Практика 7

Вариант 4

Задание

Необходимо построить рекуррентную нейронную сеть, которая будет прогнозировать значение некоторого периодического сигнала. К каждому варианту предоставляется код, который генерирует последовательность. Для выполнения задания необходимо:

- Преобразовать последовательность в датасет, который можно подавать на вход нейронной сети (можно использовать функцию `gen_data_from_sequence` из примера)
- Разбить датасет на обучающую, контрольную и тестовую выборку
- Построить и обучить модель
- Построить график последовательности, предсказанной на тестовой выборке (пример построения также есть в примере). Данный график необходимо также добавить в pr

Генерация данных

Датасет генерируется с использованием функции, приведенной в примере:

```
def gen_data_from_sequence(seq_len = 1006, lookback = 10):
    seq = var4.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)
```

При помощи функции `gen_sequence`, содержащейся в файле с вариантом задания, генерируется последовательность. Часть сгенерированной последовательности изображена на графике на рис. 1.

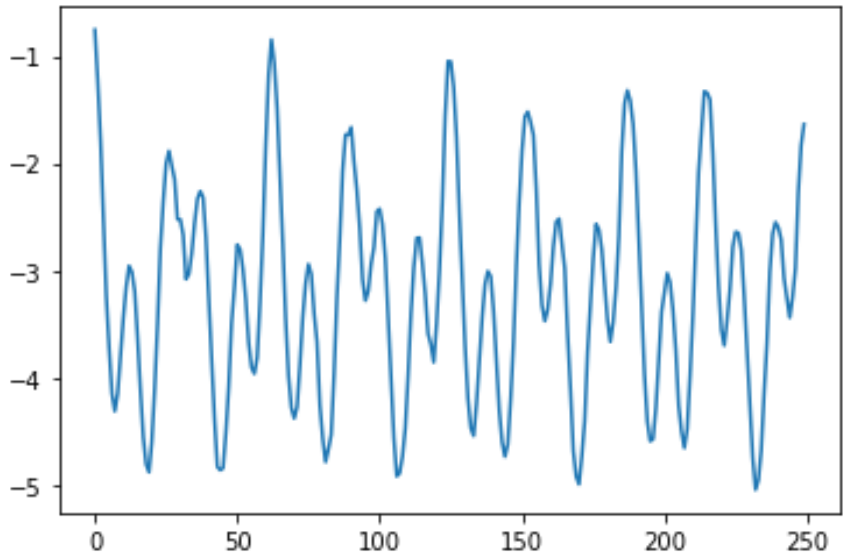

Рисунок 1

Индексы элементов в последовательности можно считать значениями времени. В качестве входных данных используются векторы длины lookback, содержащие значения последовательности на интервале $[i, i + lookback - 1]$. В качестве выходных данных используются значения последовательности в момент $i + lookback$. Здесь $i = 0, 1, \ldots, len(seq) - lookback - 1$. Таким образом, нейросеть должна предсказывать следующее значение последовательности по lookback предыдущим.

Разбиение выборки

Разобьем датасет на обучающую и тестовую выборки, а контрольную выделим уже при обучении модели. Обучающая выборка будет составлять 90% всего набора данных, тестовая – остальные 10%.

```
dataset_size = len(data)
train_size = (dataset_size // 10) * 9

train_data, train_res = data[:train_size], res[:train_size]
test_data, test_res = data[train_size:], res[train_size:]
```

Построение модели

Построим модель:

```
model = Sequential()
```

```
model.add(layers.GRU(32,recurrent_activation='sigmoid',input_shape=(None,1),return_sequences=True))
model.add(layers.GRU(32,activation='relu',input_shape=(None,1),return_sequences=True))
model.add(layers.GRU(32,input_shape=(None,1),recurrent_dropout=0.2))
model.add(layers.Dense(1))
model.compile(optimizer='nadam', loss='mse')
```

Слой LSTM из примера был заменен на еще один слой GRU (разницы замечено не было, мне просто так больше нравится), и с этого слоя было убрано прореживание входных данных, поскольку без него потери были меньше.

Модель была обучена в течение 50 эпох пакетами по 32 образца. На контрольную выборку отведено 15% данных. На рисунке 2 изображены графики потерь на тренировочной и контрольной выборках.

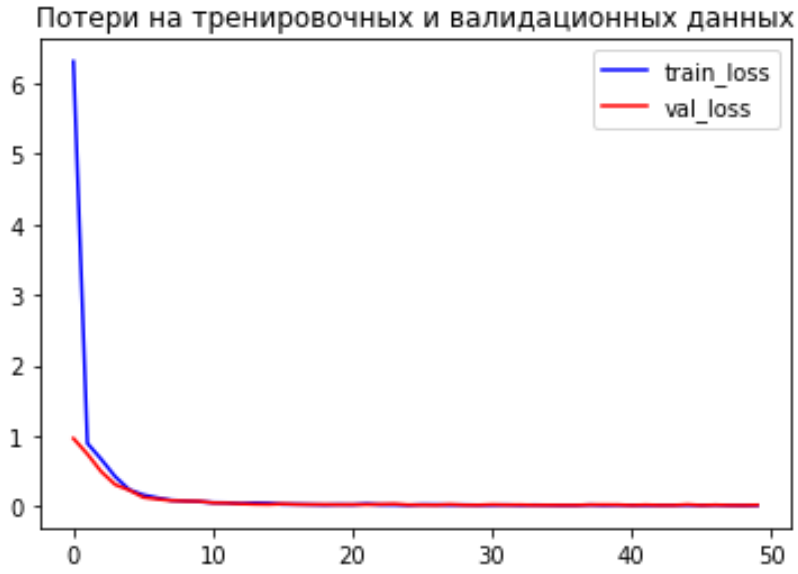

Рисунок 2

График предсказанной последовательности

На рисунке 3 изображены график сгенерированной последовательности и график предсказанной последовательности.

Рисунок 3

Как видно на рисунке, значения последовательности предсказываются в целом очень точно, но в стационарных точках наблюдаются небольшие отклонения.