Задание 13-16.

Выполнить исследование и сравнительный анализ возможностей RNN,LSTM и GRU на примере предлагаемого кода.

https://github.com/Azure/lstms\_for\_predictive\_maintenance/blob/master/Deep%20Learning%20Basics%20for%20Predictive%20Maintenance.ipynb

Работаем с TensorFlow!

Accuracy Precision Recall F1-score

LSTM 0.978495 0.960000 0.96 0.960000

Template Best Model 0.940000 0.952381 0.80 0.869565

Accuracy Precision Recall F1-score

GRU 0.978495 0.925926 1.0 0.961538

Template Best Model 0.940000 0.952381 0.8 0.869565

Accuracy Precision Recall F1-score

RNN 0.892473 0.941176 0.64 0.761905

Template Best Model 0.940000 0.952381 0.80 0.869565

1. Сравнить полученные нейронные сети по Accuracy, Precision, Recall, F1, Loss на train и test.Выполнить несколько запусков программы с разными seed:

# Setting seed for reproducabilitynp.random.seed(1234)

Выбрать лучший вариант.

import numpy as np

from keras.models import clone\_model

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

seeds = [1234, 2345, 3456, 4567, 5678]

best\_model = None

best\_accuracy = 0

for seed in seeds:

    np.random.seed(seed)

    model\_clone = clone\_model(model)

    model\_clone.compile(

        loss="binary\_crossentropy", optimizer="adam", metrics=["accuracy"]

    )

    # Fit the model

    model\_clone.fit(

        seq\_array,

        label\_array,

        epochs=10,

        batch\_size=200,

        validation\_split=0.05,

        verbose=1,

        callbacks=[

            keras.callbacks.EarlyStopping(

                monitor="val\_loss", min\_delta=0, patience=0, verbose=0, mode="auto"

            )

        ],

    )

    y\_pred\_test = model\_clone.predict(seq\_array\_test\_last)

    y\_pred\_test\_classes = np.round(y\_pred\_test)

    accuracy = accuracy\_score(y\_true\_test, y\_pred\_test\_classes)

    precision = precision\_score(y\_true\_test, y\_pred\_test\_classes)

    recall = recall\_score(y\_true\_test, y\_pred\_test\_classes)

    f1 = f1\_score(y\_true\_test, y\_pred\_test\_classes)

    print(f"Seed: {seed}")

    print(f"Accuracy: {accuracy}")

    print(f"Precision: {precision}")

    print(f"Recall: {recall}")

    print(f"F1-score: {f1}")

    print("------------------------")

    if accuracy > best\_accuracy:

        best\_model = model\_clone

        best\_accuracy = accuracy

print(f"Best model performance (accuracy): {best\_accuracy}")

Epoch 1/10

75/75 [==============================] - 4s 32ms/step - loss: 0.2288 - accuracy: 0.9008 - val\_loss: 0.1362 - val\_accuracy: 0.9412

Epoch 2/10

75/75 [==============================] - 2s 28ms/step - loss: 0.1331 - accuracy: 0.9447 - val\_loss: 0.0841 - val\_accuracy: 0.9642

Epoch 3/10

75/75 [==============================] - 2s 31ms/step - loss: 0.0950 - accuracy: 0.9595 - val\_loss: 0.1282 - val\_accuracy: 0.9425

3/3 [==============================] - 0s 5ms/step

Seed: 1234

Accuracy: 0.9139784946236559

Precision: 1.0

Recall: 0.68

F1-score: 0.8095238095238095

------------------------

Epoch 1/10

75/75 [==============================] - 4s 30ms/step - loss: 0.1996 - accuracy: 0.9172 - val\_loss: 0.1944 - val\_accuracy: 0.9207

Epoch 2/10

75/75 [==============================] - 2s 27ms/step - loss: 0.1369 - accuracy: 0.9428 - val\_loss: 0.1857 - val\_accuracy: 0.9194

Epoch 3/10

75/75 [==============================] - 2s 28ms/step - loss: 0.1185 - accuracy: 0.9503 - val\_loss: 0.1712 - val\_accuracy: 0.9399

Epoch 4/10

75/75 [==============================] - 2s 28ms/step - loss: 0.1119 - accuracy: 0.9530 - val\_loss: 0.1052 - val\_accuracy: 0.9540

Epoch 5/10

75/75 [==============================] - 2s 27ms/step - loss: 0.0889 - accuracy: 0.9642 - val\_loss: 0.0634 - val\_accuracy: 0.9744

Epoch 6/10

75/75 [==============================] - 2s 26ms/step - loss: 0.0959 - accuracy: 0.9612 - val\_loss: 0.0745 - val\_accuracy: 0.9680

3/3 [==============================] - 0s 6ms/step

Seed: 2345

Accuracy: 0.967741935483871

Precision: 0.9230769230769231

Recall: 0.96

F1-score: 0.9411764705882353

------------------------

Epoch 1/10

75/75 [==============================] - 4s 30ms/step - loss: 0.2100 - accuracy: 0.9100 - val\_loss: 0.1183 - val\_accuracy: 0.9425

Epoch 2/10

75/75 [==============================] - 2s 28ms/step - loss: 0.1253 - accuracy: 0.9488 - val\_loss: 0.1128 - val\_accuracy: 0.9488

Epoch 3/10

75/75 [==============================] - 3s 34ms/step - loss: 0.1134 - accuracy: 0.9536 - val\_loss: 0.1118 - val\_accuracy: 0.9591

Epoch 4/10

75/75 [==============================] - 3s 39ms/step - loss: 0.0930 - accuracy: 0.9616 - val\_loss: 0.1593 - val\_accuracy: 0.9348

3/3 [==============================] - 1s 7ms/step

Seed: 3456

Accuracy: 0.8817204301075269

Precision: 0.9375

Recall: 0.6

F1-score: 0.7317073170731707

------------------------

Epoch 1/10

75/75 [==============================] - 5s 44ms/step - loss: 0.2161 - accuracy: 0.9053 - val\_loss: 0.1333 - val\_accuracy: 0.9297

Epoch 2/10

75/75 [==============================] - 3s 40ms/step - loss: 0.1304 - accuracy: 0.9456 - val\_loss: 0.1594 - val\_accuracy: 0.9437

3/3 [==============================] - 0s 6ms/step

Seed: 4567

Accuracy: 0.9247311827956989

Precision: 0.95

Recall: 0.76

F1-score: 0.8444444444444444

------------------------

Epoch 1/10

75/75 [==============================] - 5s 46ms/step - loss: 0.2263 - accuracy: 0.9018 - val\_loss: 0.1561 - val\_accuracy: 0.9373

Epoch 2/10

75/75 [==============================] - 3s 40ms/step - loss: 0.1233 - accuracy: 0.9499 - val\_loss: 0.1114 - val\_accuracy: 0.9540

Epoch 3/10

75/75 [==============================] - 3s 40ms/step - loss: 0.0908 - accuracy: 0.9628 - val\_loss: 0.0731 - val\_accuracy: 0.9693

Epoch 4/10

75/75 [==============================] - 3s 44ms/step - loss: 0.0991 - accuracy: 0.9588 - val\_loss: 0.1288 - val\_accuracy: 0.9425

3/3 [==============================] - 0s 8ms/step

Seed: 5678

Accuracy: 0.946236559139785

Precision: 0.9166666666666666

Recall: 0.88

F1-score: 0.8979591836734694

------------------------

Best model performance (accuracy): 0.967741935483871

2. Выполнить исследования на примере фрагмента кода с заменой

LSTM на RNN и GRU.

from keras.layers import SimpleRNN

# build the network

nb\_features = seq\_array.shape[2]

nb\_out = label\_array.shape[1]

model = Sequential()

model.add(

    SimpleRNN(

        input\_shape=(sequence\_length, nb\_features), units=100, return\_sequences=True

    )

)

model.add(Dropout(0.2))

model.add(SimpleRNN(units=50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=nb\_out, activation="sigmoid"))

model.compile(loss="binary\_crossentropy", optimizer="adam", metrics=["accuracy"])

from keras.layers import GRU

# build the network

nb\_features = seq\_array.shape[2]

nb\_out = label\_array.shape[1]

model = Sequential()

model.add(

    GRU(input\_shape=(sequence\_length, nb\_features), units=100, return\_sequences=True)

)

model.add(Dropout(0.2))

model.add(GRU(units=50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=nb\_out, activation="sigmoid"))

model.compile(loss="binary\_crossentropy", optimizer="adam", metrics=["accuracy"])

1. `nb\_features = seq\_array.shape[2]`: Устанавливает количество входных признаков равным количеству признаков во входных данных `seq\_array`.

2. `nb\_out = label\_array.shape[1]`: Устанавливает количество выходов равным количеству меток в `label\_array`.

3. `model = Sequential()`: Инициализирует последовательную модель. Это простой тип модели Keras, используемый для построения нейронных сетей по слоям.

4. `model.add(LSTM(input\_shape=(sequence\_length, nb\_features), units=100, return\_sequences=True))`: Добавляет слой LSTM в модель с 100 единицами. `input\_shape` определяет форму входных данных, в данном случае `(sequence\_length, nb\_features)`. `return\_sequences=True` означает, что слой LSTM будет возвращать полную последовательность выходных данных, а не только последний выход.

5. `model.add(Dropout(0.2))`: Добавляет слой Dropout в модель для предотвращения переобучения. Этот слой случайным образом устанавливает 20% (0.2) своих входных данных в 0 во время обучения.

6. `model.add(LSTM(units=50, return\_sequences=False))`: Добавляет еще один слой LSTM в модель с 50 единицами. `return\_sequences=False` означает, что этот слой LSTM будет возвращать только последний выход.

7. `model.add(Dense(units=nb\_out, activation='sigmoid'))`: Добавляет слой Dense в модель. Слой Dense - это полностью связанный слой нейронов, количество которых равно количеству выходов `nb\_out`. Функция активации 'sigmoid' используется для обеспечения того, чтобы выход модели был в диапазоне от 0 до 1, что подходит для задачи бинарной классификации.

8. `model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])`: Компилирует модель с функцией потерь 'binary\_crossentropy' (подходит для задачи бинарной классификации), оптимизатором 'adam' и метрикой производительности 'accuracy'.

print(model.summary())

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_1 (LSTM) (None, 50, 100) 50400

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 50, 100) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_2 (LSTM) (None, 50) 30200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 50) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 1) 51

=================================================================

Total params: 80,651

Trainable params: 80,651

Non-trainable params: 0

1. `lstm\_1 (LSTM)`: Слой LSTM имеет формулу для расчета параметров: `num\_params = 4 \* ((size\_of\_input + 1) \* size\_of\_output + size\_of\_output^2)`. В Нашем случае, размер входа равен 100 (nb\_features), размер выхода равен 100 (units). Поэтому, `num\_params = 4 \* ((100 + 1) \* 100 + 100^2) = 50400`.

2. `dropout\_1 (Dropout)`: Слой Dropout не имеет параметров для обучения, поэтому количество параметров равно 0.

3. `lstm\_2 (LSTM)`: Для этого слоя LSTM, размер входа равен 100 (размер выхода предыдущего слоя), а размер выхода равен 50. Поэтому, `num\_params = 4 \* ((100 + 1) \* 50 + 50^2) = 30200`.

4. `dropout\_2 (Dropout)`: Снова, слой Dropout не имеет параметров для обучения, поэтому количество параметров равно 0.

5. `dense\_1 (Dense)`: Для полносвязного слоя (Dense), количество параметров равно `(size\_of\_input + 1) \* size\_of\_output`. В данном случае, размер входа равен 50, а размер выхода равен 1. Поэтому, `num\_params = (50 + 1) \* 1 = 51`.

Общее количество параметров в модели равно сумме параметров всех слоев, что в данном случае составляет 80651.

%%time

# fit the network

model.fit(seq\_array, label\_array, epochs=10, batch\_size=200, validation\_split=0.05, verbose=1,

callbacks = [keras.callbacks.EarlyStopping(monitor='val\_loss', min\_delta=0, patience=0, verbose=0, mode='auto')])

Метод `model.fit()` используется для обучения модели в течение фиксированного количества эпох (итераций по набору данных).

- `seq\_array`: это массив входных данных для обучения модели.

- `label\_array`: это массив целевых (или ожидаемых) значений, соответствующих входным данным.

- `epochs=10`: число эпох, в течение которых модель будет обучаться. Одна эпоха - это один полный проход по всему набору обучающих данных.

- `batch\_size=200`: количество обучающих примеров, используемых в одной итерации обучения (то есть в одном обновлении градиента).

- `validation\_split=0.05`: доля данных, которые будут использоваться как валидационный набор. Эти данные не будут использоваться при обучении и вместо этого будут использоваться для оценки потерь и любых метрик модели в конце каждой эпохи.

- `verbose=1`: определяет, какую информацию об обучении выводить. 0 - не выводить информацию, 1 - выводить прогресс-бар, 2 - выводить одну строку на эпоху.

- `callbacks`: это список объектов keras.callbacks.Callback, которые применяются в процессе обучения. В данном случае используется только один колбек - EarlyStopping.

`keras.callbacks.EarlyStopping` - это колбек, который завершает обучение, когда метрика перестает улучшаться. Параметры этого колбека:

- `monitor='val\_loss'`: метрика, которую нужно отслеживать. В данном случае это потери на валидационном наборе.

- `min\_delta=0`: минимальное изменение метрики, которое считается улучшением.

- `patience=0`: количество эпох, в течение которых улучшения метрики могут отсутствовать, прежде чем обучение будет остановлено.

- `verbose=0`: определяет, какую информацию о ранней остановке выводить. 0 - не выводить информацию, 1 - выводить информацию.

- `mode='auto'`: определяет, должна ли метрика отслеживаться для улучшения (больше - лучше или меньше - лучше). 'auto' означает, что направление будет автоматически определено из имени отслеживаемой метрики.

history = model.fit(

    seq\_array,

    label\_array,

    epochs=10,

    batch\_size=200,

    validation\_split=0.05,

    verbose=1,

    callbacks=[

        keras.callbacks.EarlyStopping(

            monitor="val\_loss", min\_delta=0, patience=0, verbose=0, mode="auto"

        )

    ],

)

import matplotlib.pyplot as plt

# Biểu đồ cho Loss

plt.figure(figsize=(10, 6))

plt.plot(history.history["loss"], label="Train Loss")

plt.plot(history.history["val\_loss"], label="Val Loss")

plt.title("Model Loss")

plt.ylabel("Loss")

plt.xlabel("Epochs")

plt.legend(loc="upper right")

plt.show()

# Biểu đồ cho Accuracy

plt.figure(figsize=(10, 6))

plt.plot(history.history["accuracy"], label="Train Accuracy")

plt.plot(history.history["val\_accuracy"], label="Val Accuracy")

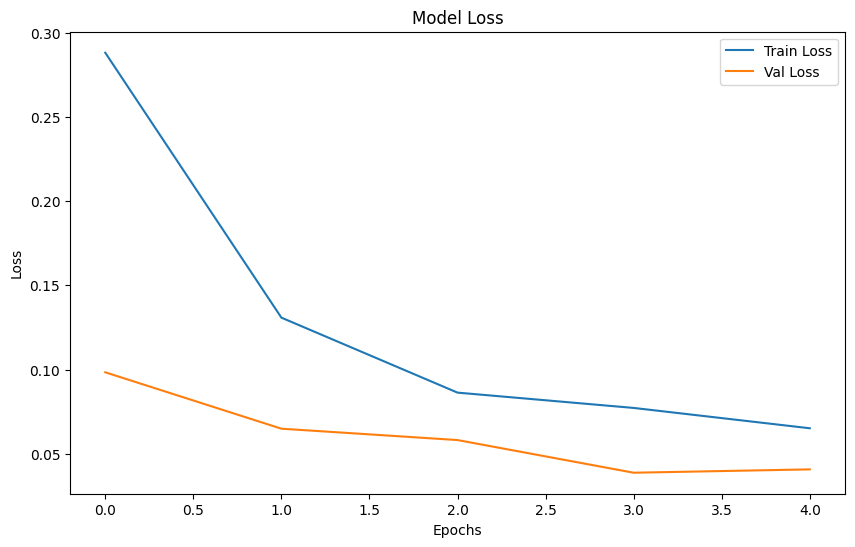
plt.title("Model Accuracy")

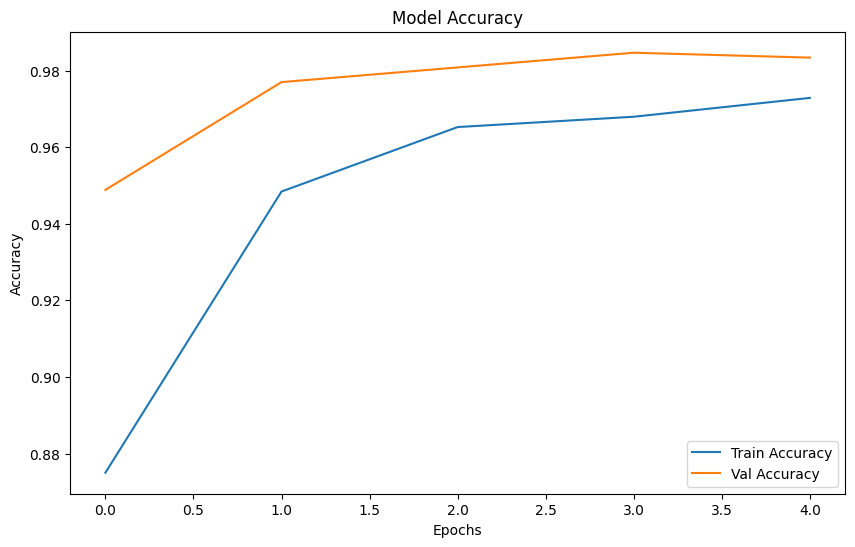
plt.ylabel("Accuracy")

plt.xlabel("Epochs")

plt.legend(loc="lower right")

plt.show()





%%time

model.fit(seq\_array, label\_array, epochs=10, batch\_size=200, validation\_split=0.05, verbose=1,

          callbacks = [keras.callbacks.EarlyStopping(monitor='val\_loss', min\_delta=0, patience=0, verbose=0, mode='auto')])

CPU times: user 11.4 s, sys: 882 ms, total: 12.3 s

Wall time: 15.5 s

<keras.src.callbacks.History at 0x7b6b747c9b40>

Время выполнения с графическим процессором быстрее, чем время выполнения с процессором