolor='lightblue'
    plt.grid(True)

    # add max value of the line and plot horisontal dotted line at the max value
    plt.axhline(y=max(x),linewidth=linewidth*0.3,color='r',linestyle='-.')
    # add max value of the line and plot horisontal dotted line at the max value
    plt.axhline(y=min(x),linewidth=linewidth*0.3,color='yellow',linestyle='-.')
    plt.show()
def␣
↪lineplot2(x1,x2,y,title=None,xlabel=None,ylabel=None,figsize=(10,6),dpi=120,fontsize=12,linew
↪5,fontsize_xtick=10,fontsize_ytick=10):
    # set figure size  size
```

```
    plt.figure(figsize=figsize)
   ␣
↪set_style(dpi=dpi,fontsize=fontsize,linewidth=linewidth,fontsize_xtick=fontsize_xtick,fontsiz

    plt.plot(x1,linewidth=linewidth)
    plt.plot(x2,linewidth=linewidth)
    plt.title(title,fontsize=fontsize+2)
    plt.xlabel(xlabel,fontsize=fontsize)
    plt.ylabel(ylabel,fontsize=fontsize)
    plt.gridcolor='lightblue'
    plt.grid(True)

    plt.legend(['x1','x2'],fontsize=fontsize-2,loc='best')

    # add max value of the line and plot horisontal dotted line at the max value
    plt.axhline(y=max(max(x1),max(x2)),linewidth=linewidth*0.
↪3,color='r',linestyle='-.')
    # add max value of the line and plot horisontal dotted line at the max value
    plt.axhline(y=min(min(x1),min(x2)),linewidth=linewidth*0.
↪3,color='yellow',linestyle='-.')
    plt.show()


def lineplot_X_X2(x, x2,␣
↪y,title=None,xlabel=None,ylabel=None,figsize=(10,6),dpi=120,fontsize=12,linewidth=1.
↪5,fontsize_xtick=10,fontsize_ytick=10):
    # set figure size  size
    plt.figure(figsize=figsize)
   ␣
↪set_style(dpi=dpi,fontsize=fontsize,linewidth=linewidth,fontsize_xtick=fontsize_xtick,fontsiz

    plt.plot(np.append(x,x2),linewidth=linewidth)
    plt.plot(x,linewidth=linewidth)

    plt.title(title,fontsize=fontsize+2)
    plt.xlabel(xlabel,fontsize=fontsize)
    plt.ylabel(ylabel,fontsize=fontsize)
    plt.gridcolor='lightblue'
    plt.grid(True)

    # add max value of the line and plot horisontal dotted line at the max value
    plt.axhline(y=max(x),linewidth=linewidth*0.3,color='r',linestyle='-.')
    # add max value of the line and plot horisontal dotted line at the max value
    plt.axhline(y=max(x2),linewidth=linewidth*0.3,color='orange',linestyle='-.')
    plt.show()
```

[71]:
```
lineplot(X,l,figsize=(10,6),xlabel="Дни эпидемии",ylabel="Количество␣
↪заболевших", title = "Исходные данные")
```



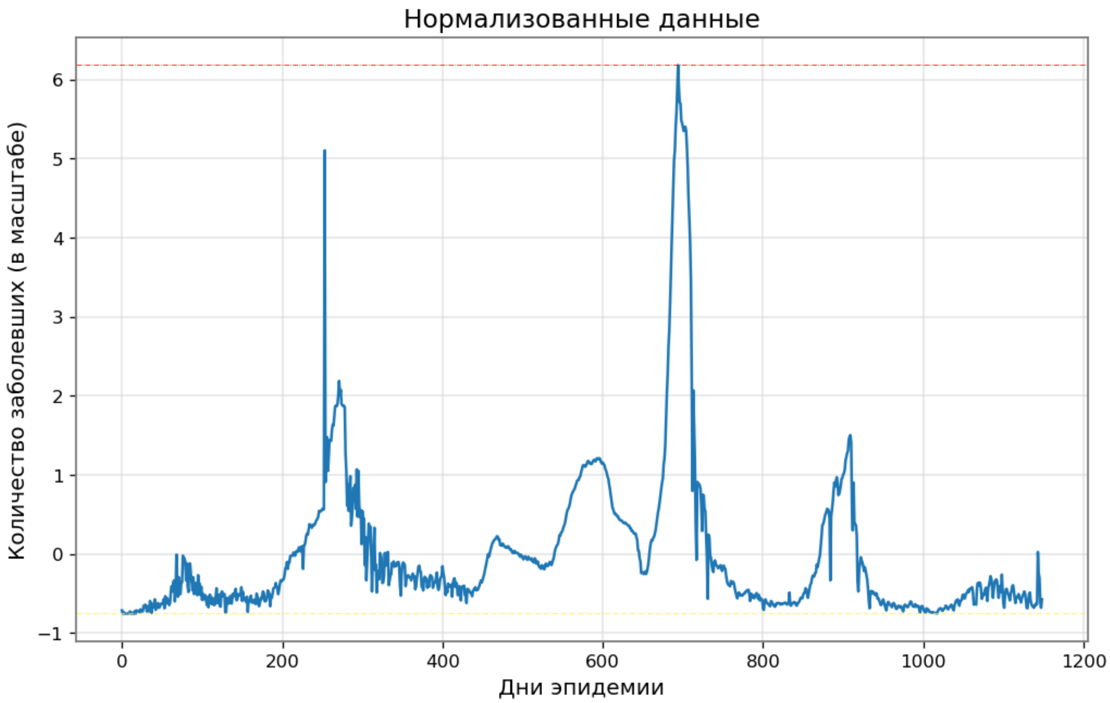
1.2 Препроцессинг данных, разделение на подмножества для обучения и проверки искусственной нейронной сети

Нормализация данных для обучения

[72]:
```
# Normalize the data
import numpy as np
from tslearn.preprocessing import TimeSeriesScalerMeanVariance
from sklearn.preprocessing import (
    MaxAbsScaler,
    MinMaxScaler,
    Normalizer,
    PowerTransformer,
    QuantileTransformer,
    RobustScaler,
    StandardScaler,
    minmax_scale,
)


scaler = TimeSeriesScalerMeanVariance()
# scaler = PowerTransformer(method='yeo-johnson')
```

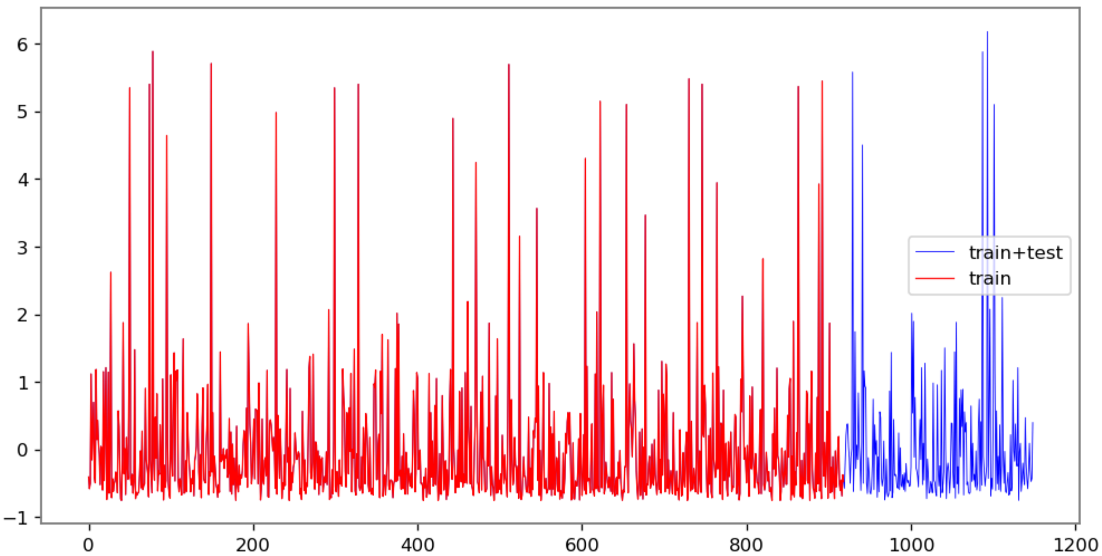
```
# scaler = StandardScaler()
# scaler = QuantileTransformer()
# xx = np.array(X).reshape(-1, 1)

X_normalized = scaler.fit_transform([X])

print(X_normalized)

X_n = X_normalized[0]

print(X_n)
lineplot(X_n,l,figsize=(10,6),xlabel="Дни эпидемии",ylabel="Количество␣
↪заболевших (в масштабе)", title = "Нормализованные данные")
```

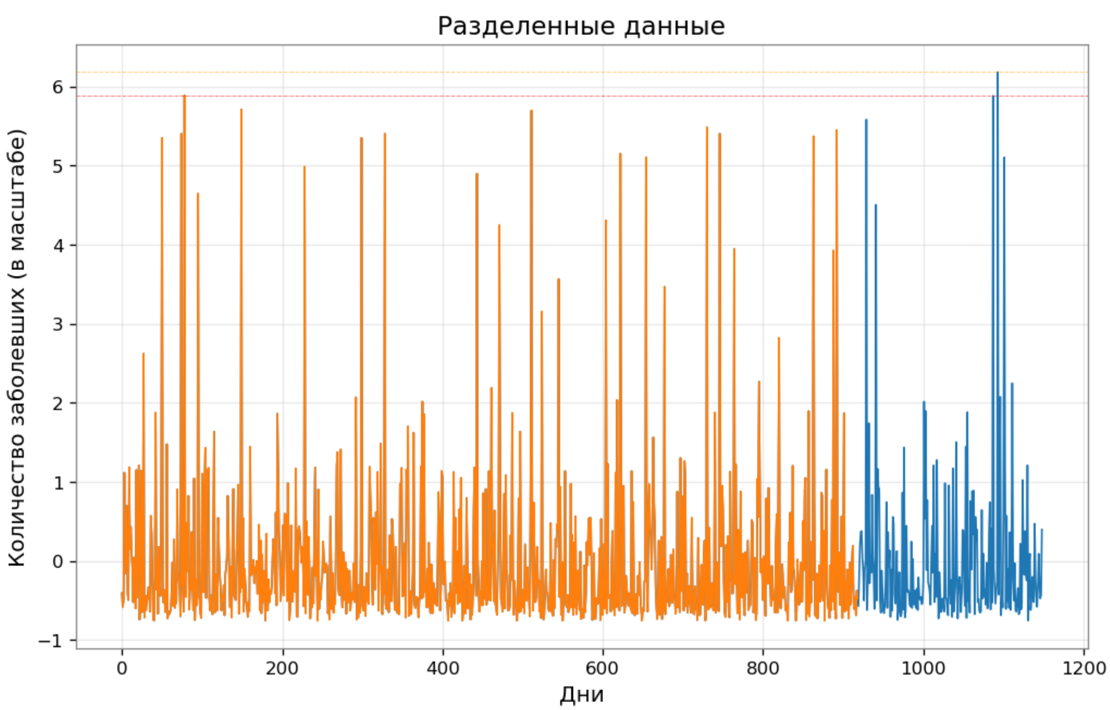
```
[[[-0.7141232 ]
  [-0.75404669]
  [-0.73408494]
  ...
  [-0.60100664]
  [-0.68085362]
  [-0.57439098]]]
[[-0.7141232 ]
 [-0.75404669]
 [-0.73408494]
 ...
 [-0.60100664]
 [-0.68085362]
 [-0.57439098]]
```



Разделяем данные

[73]:
```
from tslearn.utils import to_time_series_dataset
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_n, X_n, test_size=0.2,␣
 ↪random_state=42)
# set figure size
plt.figure(figsize=(10, 5))

plt.plot(np.append(X_train, X_test), 'b', label='train+test',linewidth=0.5)
plt.plot(X_train, 'r', label='train',linewidth=0.75)
plt.legend(loc='best', fontsize=10)
plt.show()

lineplot_X_X2(X_train, X_test,␣
 ↪y_train,figsize=(10,6),xlabel="Дни",ylabel="Количество заболевших (в␣
 ↪масштабе)", title = "Разделенные данные",linewidth = 1)
```





1.3 Создание и обучение нейронных сетей различной архитектуры и конфигурации

Какждая сеть строится на основании представленных выше данных. С помощью библиотеки tslearn, kerass (и лежащей в их основе tensorflow) собирается сеть из определенного количества слоев. Полученная модель уомпилируется. Модель обучается на представленных данных.

В диагностических целях рассчитывается ошибка масштабированных данных и ошибка в натуральном масштабе. Так как нормализатор данных tslearn не реализует метода обратного масштабирования (inverse transform), указанная операция производится в коде. Модель сохраняется в папку Models под именем, отражающим архитектуру модели и ее особенности для последующего использования.

Используются разновидности рекуррентных моделей: - GRU - LSTM

Используется количество слоев - 1 или 2

Используется оптимизатор - adam

Используется метрика (показатель качества обучения) - MSE

Для устранения переобучения используются:

- Регуляризация L1_L2 (ElasticNET)
- Слой Dropout

Таким образом, рассматриваются модели

LSTM / GRU | 1 слой / 2 слоя | Units1 | Units2 | без регуляризации / с регуляризацией | без Dropout / с Dropout |

Соответственно, имя модели формируется по шаблону

- Arch_{архитектура}
- Layers_{количество слоев}
- U1_{количество юнитов слоя 1}
- U2_{количество юнитов слоя 2}
- Regul1_{наличие регуляризации}
- Dropout1_{Наличие слоя Droput}
- Regul2_{наличие регуляризации}
- Dropout2_{Наличие слоя Droput}

[74]:
```
from keras.models import Sequential
from keras.layers import GRU, LSTM,Dropout,Dense, GaussianNoise
from keras.regularizers import l1_l2
from keras.callbacks import Callback
from keras.callbacks import EarlyStopping
```

Классы для упрощения генерации модели

Класс, описывающий свойства и архитектуру слоя. В дальнейшем на его основе формируется настроенный слой сети. Предполагается две архитектуры - GRU (быстрая) и LSTM - более медленная, но (теоретически) лучше адаптируемая. Обе архитектуры характеризуются количеством Units, обычно от 20 до 100 (в условиях задачи).

При необходимости можно включить регуляризацию (используется комплексная регуляризация ElasticNET, т.е. L1+L2).

Аналогично, можно добавить Dropout (случайное исключене определенного процента нейронов из обучения для исключения переобучения)

Параметры слоя, помимо прочего, формируют имя сети.

[75]:
```
class ArchEnum:
    GRU = 'GRU'
    LSTM = 'LSTM'
    pass

# class layer

class Layer:
    def __init__(self):
        self.Architecture = ArchEnum.GRU
        self.U = 50
        self.Regul = False
        self.L1  = 0.01
        self.L2  = 0.01
        self.Drop = False
        self.Drop_rate = 0.2

    def __str__(self):
        return f"Architecture: {self.Architecture}, U: {self.U}, Regul: {self.
↪Regul}, L1: {self.L1}, L2: {self.L2}, Drop: {self.Drop}, Drop_rate: {self.
↪Drop_rate}"
    def ToString(self):
        return f"Arch_{self.Architecture}_U_{self.U}_Reg_{self.Regul}_L1_{self.
↪L1}_L2_{self.L2}_Drop_{self.Drop}_rate_{self.Drop_rate}"
    def Constructor(self,architecture=ArchEnum.GRU,U=50,Regul=False,L1=0.01,L2=0.
↪01,Drop=False,drop_level=0.2):
        self.Architecture = architecture
        self.U = U
        self.Regul = Regul
        self.L1 = L1
        self.L2 = L2
        self.Drop = Drop
        self.Drop_rate = drop_level
        pass

    def SimpleGRU(self,U=50):
        self.Constructor(ArchEnum.GRU,U)
        pass
    def SimpleLSTM(self,U=50):
        self.Constructor(ArchEnum.LSTM,U)
        pass
    def GRU_With_Regul(self,U=50,L1=0.01,L2=0.01):
        self.Constructor(ArchEnum.GRU,U,Regul=True,L1=L1,L2=L2)
        pass
    def LSTM_With_Regul(self,U=50,L1=0.01,L2=0.01):
        self.Constructor(ArchEnum.LSTM,U,Regul=True,L1=L1,L2=L2)
        pass
```

```
    def GRU_With_Drop(self,U=50,Drop_rate=0.2):
        self.Constructor(ArchEnum.GRU,U,Drop=True,drop_level=Drop_rate)
        pass
    def LSTM_With_Drop(self,U=50,Drop_rate=0.2):
        self.Constructor(ArchEnum.LSTM,U,Drop=True,drop_level=Drop_rate)
        pass
    def GRU_With_Regul_And_Drop(self,U=50,L1=0.01,L2=0.01,drop_level=0.2):
        self.Constructor(ArchEnum.
↪GRU,U,Regul=True,L1=L1,L2=L2,Drop=True,drop_level=Drop_rate)
        pass
    def LSTM_With_Regul_And_Drop(self,U=50,L1=0.01,L2=0.01,Drop_rate=0.2):
        self.Constructor(ArchEnum.
↪LSTM,U,Regul=True,L1=L1,L2=L2,Drop=True,drop_level=Drop_rate)
        pass
```

Класс нейронной сети.

Необходимо использовать в правильном (показанном в комментариях) порядке, а именно.

1. Создать сеть
2. Предоставить ей данные (имеется вариант с заполнением данных по каждому элементу и метод PrepareData, который выполняет всю работу самостоятельно). В люом случае, подразумевается, что данные импортированы из файла csv (см. выше)
3. Наполнить модель слоями (Layer) с настроенными параметрами или одним слоем. Если выбрана регуляризация, то в модель будет добавлен слой с дополнительной настройкой, если Dropout, то дополнительный слой. Таким образом, каждый класс Layer порождает 1 или 2 слоя в сети.
4. Метод Build создает модель и компилирует ее.
5. После этого модель может быть обучена методом Fit.
6. Модель обучается (максимально) заданное в модели количество эпох и использует заданный batch. Используеися оптимизатор adam и метрика (критерий качества модели) MSE
7. При достижении погрешности 0,001 (Stop criteria) и если модель не улучшается на протяжении 15 (Pation) итераций, процесс может завершится раньше
8. Метод FitAndPlot выводит график кривой обучения

Резюмируя:

NeuralNetwork1 = NeuralNetwork()

Layer1 = Layer()

Layer1.GRU_With_Drop()

Layer2 = Layer()

Layer2.SimpleGRU()

NeuralNetwork1.Layers.append(Layer1)

NeuralNetwork1.Layers.append(Layer2)

NeuralNetwork1.PrepareData(X_unscaled=X)

NeuralNetwork1.Build()

NeuralNetwork1.FitAndPlot()

[76]:
```
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from keras.optimizers import Adadelta
from keras.optimizers import Nadam
from keras.optimizers import Adam
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from keras.optimizers import Adagrad
from keras.optimizers import Adadelta
from keras.optimizers import Adamax

class PlotLearningCurve(Callback):
    def on_train_begin(self, logs=None):
        self.losses = []
        self.val_losses = []
        self.fig, self.ax = plt.subplots()

    def on_epoch_end(self, epoch, logs=None):
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))

        self.ax.clear()
        self.ax.plot(self.losses, label='train_loss')
        self.ax.plot(self.val_losses, label='val_loss')
        self.ax.legend()
        self.ax.set_xlabel('Epoch')
        self.ax.set_ylabel('Loss')
        self.fig.canvas.draw()

class NeuralNetwork:
    # STEP1 : INITIALIZE
    def __init__(self):
        self.Name = "NN"
        self.Layers = []
        self.Model = None

        self.X = None
        self.X_normalized = None
        self.X_n = None
        self.X_train  = None
        self.X_test = None

        self.y_train = None
        self.y_test = None
```

```
        self.AddGaussianNoise = False
        self.GaussianNoise = 0.005

        self.EarlyStop = True

        self.Epochs = 200
        self.Batch_size = 32

        self.Stop_criteria = 0.001
        self.Patience = 15

        pass

    def __str__(self):
        return f"Name: {self.Name}, Layers: {self.Layers}  NLayers: {len(self.
↪Layers)}"
        pass
    def ToString(self):
        res = self.Name+'_'
        for layer in self.Layers:
            res += layer.ToString()
            res += "_"
        return res
    # STEP2 : ADD PREPARED DATA
    def␣
↪AddData(self,X_unscaled=None,X_normalized=None,X_n=None,X_n_train=None,X_n_test=None,␣
↪y_train = None, y_test = None):
        self.X = X_unscaled
        self.X_normalized = X_normalized
        self.X_n = X_n
        self.X_train = X_n_train
        self.X_test = X_n_test
        self.y_train = y_train
        self.y_test = y_test
        pass
    # STEP2 : ADD DATA, based on single column X from QuickLoad
    def PrepareData(self,X_unscaled):
        self.X = X_unscaled
        scaler = TimeSeriesScalerMeanVariance()
        self.X_normalized = scaler.fit_transform([X])
        self.X_n = self.X_normalized[0]
        # Split the data into training and testing sets
        self.X_train, self.X_test, self.y_train,self. y_test =␣
↪train_test_split(self.X_n, self.X_n, test_size=0.2, random_state=42)
        pass
    # STEP3 : ADD LAYERS
```

```
    def AddLayer(self,layer):
        self.Layers.append(layer)
        pass
    # STEP4 : BUILD MODEL WITH PREPARED DATA AND ADDED LAYERS
    def Build(self):
        self.Model = Sequential()
        if len(self.Layers) == 0:
            print("No layers")
            return
        if self.X.any() == None:
            print("No unscaled data")
            return
        if self.X_n.any() == None:
            print("No scaled data")
            return
        if self.X_train.any() == None:
            print("No train data")
            return
        if self.X_test.any() == None:
            print("No data")
            return

        if(self.AddGaussianNoise == True):
            self.Model.add(GaussianNoise(self.GaussianNoise,input_shape=(self.
↪X_normalized.shape[1], X_normalized.shape[2])))


        for i in range(0,len(self.Layers)):
            layer = self.Layers[i]
            if layer.Architecture == ArchEnum.GRU and layer.Regul == False:
                self.Model.add(
                        GRU(units=layer.U,
                        activation='relu',
                        return_sequences=True,
                        input_shape=(self.X_normalized.shape[1], X_normalized.
↪shape[2])))
            if layer.Architecture == ArchEnum.GRU and layer.Regul == True:
                self.Model.add(
                        GRU(units=layer.U,
                        activation='relu',
                        return_sequences=True,
                        input_shape=(self.X_normalized.shape[1], X_normalized.
↪shape[2]),
                        kernel_regularizer=l1_l2(l1=0.01, l2=0.01)))
            if layer.Architecture == ArchEnum.LSTM and layer.Regul == False:
                self.Model.add(
                        LSTM(units=layer.U,
```

```
                        activation='relu',
                        return_sequences=True,
                        input_shape=(self.X_normalized.shape[1], X_normalized.
↪shape[2])))
            if layer.Architecture == ArchEnum.GRU and layer.Regul == True:
                self.Model.add(
                        LSTM(units=layer.U,
                        activation='relu',
                        return_sequences=True,
                        input_shape=(self.X_normalized.shape[1], X_normalized.
↪shape[2]),
                        kernel_regularizer=l1_l2(l1=0.01, l2=0.01)))
            if layer.Drop == True:
                self.Model.add(Dropout(layer.Drop_rate))

        self.Model.add(Dense(1))  # Output layer for regression
        self.Model.compile(optimizer='adam', loss='mse')

    def SetOpt(self, name):
        if name == "adam":
            Opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.
↪0, amsgrad=False)
        if name == "sgd":
            Opt = SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
        if name == "rmsprop":
            Opt = RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
        if name == "adagrad":
            Opt = Adagrad(lr=0.01, epsilon=None, decay=0.0)
        if name == "adadelta":
            Opt = Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
        if name == "adamax":
            Opt = Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None,␣
↪decay=0.0)
        if name == "nadam":
            Opt = Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999,␣
↪epsilon=None, schedule_decay=0.004)

        self.Model.compile(optimizer=Opt, loss='mse')
        pass

    def Fit(self):
        # Define the EarlyStopping callback
        early_stopping = EarlyStopping(monitor='loss', patience=self.Patience,␣
↪mode='min', min_delta=self.Stop_criteria)
        # Train the model
        if self.EarlyStop == True:
```

```
            self.Model.fit(
                self.X_train, self.y_train, epochs=self.Epochs, batch_size=self.
↪Batch_size, validation_data=(self.X_test, self.y_test),
                callbacks=[early_stopping]
                )
        else:
            self.Model.fit(
                self.X_train, self.y_train, epochs=self.Epochs, batch_size=self.
↪Batch_size, validation_data=(self.X_test, self.y_test)
                )
        pass

    def FitAndPlot(self):
        # Define the EarlyStopping callback
        early_stopping = EarlyStopping(monitor='loss', patience=self.Patience,␣
↪mode='min', min_delta=self.Stop_criteria)
        # Define the custom callback
        plot_callback = PlotLearningCurve()
        # Train the model
        if self.EarlyStop == True:
            self.Model.fit(
            self.X_train, self.y_train, epochs=self.Epochs, batch_size=self.
↪Batch_size, validation_data=(self.X_test, self.y_test),
            callbacks=[plot_callback, early_stopping])
        else:
            self.Model.fit(
            self.X_train, self.y_train, epochs=self.Epochs, batch_size=self.
↪Batch_size, validation_data=(self.X_test, self.y_test),
            callbacks=[plot_callback])

    def BuidAndFit(self):
        self.Build()
        self.Fit()

    def BuidAndFitAndPlot(self):
        self.Build()
        self.FitAndPlot()

    def Predict(self, plot_data = True, plot_residuals = True):
        self.y_pred = None
        self.residuals = None

        self.y_pred =  (self.Model.predict(self.X_normalized))[0]
        self.residuals =(X_normalized[0] - y_pred)

        if plot_data == True:
```

```
            lineplot2(self.X_n,self.
↪y_pred,_,title="Сравнение",xlabel="Время",ylabel="Данные (трансформированные)")
        if plot_residuals == True:
            lineplot(self.residuals,_,␣
↪title="Остатки",xlabel="Время",ylabel="Остатки (трансформированные)")
        pass

    def Postprocess(self, plot_data = True, plot_residuals = True):
        mu = np.mean(self.X)
        var = np.var(self.X)
        sd = np.sqrt(var)
        print(mu,var,sd)

        self.X_hat = None
        self.Residuals = None

        self.X_hat = ((self.y_pred * sd) + mu).reshape(1,-1)

        self.df = None

        self.df = pd.DataFrame()
        self.df['X'] = self.X
        self.df['X_n'] = self.X_n.reshape(1,-1).tolist()[0]
        self.df['Y_n'] = self.y_pred
        self.df['X_hat'] = self.X_hat.reshape(1,-1).tolist()[0]
        self.df['Res_n'] = self.df['X_n'] - self.df['Y_n']
        self.df['Res_hat'] = self.df['X'] - self.df['X_hat']

        if plot_data == True:
            lineplot2(self.df['X'],self.
↪df['X_hat'],_,title="Сравнение",xlabel="Время",ylabel="Данные (натуральные␣
↪значения)")
        if plot_residuals == True:
            lineplot(self.df['Res_hat'],_,␣
↪title="Остатки",xlabel="Время",ylabel="Остатки (натуральные значения)")
        pass


        pass

    def SaveModel(self,path = None):
        if path == None:
            path = "./Mdl/"+self.ToString()+".h5"
            path2 = "./Mdl/"+self.ToString()+".csv"
        else:
            path = "./Mdl/" + self.ToString() + ".h5"
            path = "./Mdl/" + self.ToString() + ".csv"
```

```
        self.Model.save(path)
        self.df.to_csv(path2)
        pass
```

Типичные настройки

[77]:
```
NeuralNetwork1 = NeuralNetwork()
# Typical values

NeuralNetwork1.Epochs = 256
NeuralNetwork1.BatchSize = 64

NeuralNetwork1.EarlyStop = True
NeuralNetwork1.Stop_criteria = 0.0001
NeuralNetwork1.Patience = 25

NeuralNetwork1.AddGaussianNoise = True
NeuralNetwork1.GaussianNoise = 0.0001

Layer1 = Layer()
# Layer1.GRU_With_Drop(Drop_rate = 0.0125)
Layer1.SimpleGRU()
Layer1.U = 14

Layer2 = Layer()
Layer2.SimpleGRU()
Layer2.U = 60

NeuralNetwork1.Layers.append(Layer1)
#NeuralNetwork1.Layers.append(Layer2)
```

SetOpt - задает оптимизатор (теоретически более быстрый)

[78]:
```
# NeuralNetwork1.
 ↪AddData(X_unscaled=X,X_normalized=X_normalized,X_n=X_n,X_n_train=X_train,X_n_test=X_test,y_tr
NeuralNetwork1.PrepareData(X_unscaled=X)
NeuralNetwork1.Build()
NeuralNetwork1.SetOpt(name="adam")
```

```
e:\Alex\anaconda3\envs\med_2023\lib\site-
packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

Обучение модели и вывод результатов

[79]:
```
# NeuralNetwork1.Fit()
NeuralNetwork1.FitAndPlot()
```

```
Epoch 1/256
WARNING:tensorflow:Model was constructed with shape (None, 1149, 1) for input
KerasTensor(type_spec=TensorSpec(shape=(None, 1149, 1), dtype=tf.float32,
name='gaussian_noise_4_input'), name='gaussian_noise_4_input',
description="created by layer 'gaussian_noise_4_input'"), but it was called on
an input with incompatible shape (None, 1, 1).
WARNING:tensorflow:Model was constructed with shape (None, 1149, 1) for input
KerasTensor(type_spec=TensorSpec(shape=(None, 1149, 1), dtype=tf.float32,
name='gaussian_noise_4_input'), name='gaussian_noise_4_input',
description="created by layer 'gaussian_noise_4_input'"), but it was called on
an input with incompatible shape (None, 1, 1).
 1/29 [>...] - ETA: 39s - loss:
1.1104WARNING:tensorflow:Model was constructed with shape (None, 1149, 1) for
input KerasTensor(type_spec=TensorSpec(shape=(None, 1149, 1), dtype=tf.float32,
name='gaussian_noise_4_input'), name='gaussian_noise_4_input',
description="created by layer 'gaussian_noise_4_input'"), but it was called on
an input with incompatible shape (None, 1, 1).
29/29 [==============================] - 2s 18ms/step - loss: 0.7967 - val_loss:
0.7564
Epoch 2/256
29/29 [==============================] - 0s 6ms/step - loss: 0.6500 - val_loss:
0.6189
Epoch 3/256
29/29 [==============================] - 0s 6ms/step - loss: 0.5213 - val_loss:
0.4780
Epoch 4/256
29/29 [==============================] - 0s 8ms/step - loss: 0.3952 - val_loss:
0.3648
Epoch 5/256
29/29 [==============================] - 0s 9ms/step - loss: 0.2927 - val_loss:
0.2594
Epoch 6/256
29/29 [==============================] - 0s 8ms/step - loss: 0.2020 - val_loss:
0.1773
Epoch 7/256
29/29 [==============================] - 0s 7ms/step - loss: 0.1336 - val_loss:
0.1148
Epoch 8/256
29/29 [==============================] - 0s 7ms/step - loss: 0.0842 - val_loss:
0.0731
Epoch 9/256
29/29 [==============================] - 0s 8ms/step - loss: 0.0524 - val_loss:
0.0459
Epoch 10/256
29/29 [==============================] - 0s 7ms/step - loss: 0.0339 - val_loss:
0.0285
Epoch 11/256
29/29 [==============================] - 0s 7ms/step - loss: 0.0219 - val_loss:
```

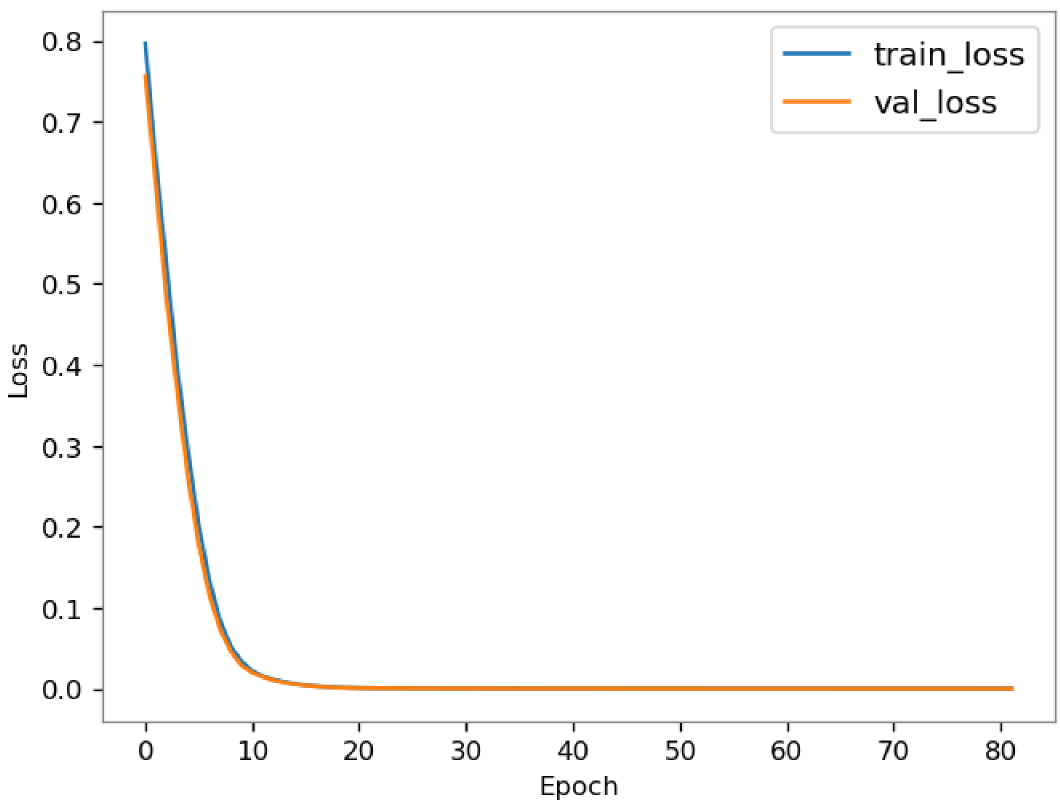
```
0.0198
Epoch 12/256
29/29 [==============================] - 0s 7ms/step - loss: 0.0154 - val_loss:
0.0144
Epoch 13/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0113 - val_loss:
0.0103
Epoch 14/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0084 - val_loss:
0.0075
Epoch 15/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0061 - val_loss:
0.0055
Epoch 16/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0044 - val_loss:
0.0040
Epoch 17/256
29/29 [==============================] - 0s 15ms/step - loss: 0.0032 - val_loss:
0.0029
Epoch 18/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0024 - val_loss:
0.0020
Epoch 19/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0017 - val_loss:
0.0015
Epoch 20/256
29/29 [==============================] - 0s 6ms/step - loss: 0.0013 - val_loss:
0.0011
Epoch 21/256
29/29 [==============================] - 0s 6ms/step - loss: 9.7131e-04 -
val_loss: 8.4599e-04
Epoch 22/256
29/29 [==============================] - 0s 5ms/step - loss: 7.7065e-04 -
val_loss: 6.5822e-04
Epoch 23/256
29/29 [==============================] - 0s 7ms/step - loss: 6.3265e-04 -
val_loss: 5.2309e-04
Epoch 24/256
29/29 [==============================] - 0s 5ms/step - loss: 5.3050e-04 -
val_loss: 4.4675e-04
Epoch 25/256
29/29 [==============================] - 0s 5ms/step - loss: 4.6682e-04 -
val_loss: 3.8886e-04
Epoch 26/256
29/29 [==============================] - 0s 6ms/step - loss: 4.2097e-04 -
val_loss: 3.4872e-04
Epoch 27/256
29/29 [==============================] - 0s 6ms/step - loss: 3.8732e-04 -
```

```
val_loss: 3.2381e-04
Epoch 28/256
29/29 [==============================] - 0s 5ms/step - loss: 3.6540e-04 -
val_loss: 3.0378e-04
Epoch 29/256
29/29 [==============================] - 0s 6ms/step - loss: 3.4465e-04 -
val_loss: 2.8594e-04
Epoch 30/256
29/29 [==============================] - 0s 6ms/step - loss: 3.3010e-04 -
val_loss: 2.7401e-04
Epoch 31/256
29/29 [==============================] - 0s 6ms/step - loss: 3.1569e-04 -
val_loss: 2.6400e-04
Epoch 32/256
29/29 [==============================] - 0s 6ms/step - loss: 3.0423e-04 -
val_loss: 2.5540e-04
Epoch 33/256
29/29 [==============================] - 0s 6ms/step - loss: 2.9240e-04 -
val_loss: 2.4692e-04
Epoch 34/256
29/29 [==============================] - 0s 6ms/step - loss: 2.8197e-04 -
val_loss: 2.3652e-04
Epoch 35/256
29/29 [==============================] - 0s 9ms/step - loss: 2.7248e-04 -
val_loss: 2.2908e-04
Epoch 36/256
29/29 [==============================] - 0s 6ms/step - loss: 2.6261e-04 -
val_loss: 2.2661e-04
Epoch 37/256
29/29 [==============================] - 0s 7ms/step - loss: 2.5352e-04 -
val_loss: 2.1604e-04
Epoch 38/256
29/29 [==============================] - 0s 7ms/step - loss: 2.4402e-04 -
val_loss: 2.1057e-04
Epoch 39/256
29/29 [==============================] - 0s 6ms/step - loss: 2.3583e-04 -
val_loss: 2.0190e-04
Epoch 40/256
29/29 [==============================] - 0s 6ms/step - loss: 2.2915e-04 -
val_loss: 1.9539e-04
Epoch 41/256
29/29 [==============================] - 0s 6ms/step - loss: 2.2019e-04 -
val_loss: 1.9382e-04
Epoch 42/256
29/29 [==============================] - 0s 6ms/step - loss: 2.1026e-04 -
val_loss: 1.8557e-04
Epoch 43/256
29/29 [==============================] - 0s 6ms/step - loss: 2.0143e-04 -
```

```
val_loss: 1.7724e-04
Epoch 44/256
29/29 [==============================] - 0s 6ms/step - loss: 1.9211e-04 -
val_loss: 1.7018e-04
Epoch 45/256
29/29 [==============================] - 0s 6ms/step - loss: 1.8439e-04 -
val_loss: 1.6323e-04
Epoch 46/256
29/29 [==============================] - 0s 7ms/step - loss: 1.7646e-04 -
val_loss: 1.6079e-04
Epoch 47/256
29/29 [==============================] - 0s 6ms/step - loss: 1.6967e-04 -
val_loss: 1.5141e-04
Epoch 48/256
29/29 [==============================] - 0s 7ms/step - loss: 1.6227e-04 -
val_loss: 1.4421e-04
Epoch 49/256
29/29 [==============================] - 0s 6ms/step - loss: 1.5448e-04 -
val_loss: 1.4013e-04
Epoch 50/256
29/29 [==============================] - 0s 6ms/step - loss: 1.4789e-04 -
val_loss: 1.3349e-04
Epoch 51/256
29/29 [==============================] - 0s 6ms/step - loss: 1.4173e-04 -
val_loss: 1.2957e-04
Epoch 52/256
29/29 [==============================] - 0s 6ms/step - loss: 1.3589e-04 -
val_loss: 1.2609e-04
Epoch 53/256
29/29 [==============================] - 0s 6ms/step - loss: 1.2943e-04 -
val_loss: 1.1886e-04
Epoch 54/256
29/29 [==============================] - 0s 6ms/step - loss: 1.2513e-04 -
val_loss: 1.1375e-04
Epoch 55/256
29/29 [==============================] - 0s 6ms/step - loss: 1.1967e-04 -
val_loss: 1.0928e-04
Epoch 56/256
29/29 [==============================] - 0s 6ms/step - loss: 1.1429e-04 -
val_loss: 1.0558e-04
Epoch 57/256
29/29 [==============================] - 0s 6ms/step - loss: 1.0937e-04 -
val_loss: 1.0139e-04
Epoch 58/256
29/29 [==============================] - 0s 6ms/step - loss: 1.0524e-04 -
val_loss: 9.7744e-05
Epoch 59/256
29/29 [==============================] - 0s 6ms/step - loss: 1.0107e-04 -
```

```
val_loss: 9.4475e-05
Epoch 60/256
29/29 [==============================] - 0s 6ms/step - loss: 9.7937e-05 -
val_loss: 9.1112e-05
Epoch 61/256
29/29 [==============================] - 0s 8ms/step - loss: 9.4512e-05 -
val_loss: 8.8371e-05
Epoch 62/256
29/29 [==============================] - 0s 6ms/step - loss: 9.1070e-05 -
val_loss: 8.6347e-05
Epoch 63/256
29/29 [==============================] - 0s 7ms/step - loss: 8.8754e-05 -
val_loss: 8.3124e-05
Epoch 64/256
29/29 [==============================] - 0s 6ms/step - loss: 8.4852e-05 -
val_loss: 8.5504e-05
Epoch 65/256
29/29 [==============================] - 0s 7ms/step - loss: 8.3223e-05 -
val_loss: 7.9437e-05
Epoch 66/256
29/29 [==============================] - 0s 7ms/step - loss: 7.9618e-05 -
val_loss: 7.7643e-05
Epoch 67/256
29/29 [==============================] - 0s 7ms/step - loss: 7.7863e-05 -
val_loss: 7.4620e-05
Epoch 68/256
29/29 [==============================] - 0s 6ms/step - loss: 7.5257e-05 -
val_loss: 7.3479e-05
Epoch 69/256
29/29 [==============================] - 0s 6ms/step - loss: 7.3854e-05 -
val_loss: 7.2895e-05
Epoch 70/256
29/29 [==============================] - 0s 6ms/step - loss: 7.1341e-05 -
val_loss: 6.9554e-05
Epoch 71/256
29/29 [==============================] - 0s 6ms/step - loss: 6.9369e-05 -
val_loss: 6.7721e-05
Epoch 72/256
29/29 [==============================] - 0s 6ms/step - loss: 6.7602e-05 -
val_loss: 6.5291e-05
Epoch 73/256
29/29 [==============================] - 0s 6ms/step - loss: 6.5361e-05 -
val_loss: 6.4367e-05
Epoch 74/256
29/29 [==============================] - 0s 6ms/step - loss: 6.4340e-05 -
val_loss: 6.2437e-05
Epoch 75/256
29/29 [==============================] - 0s 6ms/step - loss: 6.2604e-05 -
```

```
val_loss: 6.1071e-05
Epoch 76/256
29/29 [==============================] - 0s 6ms/step - loss: 6.1233e-05 -
val_loss: 6.0937e-05
Epoch 77/256
29/29 [==============================] - 0s 6ms/step - loss: 5.9723e-05 -
val_loss: 6.0905e-05
Epoch 78/256
29/29 [==============================] - 0s 6ms/step - loss: 5.8138e-05 -
val_loss: 5.6721e-05
Epoch 79/256
29/29 [==============================] - 0s 6ms/step - loss: 5.6876e-05 -
val_loss: 5.5021e-05
Epoch 80/256
29/29 [==============================] - 0s 7ms/step - loss: 5.5275e-05 -
val_loss: 5.3764e-05
Epoch 81/256
29/29 [==============================] - 0s 5ms/step - loss: 5.3699e-05 -
val_loss: 5.2461e-05
Epoch 82/256
29/29 [==============================] - 0s 6ms/step - loss: 5.2081e-05 -
val_loss: 5.0734e-05
```



[80]:

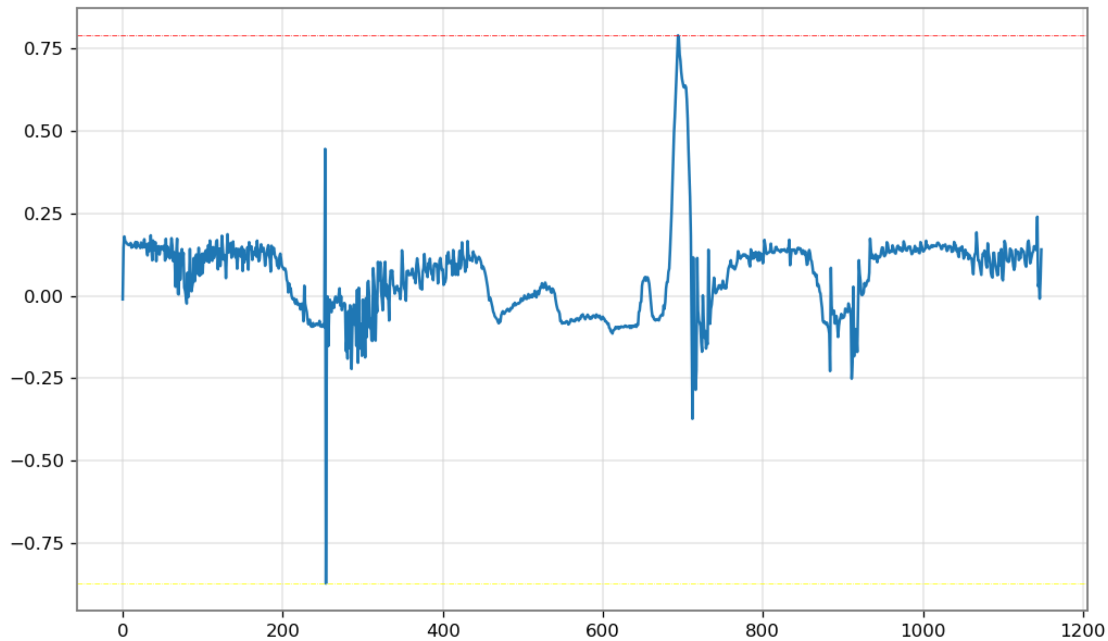
```
NeuralNetwork1.Model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gaussian_noise_4 (GaussianN  (None, 1149, 1)          0
 oise)

 gru_4 (GRU)                 (None, 1149, 14)          714

 dense_4 (Dense)             (None, 1149, 1)           15

=================================================================
Total params: 729
Trainable params: 729
Non-trainable params: 0
_________________________________________________________________
```

[81]:

```
y_pred =  (NeuralNetwork1.Model.predict(X_normalized))[0]
residuals =(X_normalized[0] - y_pred)

# Print the accuracy
# lineplot(X_normalized[0],y_pred)
lineplot(residuals,X_normalized[0])
```

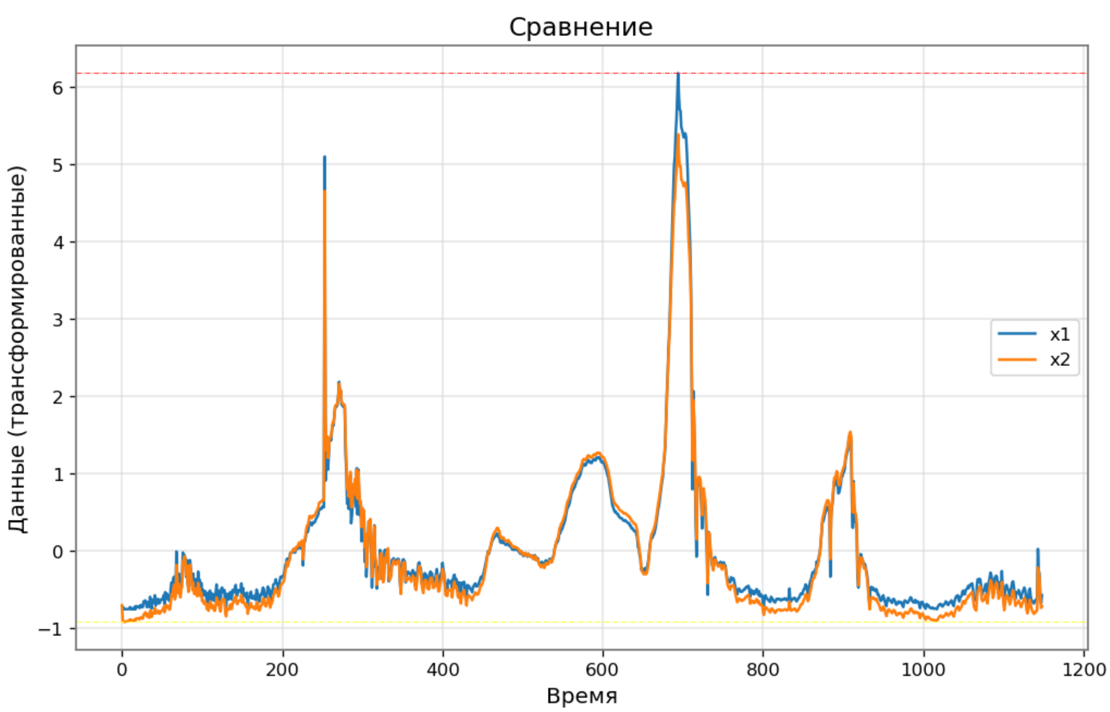
```
WARNING:tensorflow:5 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x00000205030E4EE0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 255ms/step
```

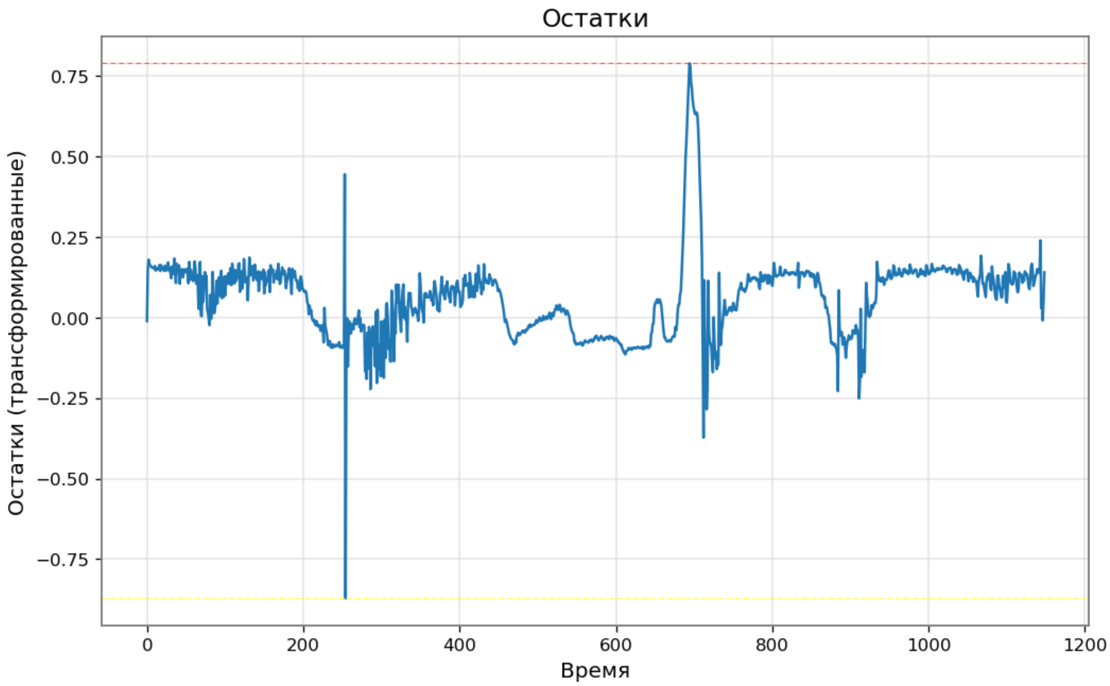


[82]:

```
NeuralNetwork1.Predict(plot_data = True, plot_residuals = True)
```

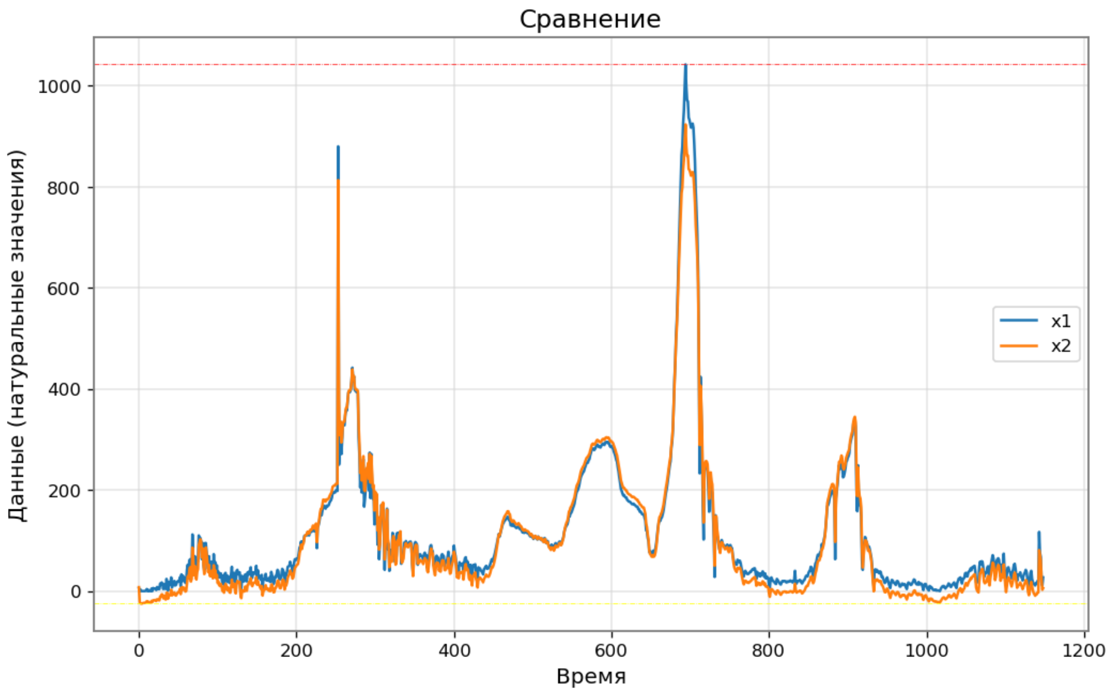
```
1/1 [==============================] - 0s 76ms/step
```

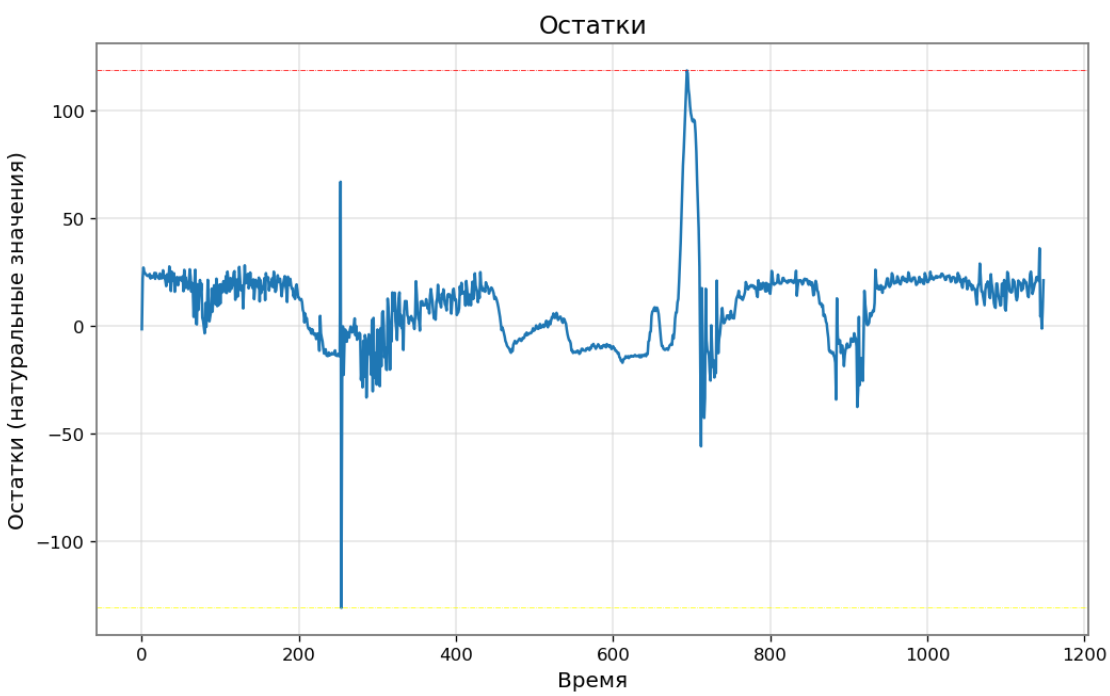




[83]:
```
import datetime
print(datetime.datetime.now())
NeuralNetwork1.Postprocess(plot_data = True, plot_residuals = True)
```

```
2023-08-10 21:47:48.498671
113.32375979112271 22586.319896742993 150.2874575496671
```


Сравнение
Данные (натуральные значения)
Время
x1
x2
0
200
400
600
800
1000
1200
−
0
200
400
600
800
1000


Остатки
Остатки (натуральные значения)
Время
0
200
400
600
800
1000
1200
−100
−50
0
50
100

[84]:

```
df = pd.DataFrame()
df['X'] = NeuralNetwork1.X
df['X_n'] = NeuralNetwork1.X_n.reshape(1,-1).tolist()[0]
df['Y_n'] = NeuralNetwork1.y_pred
df['X_hat'] = NeuralNetwork1.X_hat.reshape(1,-1).tolist()[0]
df['Res_n'] = df['X_n'] - df['Y_n']
df['Res_hat'] = df['X'] - df['X_n']
print(df)
```

```
         X       X_n       Y_n      X_hat     Res_n    Res_hat
0      6.0 -0.714123 -0.703530   7.592087 -0.010594   6.714123
1      0.0 -0.754047 -0.899378 -21.841431  0.145331   0.754047
2      3.0 -0.734085 -0.913948 -24.031143  0.179863   3.734085
3      0.0 -0.754047 -0.920399 -25.000610  0.166352   0.754047
4      0.0 -0.754047 -0.917029 -24.494263  0.162983   0.754047
...    ...       ...       ...        ...       ...        ...
1144  76.0 -0.248349 -0.277447  71.627007  0.029098  76.248349
1145  68.0 -0.301580 -0.340491  62.152283  0.038910  68.301580
1146  23.0 -0.601007 -0.592390  24.294960 -0.008617  23.601007
1147  11.0 -0.680854 -0.737311   2.515198  0.056457  11.680854
1148  27.0 -0.574391 -0.714775   5.901978  0.140384  27.574391

[1149 rows x 6 columns]
```

[85]:

```
NeuralNetwork1.SaveModel()
```