# **Лабораторная работа 6. Использование нейросетей для временных рядов**

## **Цели**

Получить базовые представления о работе и возможностях нейросетей. Построить нейросеть для решения задачи прогнозирования временных рядов с помощью готовых библиотек языка Python.

## **Задачи**

1. Подключить библиотеку TensorFlow
2. Построить нейросеть модель для решения задачи регрессии с использованием рекуррентных слоев

## **Теоретические сведения**

Методические указания для решения поставленного задания

### **Введение в нейронные сети**

Нейронные сети – метод машинного обучения, часто используемый на практике. Нейронная сеть строится из простых вычислительных элементов – искусственных нейронов. Искусственные нейроны – попытка воссоздания работы нейрона головного мозга в упрощенном виде.

Нейронные сети способны решать большое количество задач, но лучше их применять для решения эвристических задач. В данной работе нейросеть будет предсказывать стоимость акций на основе анализа временных рядов

Основное отличие нейронных сетей от других методов машинного обучения заключается в выборе признаков среди набора данных. В методах машинного обучения выбором занимается человек, нейросети же определяют важные признаки во время своего обучения. Из-за данной особенности возрастают требования к мощности устройства, на котором происходит обучение сети.

Наиболее простой тип нейронной сети – сеть прямого распространения. Более подходящая для решения поставленной задачи архитектура – рекуррентная. Рекуррентная архитектура подразумевает циклы внутри сети, и подходит для задач прогнозирования последовательностей и работы с текстом.

### **Нейронные сети в Python**

Для работы с нейронными сетями в данном курсе рекомендуется использовать библиотеку TensorFlow и её программный интерфейс Keras. Для установки библиотеки:

1. Откройте Anaconda
2. В меню слева выберете вкладку Environments
3. В списке установленных библиотек смените значение поля Installed на Not installed
4. В поле Search Packages впишите tensorflow
5. Среди найденных библиотек отметьте tensorflow
6. Нажмите на появившуюся снизу кнопку Apply

Пример построения и обучения нейросети с рекуррентной архитектурой будет приведен ниже, демонстрация будет проводится на участке отведенной бирже Euronext в наборе данных [Stock Exchange Data](https://www.kaggle.com/datasets/mattiuzc/stock-exchange-data) (рус. Данные фондовых бирж)

#### **Подготовка данных**

Подключение библиотек – matplotlib для графиков, numpy для работы с тензорами

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** tensorflow **as** tf

Загрузка данных

PATH = "dataset/indexData.csv"  
dataset = pd.read\_csv(PATH)

Извлечение параметра цены из набора данных в переменную price хранящую одномерный тензор размерности (5507,). Где 5507 – количество записей с ценой на момент закрытия биржи в определенную дату

EXCHANGE = "N100"  
dataset = dataset[dataset["Index"] == EXCHANGE]  
dataset = dataset.fillna(method='ffill')  
  
price = np.array(pd.to\_numeric(dataset["Close"]))

Для построения нейронной сети будет использоваться рекуррентная архитектура, а именно слои LSTM (рус. Долгая краткосрочная память). Слои LSTM принимают на вход трехмерные тензоры, где:

1. Первое число – общее количество записей
2. Второе число – количество записей, предшествующих записи с ценой
3. Третье число – количество записей с ценой

Ниже приведен код для создания трехмерного тензора train\_input на основе одномерного тензора price. Количество записей, предшествующих записи с ценой, т. е. второе число трехмерного тензора – 20

TIME\_STEPS = 20  
  
train\_input = np.array([])  
train\_output = np.array([])  
  
**for** index **in** range(TIME\_STEPS, price.size):  
 train\_input = np.append(  
 train\_input,  
 np.array(price[index - TIME\_STEPS: index]),  
 )  
 train\_output = np.append(  
 train\_output,  
 np.array(price[index]),  
 )  
  
train\_input = train\_input.reshape(int(train\_input.size / TIME\_STEPS), TIME\_STEPS, 1)  
train\_output = train\_output.reshape(train\_output.size, 1)

После чего получился тензор train\_input размерности (5487, 20, 1). Количество записей уменьшилось с 5507 до 5487 потому, что 20 записей в начале набора не могут иметь 20 предшествующих записей. Второе число зависит от переменной TIME\_STEPS и выбирается по усмотрению. Третье число 1, потому что у нас один входной признак – цена на момент закрытия.

Вместе с тензором train\_input для содержащем входные признаки – последовательность цен за 20 дней, был создан тензор train\_output для хранения выходных признаков – цену на 21 день. Размерность train\_output – (5487, 1), где первое число количество записей, а второе количество предсказаний.

Таким образом сеть будет предсказывать значение цены опираясь на значения предыдущих 20 значений.

#### **Проектирование архитектуры нейросети**

Создаётся переменная model для хранения нейросети, после чего в неё добавляются три слоя

1. Слой LSTM – 256 нейронов, размерность входных данных (20, 1), функция активации relu (рус. Выпрямленный линейный блок), возвращает всю последовательность поданных данных
2. Слой LSTM – 64 нейрона, размерность входных данных настраивается автоматически, функция активации relu, возвращает только результат
3. Слой Dense (рус. Полносвязный слой) – 1 нейрон

Данная архитектура не является эталонной для данной задачи, но показывает хорошие результаты. Попробуйте изменить количество слоёв LSTM и количество нейронов на этих слоях, также попробуйте другие функции активации

model = tf.keras.models.Sequential()  
  
model.add(  
 tf.keras.layers.LSTM(  
 256,  
 input\_shape =  
 (  
 train\_input.shape[1],  
 train\_input.shape[2]  
 ),  
 activation="relu",  
 return\_sequences=True)  
)  
  
model.add(  
 tf.keras.layers.LSTM(  
 64,  
 activation="relu",  
 return\_sequences=False  
 )  
)  
  
model.add(  
 tf.keras.layers.Dense(1)  
)  
  
model.compile(  
 optimizer='Adam',  
 loss='mse',  
)  
  
model.build(train\_input.shape)  
model.summary()

После добавления слоев, нейросеть подготавливается методами compile и build. Архитектуру сети можно узнать, вызвав метод summary

Используемые параметры обучения:

* Оптимизатор – Adam (рус. Адаптивная оценка моментов). Не эталонный – попробуйте другие
* Метрика ошибки – mse (рус. Среднеквадратическая ошибка)

#### **Обучение нейросети**

Нейросеть начинает обучаться при вызове метода fit в который передаются параметры

1. train\_input – Входные признаки
2. train output – Выходные признаки
3. validation split – разделение выборки на обучающую и проверочную, выбрано использовать 0.2 выборки как проверочные
4. epochs – количество эпох для обучения сети
5. verbose – вывод данных об обучении на экран

Количество эпох не является эталонным, попробуйте разные значения. Также на вашем устройстве может не хватить оперативной памяти для обучения нейросети – для решения этой проблемы используется параметр batch\_size, попробуйте разные значения, которые обычно принимаются степенью двойки.

history = model.fit(  
 train\_input,  
 train\_output,  
 *# batch\_size=128,*  
 validation\_split=0.2,  
 epochs=20,  
 verbose=True,  
)

Далее, проверяем пригодность нейросети проверяя среднеквадратическую ошибку её предсказаний на тестовом наборе и вместе с тем сохраняя модель

error = int(  
 model.evaluate(  
 train\_input,  
 train\_output  
 )  
)  
  
model.save(f"model/{model.loss} {error}.h5")

Позже модель можно будет загрузить, передав её в переменную методом load\_model

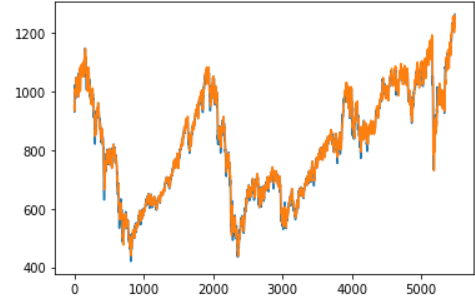
model = tf.keras.models.load\_model("model/mse 258.h5")

Далее строим график для проверки работы нейросети

plt.plot(train\_output)  
plt.plot(  
 model.predict(  
 train\_input,  
 verbose=False  
 )  
)  
plt.show()

Ниже представлен график работы нейросети:

1. Зеленая линия – эталонные значения
2. Оранжевая линия – предсказания сети



Среднеквадратическая ошибка сети – 258

#### **Предсказание**

Далее представлен код для предсказания значений цен на следующие 365 дней отсутствующих в наборе. Так как нейросеть обучалась на последовательностях из 20 значений, то и предсказания она делает, получая на вход только такие же последовательности, за создание таких последовательностей отвечает цикл в коде ниже

PREDICTIONS\_NUMBER = 365  
  
prediction\_input = train\_input.copy()  
prediction\_output = np.array([])  
  
**for** \_ **in** range(PREDICTIONS\_NUMBER):  
 prediction = model.predict(  
 prediction\_input[-1].reshape(1, TIME\_STEPS, 1),  
 verbose=False  
 )  
 prediction\_output = np.append(  
 prediction\_output,  
 prediction  
 )  
  
 new\_time\_series = prediction\_input[-1][1:]  
 new\_time\_series = np.append(new\_time\_series, [prediction])  
  
 shape = prediction\_input.shape  
  
 prediction\_input = np.append(  
 prediction\_input,  
 [new\_time\_series]  
 )  
  
 prediction\_input = prediction\_input.reshape(  
 int(prediction\_input.shape[0] / shape[1]),  
 shape[1],  
 shape[2]  
 )  
  
prediction\_output = prediction\_output.reshape(  
 PREDICTIONS\_NUMBER,  
 1  
)  
  
prediction\_output

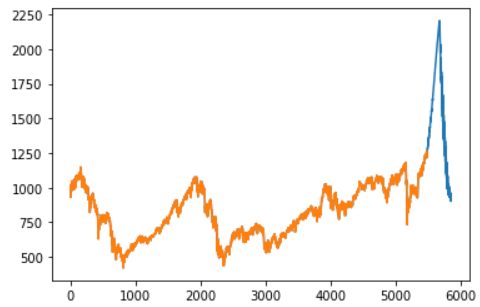
После выполнения кода на экран должны выводится предсказанные значения в текстовом виде.

Для вывода этих же данных используется следующий код

prediction\_output = np.append(  
 train\_output,  
 prediction\_output  
)  
  
plt.plot(prediction\_output)  
plt.plot(train\_output)  
plt.show()

Полученный график:

* Оранжевая линия – имеющиеся данные
* Синяя линия – предсказанные значения



## **Задание**

Выбрать с сайта [kaggle.com](https://www.kaggle.com/) набор данных в формате .csv, содержащий данные типа «дата-значение». Загрузить и подготовить его к дальнейшей обработке. Наборы данных не должны повторяться внутри группы. Задание индивидуальное. Требования:

1. Подготовить данные к обработке слоем LSTM
2. Построить нейросеть
3. Обучить нейросеть
4. Сравнить эталонные значения и предсказания на данных для обучения
5. Предсказать следующие 20 значений
6. Сохранить нейросеть
7. Сохранить IPython Notebook

### **Продвинутое задание**

Улучшить архитектуру нейросети с помощью Keras Tuner