## Вариант 8

Необходимо построить рекуррентную нейронную сеть, которая будет прогнозировать значение некоторого периодического сигнала.

К каждому варианту предоставляется код, который генерирует последовательность. Для выполнения задания необходимо:

- 1. Преобразовать последовательность в датасет, который можно подавать на вход нейронной сети (можно использовать функцию gen data from sequence из примера)
- 2. Разбить датасет на обучающую, контрольную и тестовую выборку
- 3. Построить и обучить модель
- 4. Построить график последовательности, предсказанной на тестовой выборке (пример построения также есть в примере). Данный график необходимо также добавить в рг

Также, в файлах с кодом вариантов есть функция draw\_sequence, которая позволяет нарисовать часть последовательности

## Выполнение работы.

Функция, которая генерирует последовательность, представлена в листинге 1.

### Листинг 1 – Функция генерации последовательности

```
def gen_sequence(seq_len=1000):
    seq = [abs(math.sin(i / 20)) + func(i) +
    random.normalvariate(0, 0.04) for i in range(seq_len)]
    return np.array(seq)
```

Последовательность преобразована в датасет посредством функции gen\_data\_from\_sequence (листинг 2).

# Листинг 2 – Функция gen\_data\_from\_sequence

```
def gen_data_from_sequence(seq_len=1006, lookback=10):
    seq = gen_sequence(seq_len)
    past = np.array([[[seq[j]] for j in range(i, i + lookback)]
    for i in range(len(seq) - lookback)])
```

```
future = np.array([[seq[i]] for i in range(lookback,
len(seq))])
return past, future
```

Датасет был разбит на обучающую, контрольную и тестовую выборку (листинг 3). Обучающие данные используют 70% от датасета, контрольные – 50% от датасета за вычетом обучающих данных, и тестовые – оставшиеся данные.

## Листинг 3 – Разбиение данных

```
data, res = gen_data_from_sequence()
dataset_size = len(data)
train_size = (dataset_size // 10) * 7
val_size = (dataset_size - train_size) // 2
train_data, train_res = data[:train_size], res[:train_size]
val_data, val_res = data[train_size:train_size + val_size],
res[train_size:train_size + val_size]
test_data, test_res = data[train_size + val_size:], res[train_size + val_size:]
```

Затем была построена и обучена модель (листинг 4).

### Листинг 4 – Построение и обучение модели

После этого, были построены графики последовательности, предсказанной на тестовой выборке (листинг 5).

#### Листинг 5 – Построение графиков

```
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(range(len(loss)), loss)
```

```
plt.plot(range(len(val_loss)), val_loss)
plt.show()

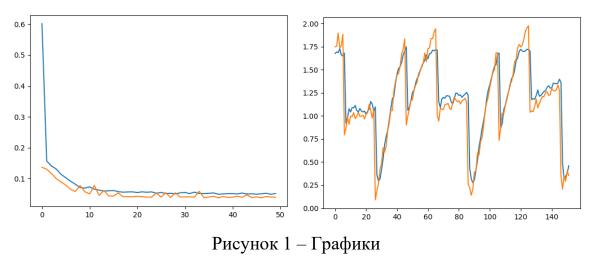
predicted_res = model.predict(test_data)
pred_length = range(len(predicted_res))
plt.plot(pred_length, predicted_res)
plt.plot(pred_length, test_res)
plt.show()
```

Результат выполнения программы:

Epoch 50/50

22/22 [=============] - 0s 8ms/step - loss: 0.0502 - val\_loss: 0.0474

Потери на обучающих данных составляют 0.05, на контрольных -0.047. Графики представлены на рис. 1.



Первый график – график ошибок на тренировочных (синяя кривая) и на контрольных данных (оранжевая). На втором графике показано предсказание некоторого периодического сигнала (синий график) и то, как сигнал должен выглядеть (оранжевый).

Если добавить нормализацию данных (листинг 5), то потери на тренировочных упадут до 0.043, а на контрольных — до 0.037. Графики см. на рис. 2.

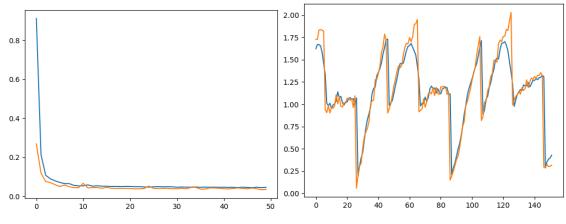


Рисунок 2 – Графики