МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент |  |  |  | В. Ю. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
| прогнозировние значений процесса с помощью нейронных сетей |
| по курсу: интеллектуальный анализ данных на основе методов машинного обучения |
|  |
|  |

РАБОТУ ВЫПОЛНИЛА

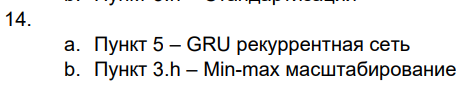
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4134К |  |  |  | Д. В. Самарин,  Р. Р. Усов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы**

Решение задачи прогностического анализ данных на основе моделей нейронных сетей

1. **Вариант: 14**





1. **Ход работы**
   1. **Генерация исходных данных**

dt = 0.05

t1 = -3

t2 = 3

k = 5

def func(t):

return np.cos(4 \* np.pi \* t)

def X\_Y(k, dt):

T = np.arange(t1, t2, dt)

F = func(T)

Fs = []

X = []

Y = []

Ts = []

for i in range(len(T)-k-1):

Xs = []

for j in range(k):

Xs.append(F[i+j])

X.append(Xs)

Y.append(F[i+k])

Ts.append(T[i+k])

Fs.append(F[i+k])

return Fs, Ts, np.array(X), np.array(Y)

Функция, создающая 4 массива:

- Ts - Массив значений параметра t, который соответвует значениям функции из массива Y

- X - Двумерный массив значений функции F, выбранной в порядке k идущих подряд, начиная с первого по максимально возможный индекс из массива F. Является данными, подающимися на вход модели

- Y - Массив значений функции F, начиная с k+1-й позиции, до последней. Должен являться результатом прогнозирования

- Fs - Массив значений функции F, который соответвует значениям функции из массива Ts

F, T, X, Y = X\_Y(k, dt)

Преобразование данных X и Y с помощью масштабирования

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

Разделение данных на обучающу, тестовую и валидационную выборку в отношении 60/20/20 %

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

* 1. **Обучение моделей нейронных сетей**

Создание 4-х моделей:

- Полносвязная модель для масштабированных данных

- Полносвязная модель для немасштабированных данных

- Модель на основе GRU блоков для масштабированных данных

- Модель на основе GRU блоков для немасштабированных данных

net\_dense = Sequential()

net\_dense.add(Input(shape=(X\_train.shape[1],)))

net\_dense.add(Dense(units=64))

net\_dense.add(Dense(units=32))

net\_dense.add(Dense(units=1))

net\_dense\_2 = Sequential()

net\_dense\_2.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_2.add(Dense(units=64))

net\_dense\_2.add(Dense(units=32))

net\_dense\_2.add(Dense(units=1))

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net\_2 = Sequential()

net\_2.add(Input(shape=(X\_train.shape[1], 1)))

net\_2.add(GRU(units=64, return\_sequences=True))

net\_2.add(GRU(units=32))

net\_2.add(Dense(units=1))

#Без масштабирования

net\_dense.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

print(f'Минимальная квадратичная ошибка на тестовых данных полносвязной модели без масштабирования: {mse}, R2: {r2}')

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

r2 = r2\_score(Y\_test, Y\_1)

print(f'Минимальная квадратичная ошибка на тестовых данных модели с GRU слоем без масштабирования: {mse}, R2: {r2}')

#С масштабированием

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scaled, Y\_scaled, test\_size=0.2, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense\_2.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_2.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_2.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_2.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

print(f'Минимальная квадратичная ошибка на тестовых данных полносвязной модели с масштабированием: {mse}, R2: {r2}')

net\_2.compile(optimizer='adam', loss='mean\_squared\_error')

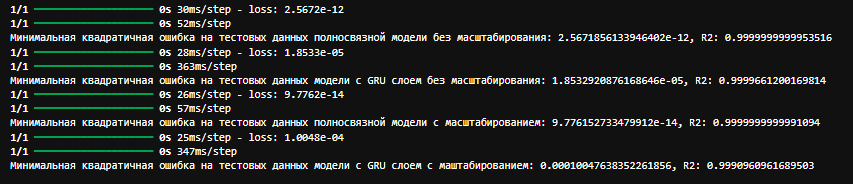
history = net\_2.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net\_2.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net\_2.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

r2 = r2\_score(Y\_test, Y\_1)

print(f'Минимальная квадратичная ошибка на тестовых данных модели с GRU слоем с маштабированием: {mse}, R2: {r2}')



В результате анализа точности четырёх моделей, лучшей оказалась полносвязная модель с масштабированием

Построим графики зависимостей ошибки mse и r2 от параметров k и dt для каждой модели

* 1. **Grid поиск параметров K и dt для каждой модели**

ks = []

mses = []

r2s = []

for k in range(3, 7):

F, T, X, Y = X\_Y(k, dt)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

r2 = r2\_score(Y\_test, Y\_1)

ks.append(k)

mses.append(mse)

r2s.append(r2)

print(f'mse: {mses} k: {ks} r2s: {r2s}')

fig, axs = plt.subplots(2, 2, figsize=(24, 13), dpi=80)

axs[0, 0].plot(ks, r2s, color='r', label='R2 от K для модели с GRU слоем с масштабированием')

axs[0, 0].set\_xlabel("K")

axs[0, 0].set\_ylabel("R2")

axs[0, 0].set\_title("R2 от K для модели с GRU слоем с масштабированием")

axs[0, 0].legend()

axs[0, 1].plot(ks, mses, color='b', label='MSE от K для модели с GRU слоем с масштабированием')

axs[0, 1].set\_xlabel("K")

axs[0, 1].set\_ylabel("MSE")

axs[0, 1].set\_title("MSE от K для модели с GRU слоем с масштабированием")

axs[0, 1].legend()

dts = []

mses = []

r2s = []

k = 5

dt=0.01

while dt <= 0.05:

F, T, X, Y = X\_Y(k, dt)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

r2 = r2\_score(Y\_test, Y\_1)

dts.append(dt)

mses.append(mse)

r2s.append(r2)

dt += 0.01

print(f'mse: {mses} dt: {dts} r2s: {r2s}')

axs[1, 0].plot(dts, r2s, color='r', label='R2 от dt для модели с GRU слоем с масштабированием')

axs[1, 0].set\_xlabel("dt")

axs[1, 0].set\_ylabel("R2")

axs[1, 0].set\_title("R2 от dt для модели с GRU слоем с масштабированием")

axs[1, 0].legend()

axs[1, 1].plot(dts, mses, color='b', label='MSE от dt для модели с GRU слоем с масштабированием')

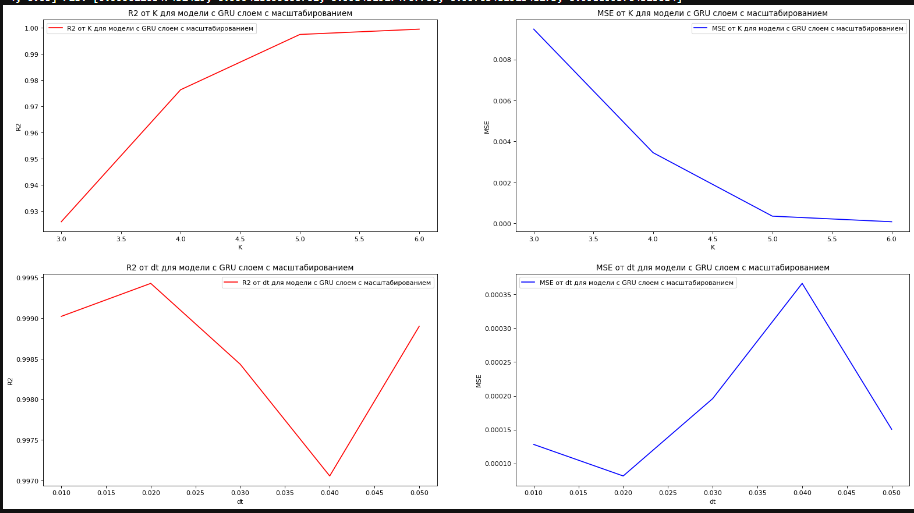
axs[1, 1].set\_xlabel("dt")

axs[1, 1].set\_ylabel("MSE")

axs[1, 1].set\_title("MSE от dt для модели с GRU слоем с масштабированием")

axs[1, 1].legend()

plt.show()



По графикам видно, что для полносвязной модели с GRU блоками и маштабированием лучшим значение k является 6, а лучшим значением dt 0.02

ks = []

mses = []

r2s = []

for k in range(3, 7):

F, T, X, Y = X\_Y(k, dt)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense\_test = Sequential()

net\_dense\_test.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_test.add(Dense(units=64))

net\_dense\_test.add(Dense(units=32))

net\_dense\_test.add(Dense(units=1))

net\_dense\_test.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_test.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_test.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_test.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

ks.append(k)

mses.append(mse)

r2s.append(r2)

print(f'mse: {mses} k: {ks} r2s: {r2s}')

fig, axs = plt.subplots(2, 2, figsize=(24, 13), dpi=80)

axs[0, 0].plot(ks, r2s, color='r', label='R2 от K для полносвязной модели с масштабированием')

axs[0, 0].set\_xlabel("K")

axs[0, 0].set\_ylabel("R2")

axs[0, 0].set\_title("R2 от K для полносвязной модели с масштабированием")

axs[0, 0].legend()

axs[0, 1].plot(ks, mses, color='b', label='MSE от K для полносвязной модели с масштабированием')

axs[0, 1].set\_xlabel("K")

axs[0, 1].set\_ylabel("MSE")

axs[0, 1].set\_title("MSE от K для полносвязной модели с масштабированием")

axs[0, 1].legend()

dts = []

mses = []

r2s = []

k = 5

dt=0.01

while dt <= 0.05:

F, T, X, Y = X\_Y(k, dt)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense\_test = Sequential()

net\_dense\_test.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_test.add(Dense(units=64))

net\_dense\_test.add(Dense(units=32))

net\_dense\_test.add(Dense(units=1))

net\_dense\_test.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_test.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_test.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_test.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

dts.append(dt)

mses.append(mse)

r2s.append(r2)

dt += 0.01

print(f'mse: {mses} dt: {dts} r2s: {r2s}')

axs[1, 0].plot(dts, r2s, color='r', label='R2 от dt для полносвязной модели с масштабированием')

axs[1, 0].set\_xlabel("dt")

axs[1, 0].set\_ylabel("R2")

axs[1, 0].set\_title("R2 от dt для полносвязной модели с масштабированием")

axs[1, 0].legend()

axs[1, 1].plot(dts, mses, color='b', label='MSE от dt для полносвязной модели с масштабированием')

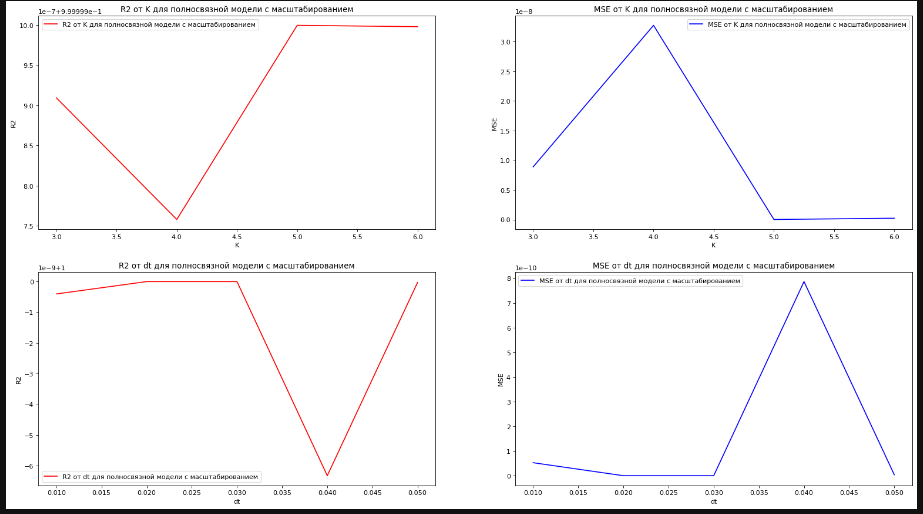
axs[1, 1].set\_xlabel("dt")

axs[1, 1].set\_ylabel("MSE")

axs[1, 1].set\_title("MSE от dt для полносвязной модели с масштабированием")

axs[1, 1].legend()

plt.show()



Для полносвязной модели с маштабированием лучшим значением k является 6, а dt 0.03

ks = []

mses = []

r2s = []

for k in range(3, 7):

F, T, X, Y = X\_Y(k, dt)

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense\_test = Sequential()

net\_dense\_test.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_test.add(Dense(units=64))

net\_dense\_test.add(Dense(units=32))

net\_dense\_test.add(Dense(units=1))

net\_dense\_test.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_test.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_test.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_test.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

ks.append(k)

mses.append(mse)

r2s.append(r2)

print(f'mse: {mses} k: {ks} r2s: {r2s}')

fig, axs = plt.subplots(2, 2, figsize=(24, 13), dpi=80)

axs[0, 0].plot(ks, r2s, color='r', label='R2 от K для полносвязной модели без масштабирования')

axs[0, 0].set\_xlabel("K")

axs[0, 0].set\_ylabel("R2")

axs[0, 0].set\_title("R2 от K для полносвязной модели без масштабирования")

axs[0, 0].legend()

axs[0, 1].plot(ks, mses, color='b', label='MSE от K для полносвязной модели без масштабирования')

axs[0, 1].set\_xlabel("K")

axs[0, 1].set\_ylabel("MSE")

axs[0, 1].set\_title("MSE от K для полносвязной модели без масштабирования")

axs[0, 1].legend()

dts = []

mses = []

r2s = []

k = 5

dt=0.01

while dt <= 0.05:

F, T, X, Y = X\_Y(k, dt)

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense\_test = Sequential()

net\_dense\_test.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_test.add(Dense(units=64))

net\_dense\_test.add(Dense(units=32))

net\_dense\_test.add(Dense(units=1))

net\_dense\_test.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_test.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_test.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_test.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

dts.append(dt)

mses.append(mse)

r2s.append(r2)

dt += 0.01

print(f'mse: {mses} dt: {dts} r2s: {r2s}')

axs[1, 0].plot(dts, r2s, color='r', label='R2 от dt для полносвязной модели без масштабирования')

axs[1, 0].set\_xlabel("dt")

axs[1, 0].set\_ylabel("R2")

axs[1, 0].set\_title("R2 от dt для полносвязной модели без масштабирования")

axs[1, 0].legend()

axs[1, 1].plot(dts, mses, color='b', label='MSE от dt для полносвязной модели без масштабирования')

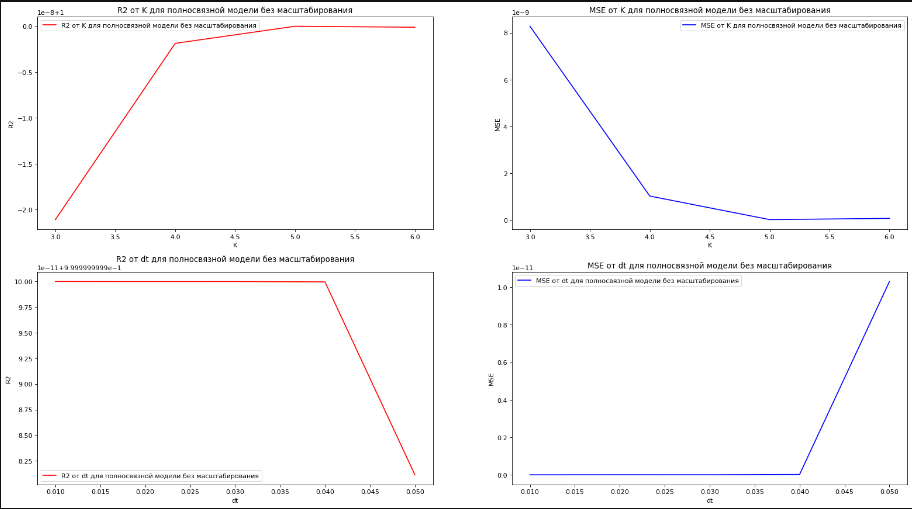
axs[1, 1].set\_xlabel("dt")

axs[1, 1].set\_ylabel("MSE")

axs[1, 1].set\_title("MSE от dt для полносвязной модели без масштабирования")

axs[1, 1].legend()

plt.show()



Для полносвязной модели без масштабирования лучшим значением k является 6, а dt 0.01

ks = []

mses = []

r2s = []

for k in range(3, 7):

F, T, X, Y = X\_Y(k, dt)

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

r2 = r2\_score(Y\_test, Y\_1)

ks.append(k)

mses.append(mse)

r2s.append(r2)

print(f'mse: {mses} k: {ks} r2s: {r2s}')

fig, axs = plt.subplots(2, 2, figsize=(24, 13), dpi=80)

axs[0, 0].plot(ks, r2s, color='r', label='R2 от K для модели с GRU слоем без масштабирования')

axs[0, 0].set\_xlabel("K")

axs[0, 0].set\_ylabel("R2")

axs[0, 0].set\_title("R2 от K для модели с GRU слоем без масштабирования")

axs[0, 0].legend()

axs[0, 1].plot(ks, mses, color='b', label='MSE от K для модели с GRU слоем без масштабирования')

axs[0, 1].set\_xlabel("K")

axs[0, 1].set\_ylabel("MSE")

axs[0, 1].set\_title("MSE от K для модели с GRU слоем без масштабирования")

axs[0, 1].legend()

dts = []

mses = []

r2s = []

k = 5

dt=0.01

while dt <= 0.05:

F, T, X, Y = X\_Y(k, dt)

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

r2 = r2\_score(Y\_test, Y\_1)

dts.append(dt)

mses.append(mse)

r2s.append(r2)

dt += 0.01

print(f'mse: {mses} dt: {dts} r2s: {r2s}')

axs[1, 0].plot(dts, r2s, color='r', label='R2 от dt для модели с GRU слоем без масштабирования')

axs[1, 0].set\_xlabel("dt")

axs[1, 0].set\_ylabel("R2")

axs[1, 0].set\_title("R2 от dt для модели с GRU слоем без масштабирования")

axs[1, 0].legend()

axs[1, 1].plot(dts, mses, color='b', label='MSE от dt для модели с GRU слоем без масштабирования')

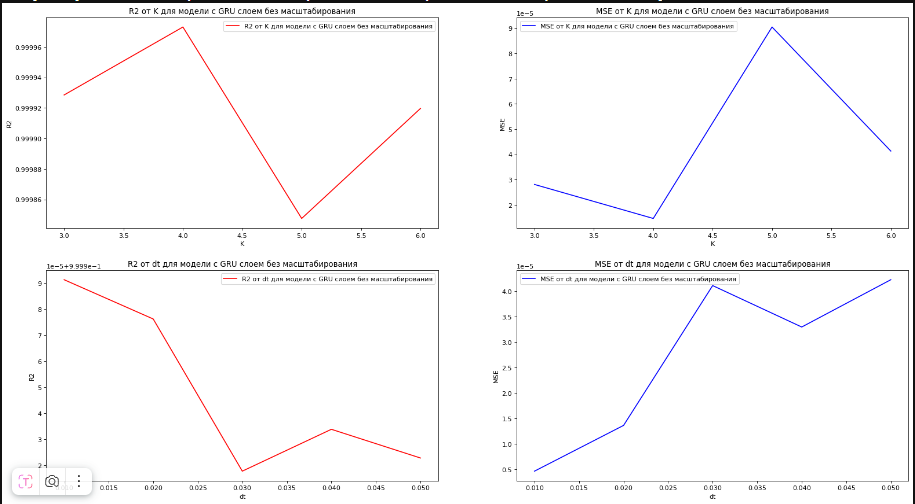
axs[1, 1].set\_xlabel("dt")

axs[1, 1].set\_ylabel("MSE")

axs[1, 1].set\_title("MSE от dt для модели с GRU слоем без масштабирования")

axs[1, 1].legend()

plt.show()



Для модели с GRU блоками без масштабирования лучшим значением k является 4, а dt 0.01

Построим графики рассчитанных значений для каждой модели при её лучших значениях k и dt

F, T, X, Y = X\_Y(6, 0.01)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

F\_scaled = scaler.fit\_transform(np.reshape(F, (len(F),1)))

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

fig, axs = plt.subplots(2, 2, figsize=(24, 13), dpi=80)

axs[0, 0].scatter(T\_test, Y\_1, color='r', label='Предсказанное', alpha=0.5)

axs[0, 0].plot(T, F\_scaled, color='g', label='GRU с масштабированием')

axs[0, 0].set\_xlabel("t")

axs[0, 0].set\_ylabel("X")

axs[0, 0].set\_title("GRU с масштабированием")

axs[0, 0].legend()

F, T, X, Y = X\_Y(6, 0.01)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

F\_scaled = scaler.fit\_transform(np.reshape(F, (len(F),1)))

net\_dense\_test = Sequential()

net\_dense\_test.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_test.add(Dense(units=64))

net\_dense\_test.add(Dense(units=32))

net\_dense\_test.add(Dense(units=1))

net\_dense\_test.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_test.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_test.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_test.predict(X\_test)

axs[0, 1].scatter(T\_test, Y\_1, color='r', label='Предсказанное', alpha=0.5)

axs[0, 1].plot(T, F\_scaled, color='g', label='Полносвязная с масштабированием')

axs[0, 1].set\_xlabel("t")

axs[0, 1].set\_ylabel("X")

axs[0, 1].set\_title("Полносвязная с масштабированием")

axs[0, 1].legend()

F, T, X, Y = X\_Y(6, 0.02)

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net = Sequential()

net.add(Input(shape=(X\_train.shape[1], 1)))

net.add(GRU(units=64, return\_sequences=True))

net.add(GRU(units=32))

net.add(Dense(units=1))

net.compile(optimizer='adam', loss='mean\_squared\_error')

history = net.fit(np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)), Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], X\_dev.shape[1], 1)), Y\_dev))

mse = net.evaluate(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)), Y\_test)

Y\_1 = net.predict(np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)))

axs[1, 0].scatter(T\_test, Y\_1, color='r', label='Предсказанное', alpha=0.5)

axs[1, 0].plot(T, F, color='g', label='GRU без масштабирования')

axs[1, 0].set\_xlabel("t")

axs[1, 0].set\_ylabel("X")

axs[1, 0].set\_title("GRU без масштабирования")

axs[1, 0].legend()

F, T, X, Y = X\_Y(6, 0.02)

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X, Y, T, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense\_test = Sequential()

net\_dense\_test.add(Input(shape=(X\_train.shape[1],)))

net\_dense\_test.add(Dense(units=64))

net\_dense\_test.add(Dense(units=32))

net\_dense\_test.add(Dense(units=1))

net\_dense\_test.compile(optimizer='adam', loss='mean\_squared\_error')

history = net\_dense\_test.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense\_test.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense\_test.predict(X\_test)

axs[1, 1].scatter(T\_test, Y\_1, color='r', label='Предсказанное', alpha=0.5)

axs[1, 1].plot(T, F, color='g', label='Полносвязная без масштабирования')

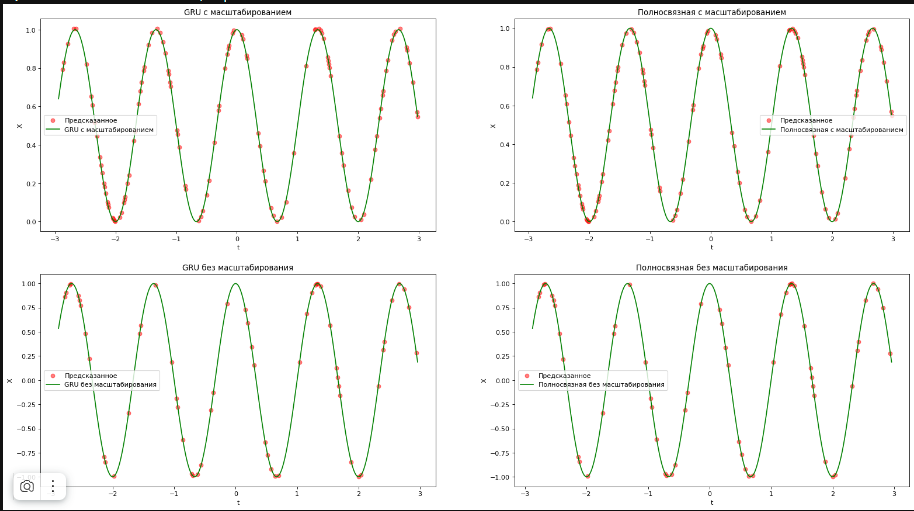
axs[1, 1].set\_xlabel("t")

axs[1, 1].set\_ylabel("X")

axs[1, 1].set\_title("Полносвязная без масштабирования")

axs[1, 1].legend()

plt.show()



F, T, X, Y = X\_Y(6, 0.03)

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y\_scaled = scaler.fit\_transform(np.reshape(Y, (Y.shape[0],1)))

X\_train, X\_test, Y\_train, Y\_test, T\_train, T\_test = train\_test\_split(X\_scaled, Y\_scaled, T, test\_size=0.2, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev, T\_test, T\_dev = train\_test\_split(X\_test, Y\_test, T\_test, test\_size=0.5, random\_state=42)

net\_dense = Sequential()

net\_dense.add(Input(shape=(X\_train.shape[1],)))

net\_dense.add(Dense(units=64))

net\_dense.add(Dense(units=32))

net\_dense.add(Dense(units=1))

net\_dense.compile(optimizer='adam', loss='mean\_squared\_error')

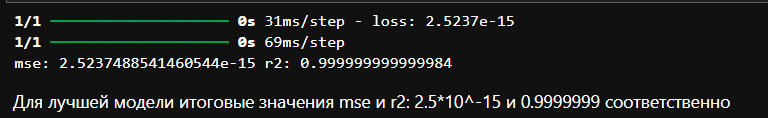
history = net\_dense.fit(X\_train, Y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = net\_dense.evaluate(X\_test, Y\_test)

Y\_1 = net\_dense.predict(X\_test)

r2 = r2\_score(Y\_test, Y\_1)

print(f"mse: {mse} r2: {r2}")



1. **Выводы**

В ходе выполнения лабораторной работы, было найдено решение задачи прогностического анализ данных на основе моделей нейронных сетей. Лучший результат показала полносвязная нейронная сеть с масштабированием данных