1. **Прогнозирование временных рядов на основе прототипов Seq-2-Seq моделей с использованием однонаправленных и двунаправленных LSTM с глубоким стеком и реализации в Keras**

Ссылка на первоисточник «[https://weiminwang.blog/2017/09/29/multivariate-time-series-forecast-using-seq2seq-in-tensorflow/»](https://weiminwang.blog/2017/09/29/multivariate-time-series-forecast-using-seq2seq-in-tensorflow/%C2%BB) Вэйминь Ванга.

Этот ноутбук реализует четыре архитектуры на основе кодировщика-декодера s2q2seq с использованием сетей LSTM. Одно / многоуровневые однонаправленные и двунаправленные LSTM были опробованы и реализованы на наборе данных одномерного временного ряда «аукциннная цена 1000 кликов» с целью выыбора наиболее подхлдящей для практики модели.

Основная цель - выбрать наиболее эффективную модель LSTM для дальнейшей реализации в прогнозировании и изучении сложных наборов временных данных в системе аукционных торгов и построении торгового алгоритма.

In [1]:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **time**

**from** **keras.models** **import** model\_from\_json

%matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

sns.set\_style("darkgrid")

**import** **tensorflow** **as** **tf**

**from** **tensorflow.keras.models** **import** Model

**from** **tensorflow.keras.layers** **import** Input, LSTM, Dense, LSTMCell, RNN, Bidirectional, concatenate

**from** **tensorflow.keras.optimizers** **import** Adam

**from** **tensorflow.python.keras.callbacks** **import** TensorBoard

**from** **sklearn.preprocessing** **import** MinMaxScaler

**Обработка исходных данных**

Исходный набор данных требует небольшой очистки, которую проделаем с помощью программных фильтров

In [2]:

data=pd.read\_csv('cl.csv', sep=';', error\_bad\_lines=**False**)

data = data.query('Bid < 5000')

data = data.reset\_index()

C:\Users\nnig9\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (12) have mixed types. Specify dtype option on import or set low\_memory=False.

interactivity=interactivity, compiler=compiler, result=result)

In [3]:

data['Bid'] = data['Bid'].apply(**lambda** x: float(x))

data['date'] = pd.to\_datetime(data['date'])

In [4]:

data[:-10000]

Out[4]:

|  | **index** | **Click\_ID** | **date** | **banner\_type** | **banner\_width** | **banner\_heigth** | **IP** | **Device\_ID** | **Bid** | **OS** | **OS\_Version** | **Device\_mode** | **Connection\_Type** | **Unnamed: 12** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 35803771 | 2019-09-23 21:46:51 | simple | 768 | 1024 | 91.233.43.235 | 26D00212-C469-4A3C-824D-8FAD9CA1554C | 353.0 | iOS | ios 9.3 | Apple iPad 1 | WiFi | NaN |
| **1** | 1 | 35803772 | 2019-09-23 21:47:17 | simple | 320 | 480 | 5.18.232.72 | e51b5d0c-e147-4c7c-a7d9-f54fdbe84918 | 712.0 | Android | android 8 | huawei honor 9 lite | None | NaN |
| **2** | 2 | 35803773 | 2019-09-23 21:47:40 | simple | 320 | 480 | 62.33.118.27 | 5b47abce-0e7c-46c4-97cf-06e8a4746ef2 | 353.0 | Android | android 5.0.2 | lenovo a6010 | WiFi | NaN |
| **3** | 3 | 35803774 | 2019-09-23 21:49:14 | simple | 320 | 480 | 46.8.244.223 | 19b84b3c-ddcc-4010-815b-1951f77515bf | 353.0 | Android | android 4.4 | None | WiFi | NaN |
| **4** | 4 | 35803775 | 2019-09-23 21:49:30 | simple | 320 | 480 | 85.93.58.149 | b61f2a4d-e008-4526-8654-510c5e4b774f | 353.0 | Android | android 6.0 | micromax q354 | WiFi | NaN |

1387525 rows × 14 columns

In [5]:

*###### series = data['Bid'][:].reset\_index()*

series = data['Bid'][-3000:].reset\_index()

series = series.drop('index',axis=1)

In [6]:

*#series*

In [7]:

seq = series.copy()

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

X =scaler.fit\_transform(seq.values.reshape(-1,1))

*#seq = np.log(seq.values)*

In [8]:

series = X

seq = series.copy()

seqq = series[-1000:].copy()

plt.figure(figsize=(15,7))

plt.title("Sequence", fontsize = 18)

plt.xlabel("Index", fontsize = 18)

plt.ylabel("Value", fontsize = 18)

plt.plot(seq[:])

plt.show()

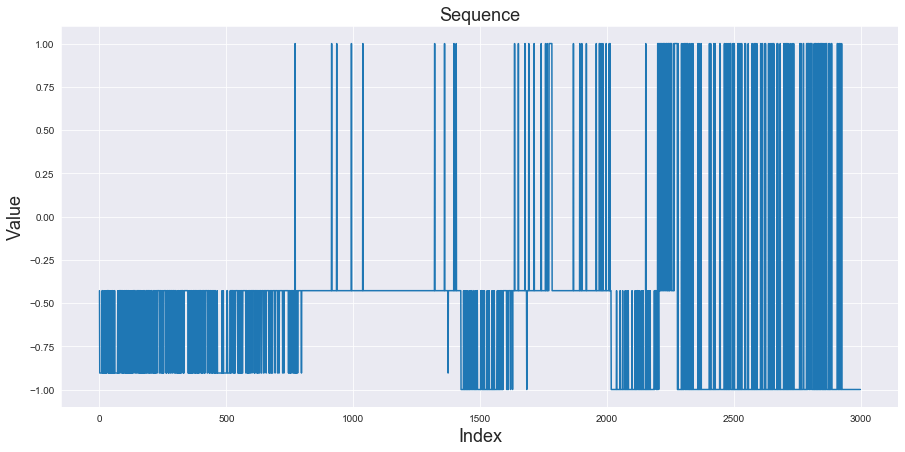


Рис. График исходного временного ряда

In [9]:

*#scaler = MinMaxScaler(feature\_range=(-1,1))*

*#X =scaler.fit\_transform(seq.values.reshape(-1,1))*

In [10]:

*#X[:100]*

In [11]:

*#data['Bid']*

*# plt.plot(X[:100])*

*# plt.plot(np.log(data['Bid'][:200]))*

In [12]:

*#1397525*

x\_train = X[:-400]

x\_test = X[-400:]

In [13]:

x\_train

Out[13]:

array([[-0.42857143],

[-0.9047619 ],

[-0.9047619 ],

...,

[-1. ],

[-1. ],

[-1. ]])

**Обратите внимание**, что в приведенных выше ячейках мы не делали никаких изменений в серии, чтобы преобразовать ее в модель обучения с учителем. Вскоре вы увидите причину, по которой мы этого не сделали.

В ячейке ниже определяется длина входной и выходной последовательности. Мы также определяем номера входных характеристик (столбцов) мы даем модели, а номер ожидаемых выходных характеристик (столбцов). В случае многомерных данных нет.номера входных функций будет> 1 в зависимости от того, сколько вы включаете в модель.

In [14]:

input\_seq\_len = 60

output\_seq\_len = 20

n\_in\_features = 1

n\_out\_features = 1

batch\_size = 10

batch\_size

Out[14]:

10

**Помощник для создания тренировочных пакетов.**

Всего "input\_seq\_len + Output\_seq\_len" нет. точек будет формировать единую обучающую выборку, где наши входы (X) (входы кодировщика) будут «input\_seq», а наши цели (Y) (цели декодера) будут «output\_seq». По этой причине мы не сместили последовательность на один шаг при генерации разбиений на тренировка-тест.  
Здесь мы выбираем случайные индексы и создаем такие выборки, чтобы сформировать пакет размером = batch\_size.

In [15]:

**def** generate\_train\_sequences(x):

total\_start\_points = len(x) - input\_seq\_len - output\_seq\_len

start\_x\_idx = np.random.choice(range(total\_start\_points), total\_start\_points, replace = **False**)

input\_batch\_idxs = [(range(i, i+input\_seq\_len)) **for** i **in** start\_x\_idx]

input\_seq = np.take(x, input\_batch\_idxs, axis = 0)

output\_batch\_idxs = [(range(i+input\_seq\_len, i+input\_seq\_len+output\_seq\_len)) **for** i **in** start\_x\_idx]

output\_seq = np.take(x, output\_batch\_idxs, axis = 0)

input\_seq =(input\_seq.reshape(input\_seq.shape[0],input\_seq.shape[1],n\_in\_features))

output\_seq=(output\_seq.reshape(output\_seq.shape[0],output\_seq.shape[1],n\_out\_features))

**return** input\_seq, output\_seq

In [16]:

*#generate\_train\_sequences(X)*

**Построить архитектуру модели**

In [17]:

**def** create\_model(layers, bidirectional = **False**):

n\_layers = len(layers)

*## Encoder*

encoder\_inputs = Input(shape = (**None**, n\_in\_features))

lstm\_cells = [LSTMCell(hidden\_dim) **for** hidden\_dim **in** layers]

**if** bidirectional:

encoder = Bidirectional(RNN(lstm\_cells, return\_state=**True**))

encoder\_outputs\_and\_states = encoder(encoder\_inputs)

bi\_encoder\_states = encoder\_outputs\_and\_states[1:]

encoder\_states = []

**for** i **in** range(int(len(bi\_encoder\_states) / 2)):

temp = []

**for** j **in** range(2):

temp.append(concatenate([bi\_encoder\_states[i][j], bi\_encoder\_states[n\_layers + i][j]], axis = -1))

encoder\_states.append(temp)

**else**:

encoder = RNN(lstm\_cells, return\_state = **True**)

encoder\_outputs\_and\_states = encoder(encoder\_inputs)

encoder\_states = encoder\_outputs\_and\_states[1:]

*## Decoder*

decoder\_inputs = Input(shape = (**None**, n\_out\_features))

**if** bidirectional:

decoder\_cells = [LSTMCell(hidden\_dim\*2) **for** hidden\_dim **in** layers]

**else**:

decoder\_cells = [LSTMCell(hidden\_dim) **for** hidden\_dim **in** layers]

decoder\_lstm = RNN(decoder\_cells, return\_sequences = **True**, return\_state=**True**)

decoder\_outputs\_and\_states = decoder\_lstm(decoder\_inputs, initial\_state = encoder\_states)

decoder\_outputs = decoder\_outputs\_and\_states[0]

decoder\_dense = Dense(n\_out\_features)

decoder\_outputs = decoder\_dense(decoder\_outputs)

model = Model([encoder\_inputs,decoder\_inputs], decoder\_outputs)

**return** model

**Обратите внимание**, что если выбрана двунаправленность, тогда длина ячеек декодера будет вдвое больше, чем у кодера, поскольку мы объединяем прямое и обратное состояния кодера.

**Запускаем model.fit**

In [18]:

callbacks = [TensorBoard(log\_dir='tb\_logs\_s2s', histogram\_freq=1, write\_images=**True**)]

In [19]:

**def** run\_model(model,batches,epochs,batch\_size):

**global** history

**for** \_ **in** range(batches):

input\_seq, output\_seq = generate\_train\_sequences(x\_train)

encoder\_input\_data = input\_seq

decoder\_target\_data = output\_seq

decoder\_input\_data = np.zeros(decoder\_target\_data.shape)

history = model.fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,

batch\_size=batch\_size,

epochs=epochs,

validation\_split=0.1,

shuffle=**False**,

callbacks=callbacks)

total\_loss.append(history.history['loss'])

total\_val\_loss.append(history.history['val\_loss'])

In [20]:

**def** plot\_loss(train\_loss,val\_loss):

plt.figure(figsize=(12,7))

plt.plot(train\_loss)

plt.plot(val\_loss)

plt.xlabel('Epoch')

plt.ylabel('Mean Sqquared Error Loss')

plt.title('Loss Over Time')

plt.legend(['Train','Valid'])

plt.show()

1. **1. Однонаправленная 1-слойная модель**

In [21]:

model\_1 = create\_model(layers=[100],bidirectional=**False**)

In [22]:

*# from keras.models import model\_from\_json*

*# # load json and create model*

*# json\_file = open('model\_1.json', 'r')*

*# loaded\_model\_json = json\_file.read()*

*# json\_file.close()*

*# model\_1 = model\_from\_json(loaded\_model\_json)*

*# # load weights into new model*

*# model\_1.load\_weights("model\_1.h5")*

*# print("Loaded model from disk")*

**Обратите внимание**, что аргумент Layers должен быть списком latent\_dims в каждом слое. Пример для 2 слоев по 40 димов в каждом ==> [40,40].

In [23]:

total\_loss = []

total\_val\_loss = []

model\_1.compile(Adam(), loss = 'mean\_squared\_error')

In [24]:

start\_time = time.time()

run\_model(model\_1,batches=1, epochs=100, batch\_size=batch\_size)

end\_time = time.time()

Epoch 1/100

1/227 [..............................] - ETA: 0s - loss: 0.4807WARNING:tensorflow:From C:\Users\nnig9\anaconda3\lib\site-packages\tensorflow\python\ops\summary\_ops\_v2.py:1277: stop (from tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-07-01.

Instructions for updating:

use `tf.profiler.experimental.stop` instead.

2/227 [..............................] - ETA: 21s - loss: 0.4735WARNING:tensorflow:Callbacks method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0240s vs `on\_train\_batch\_end` time: 0.1610s). Check your callbacks.

227/227 [==============================] - 4s 18ms/step - loss: 0.1686 - val\_loss: 0.1426

Epoch 2/100

227/227 [==============================] - 4s 17ms/step - loss: 0.1584 - val\_loss: 0.1420

Epoch 3/100

227/227 [==============================] - 4s 17ms/step - loss: 0.1577 - val\_loss: 0.1413

Epoch 4/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1569 - val\_loss: 0.1395

Epoch 5/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1531 - val\_loss: 0.1300

Epoch 6/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1521 - val\_loss: 0.1306

Epoch 7/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1506 - val\_loss: 0.1300

Epoch 8/100

227/227 [==============================] - 4s 17ms/step - loss: 0.1499 - val\_loss: 0.1285

Epoch 9/100

227/227 [==============================] - 4s 17ms/step - loss: 0.1490 - val\_loss: 0.1282

Epoch 10/100

, , , , , , , ,

Epoch 90/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0920 - val\_loss: 0.0999

Epoch 91/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0911 - val\_loss: 0.0983

Epoch 92/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0914 - val\_loss: 0.0924

Epoch 93/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0907 - val\_loss: 0.1127

Epoch 94/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0986 - val\_loss: 0.1055

Epoch 95/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0898 - val\_loss: 0.0981

Epoch 96/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0862 - val\_loss: 0.1117

Epoch 97/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0922 - val\_loss: 0.0997

Epoch 98/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0875 - val\_loss: 0.0966

Epoch 99/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0845 - val\_loss: 0.1028

Epoch 100/100

227/227 [==============================] - 4s 17ms/step - loss: 0.0879 - val\_loss: 0.0966

In [25]:

*#history.history['loss']*

In [26]:

run\_time = (end\_time - start\_time)/60

run\_time

Out[26]:

6.43873147169749

In [27]:

total\_loss = [j **for** i **in** total\_loss **for** j **in** i]

total\_val\_loss = [j **for** i **in** total\_val\_loss **for** j **in** i]

In [29]:

history.history.keys()

Out[29]:

dict\_keys(['loss', 'val\_loss'])

In [30]:

**def** plot\_loss(train\_loss,val\_loss):

plt.figure(figsize=(12,7))

plt.plot(train\_loss)

plt.plot(val\_loss)

plt.xlabel('Epoch')

plt.ylabel('Mean Sqquared Error Loss')

plt.title('Loss Over Time')

plt.legend(['Train','Valid'])

plt.show()

In [31]:

plot\_loss(total\_loss,total\_val\_loss)

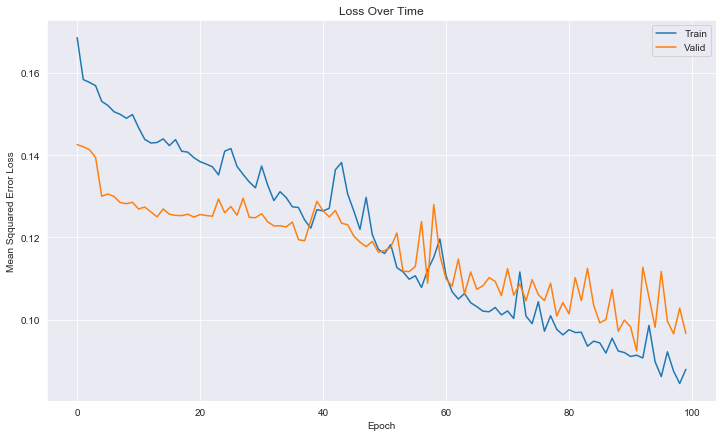


Рис. График зависимости ошибки обучения от количества эпох

In [32]:

input\_seq\_test = x\_train[-60:].reshape((1,60,1))

output\_seq\_test = x\_test[:20]

decoder\_input\_test = np.zeros((1,output\_seq\_len,1))

In [34]:

*#input\_seq\_test*

In [35]:

pred1 = model\_1.predict([input\_seq\_test,decoder\_input\_test])

In [36]:

pred\_values1 = scaler.inverse\_transform(pred1.reshape(-1,1))

output\_seq\_test1 = scaler.inverse\_transform(output\_seq\_test)

In [37]:

plt.plot(pred\_values1, label = "pred")

plt.plot(output\_seq\_test1, label = "actual")

plt.title("1-напр 1-слойный")

plt.ylabel("Bid", fontsize=12)

plt.xlabel("Time", fontsize=12)

plt.legend()

plt.savefig('uni\_dir1.png')

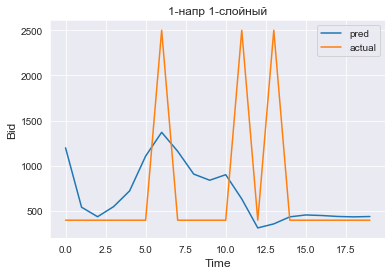


Рис. График прогнозируемой функции

In [38]:

*# serialize model to JSON*

*# model\_json = model\_1.to\_json()*

*# with open("model\_1.json", "w") as json\_file:*

*# json\_file.write(model\_json)*

*# # serialize weights to HDF5*

*# model\_1.save\_weights("model\_1.h5")*

*# print("Saved model to disk")*

1. **2. Двунаправленная 1-слойная модель**

In [39]:

model1\_bi = create\_model(layers=[60],bidirectional=**True**)

In [40]:

*# from keras.models import model\_from\_json*

*# # load json and create model*

*# json\_file = open('model1\_bi.json', 'r')*

*# loaded\_model\_json = json\_file.read()*

*# json\_file.close()*

*# model1\_bi = model\_from\_json(loaded\_model\_json)*

*# # load weights into new model*

*# model1\_bi.load\_weights("model1\_bi.h5")*

*# print("Loaded model from disk")*

In [41]:

total\_loss = []

total\_val\_loss = []

model1\_bi.compile(Adam(), loss = 'mean\_squared\_error')

In [ ]:

In [42]:

*#bi 1 layered model*

start\_time = time.time()

run\_model(model1\_bi,batches=1, epochs=100, batch\_size=batch\_size)

end\_time = time.time()

Epoch 1/100

2/227 [..............................] - ETA: 48s - loss: 0.3658WARNING:tensorflow:Callbacks method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0266s vs `on\_train\_batch\_end` time: 0.3939s). Check your callbacks.

227/227 [==============================] - 4s 19ms/step - loss: 0.1676 - val\_loss: 0.1382

Epoch 2/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1560 - val\_loss: 0.1364

Epoch 3/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1531 - val\_loss: 0.1430

Epoch 4/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1519 - val\_loss: 0.1361

Epoch 5/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1476 - val\_loss: 0.1329

Epoch 6/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1470 - val\_loss: 0.1330

Epoch 7/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1454 - val\_loss: 0.1311

Epoch 8/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1449 - val\_loss: 0.1312

Epoch 9/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1440 - val\_loss: 0.1315

Epoch 10/100

227/227 [==============================] - 4s 16ms/step - loss: 0.1426 - val\_

. . . . . . . .

Epoch 90/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0843 - val\_loss: 0.0905

Epoch 91/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0878 - val\_loss: 0.0879

Epoch 92/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0919 - val\_loss: 0.0963

Epoch 93/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0846 - val\_loss: 0.0879

Epoch 94/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0841 - val\_loss: 0.0883

Epoch 95/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0847 - val\_loss: 0.0888

Epoch 96/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0823 - val\_loss: 0.0925

Epoch 97/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0844 - val\_loss: 0.0931

Epoch 98/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0782 - val\_loss: 0.0919

Epoch 99/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0865 - val\_loss: 0.0969

Epoch 100/100

227/227 [==============================] - 4s 16ms/step - loss: 0.0826 - val\_loss: 0.0874

In [43]:

run\_time = (end\_time - start\_time)/60

run\_time

Out[43]:

6.1069254040718075

In [44]:

total\_loss = [j **for** i **in** total\_loss **for** j **in** i]

total\_val\_loss = [j **for** i **in** total\_val\_loss **for** j **in** i]

In [45]:

plot\_loss(total\_loss,total\_val\_loss)



Рис. График зависимости ошибки обучения от количества эпох

In [46]:

input\_seq\_test = x\_train[-60:].reshape((1,60,1))

output\_seq\_test = x\_test[:20]

decoder\_input\_test = np.zeros((1,output\_seq\_len,1))

In [47]:

pred = model1\_bi.predict([input\_seq\_test,decoder\_input\_test])

In [48]:

pred\_values = scaler.inverse\_transform(pred.reshape(-1,1))

output\_seq\_test = scaler.inverse\_transform(output\_seq\_test)

input\_seq\_test = scaler.inverse\_transform(input\_seq\_test.reshape(-1,1))

In [49]:

plt.plot(pred\_values, label = "pred")

plt.plot(output\_seq\_test, label = "actual")

plt.title("2-напр 1-слойный")

plt.ylabel("Bid", fontsize=12)

plt.xlabel("Time", fontsize=12)

plt.legend()

plt.savefig('bi\_dir1.png')

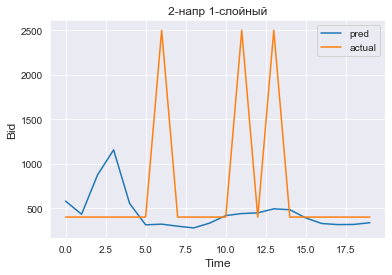


Рис. График прогнозируемой функции

In [50]:

*# serialize model to JSON*

*# model\_json = model\_1.to\_json()*

*# with open("model1\_bi.json", "w") as json\_file:*

*# json\_file.write(model\_json)*

*# # serialize weights to HDF5*

*# model\_1.save\_weights("model1\_bi.h5")*

*# print("Saved model to disk")*

1. **3. Однонаправленная 2-слойная модель**

In [63]:

model\_2 = create\_model([60,60],bidirectional=**False**)

In [64]:

*# from keras.models import model\_from\_json*

*# # load json and create model*

*# json\_file = open('model\_2.json', 'r')*

*# loaded\_model\_json = json\_file.read()*

*# json\_file.close()*

*# model\_2 = model\_from\_json(loaded\_model\_json)*

*# # load weights into new model*

*# model\_2.load\_weights("model\_2.h5")*

*# print("Loaded model from disk")*

In [65]:

total\_loss = []

total\_val\_loss = []

model\_2.compile(Adam(), loss = 'mean\_squared\_error')

In [66]:

start\_time = time.time()

run\_model(model\_2,batches=1, epochs=100, batch\_size=batch\_size)

end\_time = time.time()

Epoch 1/100

2/227 [..............................] - ETA: 52s - loss: 0.5620WARNING:tensorflow:Callbacks method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0340s vs `on\_train\_batch\_end` time: 0.4320s). Check your callbacks.

227/227 [==============================] - 5s 23ms/step - loss: 0.1694 - val\_loss: 0.1390

Epoch 2/100

227/227 [==============================] - 4s 19ms/step - loss: 0.1581 - val\_loss: 0.1368

Epoch 3/100

227/227 [==============================] - 4s 19ms/step - loss: 0.1535 - val\_loss: 0.1292

Epoch 4/100

227/227 [==============================] - 4s 19ms/step - loss: 0.1500 - val\_loss: 0.1307

Epoch 5/100

227/227 [==============================] - 4s 20ms/step - loss: 0.1490 - val\_loss: 0.1280

Epoch 6/100

227/227 [==============================] - 4s 19ms/step - loss: 0.1546 - val\_loss: 0.1370

Epoch 7/100

227/227 [==============================] - 4s 19ms/step - loss: 0.1501 - val\_loss: 0.1256

Epoch 8/100

227/227 [==============================] - 4s 19ms/step - loss: 0.1463 - val\_loss: 0.1241

Epoch 9/100

227/227 [==============================] - 4s 20ms/step - loss: 0.1439 - val\_loss: 0.1241

Epoch 10/100

227/227 [==============================] - 4s 20ms/step - loss: 0.1440 - val\_

. . . . . . . . .

Epoch 90/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0741 - val\_loss: 0.0731

Epoch 91/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0785 - val\_loss: 0.0746

Epoch 92/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0756 - val\_loss: 0.0719

Epoch 93/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0824 - val\_loss: 0.0745

Epoch 94/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0771 - val\_loss: 0.0778

Epoch 95/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0723 - val\_loss: 0.0679

Epoch 96/100

227/227 [==============================] - 4s 19ms/step - loss: 0.0737 - val\_loss: 0.0754

Epoch 97/100

227/227 [==============================] - 4s 20ms/step - loss: 0.0818 - val\_loss: 0.0823

Epoch 98/100

227/227 [==============================] - 5s 20ms/step - loss: 0.0705 - val\_loss: 0.0691

Epoch 99/100

227/227 [==============================] - 4s 20ms/step - loss: 0.0713 - val\_loss: 0.0709

Epoch 100/100

227/227 [==============================] - 4s 20ms/step - loss: 0.0707 - val\_loss: 0.0693

In [67]:

run\_time = (end\_time - start\_time)/60

run\_time

Out[67]:

7.560340174039205

In [68]:

total\_loss = [j **for** i **in** total\_loss **for** j **in** i]

total\_val\_loss = [j **for** i **in** total\_val\_loss **for** j **in** i]

In [69]:

plot\_loss(total\_loss,total\_val\_loss)

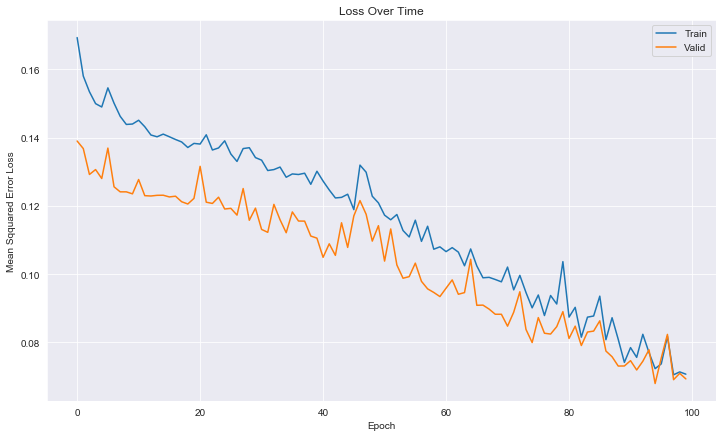


Рис. График зависимости ошибки обучения от количества эпох

In [70]:

input\_seq\_test = x\_train[-60:].reshape((1,60,1))

output\_seq\_test = x\_test[:20]

decoder\_input\_test = np.zeros((1,output\_seq\_len,1))

In [60]:

pred2 = model\_2.predict([input\_seq\_test,decoder\_input\_test])

In [61]:

pred\_values2 = scaler.inverse\_transform(pred2.reshape(-1,1))

output\_seq\_test2 = scaler.inverse\_transform(output\_seq\_test)

In [71]:

plt.plot(pred\_values2, label = "pred")

plt.plot(output\_seq\_test2, label = "actual")

plt.title("1-напр 2-слойн")

plt.ylabel("Bid", fontsize=12)

plt.xlabel("Time", fontsize=12)

plt.legend()

plt.savefig('uni\_dir2.png')

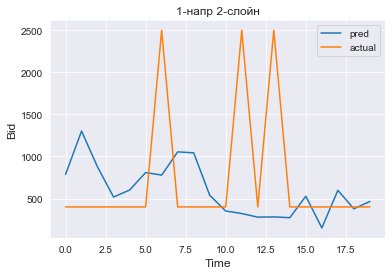


Рис. График прогнозируемой функции

In [72]:

*# # serialize model to JSON*

*# model\_json = model\_2.to\_json()*

*# with open("model\_2.json", "w") as json\_file:*

*# json\_file.write(model\_json)*

*# # serialize weights to HDF5*

*# model\_2.save\_weights("model\_2.h5")*

*# print("Saved model to disk")*

1. **4. Двунаправленная 2-слойная модель**

In [73]:

model2\_bi = create\_model([60,60],bidirectional=**True**)

In [74]:

*# загрузка джейсона*

**from** **keras.models** **import** model\_from\_json

*# json\_file = open('model2\_bi.json', 'r')*

*# loaded\_model\_json = json\_file.read()*

*# json\_file.close()*

*# model2\_bi = model\_from\_json(loaded\_model\_json)*

*# # load weights into new model*

*# model2\_bi.load\_weights("model2\_bi.h5")*

*# print("Loaded model from disk")*

In [75]:

total\_loss = []

total\_val\_loss = []

model2\_bi.compile(Adam(), loss = 'mean\_squared\_error')

In [77]:

start\_time = time.time()

run\_model(model2\_bi,batches=1, epochs=100, batch\_size=batch\_size)

end\_time = time.time()

Epoch 1/100

2/227 [..............................] - ETA: 16s - loss: 0.1087WARNING:tensorflow:Callbacks method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0270s vs `on\_train\_batch\_end` time: 0.1150s). Check your callbacks.

227/227 [==============================] - 6s 28ms/step - loss: 0.1490 - val\_loss: 0.1306

Epoch 2/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1470 - val\_loss: 0.1307

Epoch 3/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1452 - val\_loss: 0.1300

Epoch 4/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1433 - val\_loss: 0.1286

Epoch 5/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1418 - val\_loss: 0.1279

Epoch 6/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1434 - val\_loss: 0.1291

Epoch 7/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1399 - val\_loss: 0.1248

Epoch 8/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1393 - val\_loss: 0.1291

Epoch 9/100

227/227 [==============================] - 6s 27ms/step - loss: 0.1381 - val\_loss: 0.1241

Epoch 10/100

Epoch 90/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0382 - val\_

. . . . . . . .

loss: 0.0470

Epoch 91/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0315 - val\_loss: 0.0331

Epoch 92/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0315 - val\_loss: 0.0481

Epoch 93/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0330 - val\_loss: 0.0346

Epoch 94/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0393 - val\_loss: 0.0380

Epoch 95/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0309 - val\_loss: 0.0336

Epoch 96/100

227/227 [==============================] - 6s 26ms/step - loss: 0.0301 - val\_loss: 0.0399

Epoch 97/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0283 - val\_loss: 0.0364

Epoch 98/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0292 - val\_loss: 0.0366

Epoch 99/100

227/227 [==============================] - 6s 26ms/step - loss: 0.0292 - val\_loss: 0.0398

Epoch 100/100

227/227 [==============================] - 6s 27ms/step - loss: 0.0349 - val\_loss: 0.0372

In [78]:

run\_time = (end\_time - start\_time)/60

run\_time

Out[78]:

10.222440799077352

In [79]:

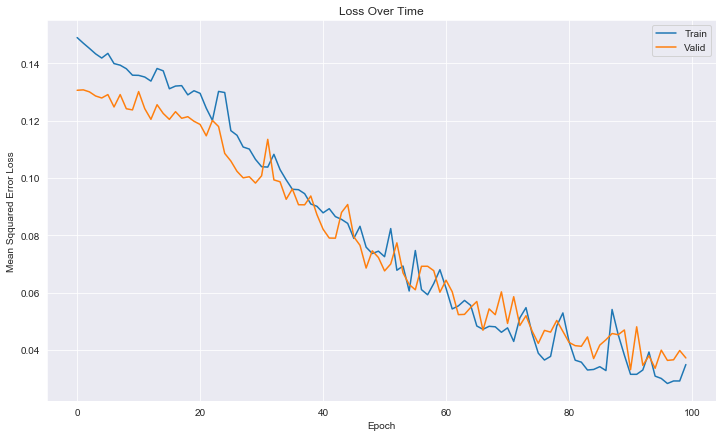
total\_loss = [j **for** i **in** total\_loss **for** j **in** i]

total\_val\_loss = [j **for** i **in** total\_val\_loss **for** j **in** i]

In [80]:

plot\_loss(total\_loss,total\_val\_loss)

plt.savefig('bi\_los2.png')



<Figure size 432x288 with 0 Axes>

Рис. График зависимости ошибки обучения от количества эпох

In [81]:

input\_seq\_test = x\_train[-60:].reshape((1,60,1))

output\_seq\_test = x\_test[:20]

decoder\_input\_test = np.zeros((1,output\_seq\_len,1))

In [82]:

pred2\_bi = model2\_bi.predict([input\_seq\_test,decoder\_input\_test])

In [83]:

pred\_values2\_bi = scaler.inverse\_transform(pred2\_bi.reshape(-1,1))

output\_seq\_test2\_bi = scaler.inverse\_transform(output\_seq\_test)

In [84]:

pred2\_bi = model2\_bi.predict([input\_seq\_test,decoder\_input\_test])

In [85]:

plt.plot(pred\_values2\_bi, label = "pred")

plt.plot(output\_seq\_test2\_bi, label = "actual")

plt.title("2-напр 2-слойн")

plt.ylabel("Bid", fontsize=12)

plt.xlabel("Time", fontsize=12)

plt.legend()

plt.savefig('bi\_dir2.png')

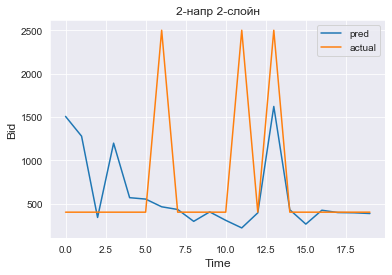


Рис. График прогнозируемой функции

In [86]:

*# serialize model to JSON*

*# model\_json = model2\_bi.to\_json()*

*# with open("model2\_bi.json", "w") as json\_file:*

*# json\_file.write(model\_json)*

*# # serialize weights to HDF5*

*# model2\_bi.save\_weights("model2\_bi.h5")*

*# print("Saved model to disk")*

1. **Прогнозирование на основе скользящего среднего (справочно)**

In [4]:

*#import numpy as np*

*#series = data['Bid']*

*#series=np.log(series)*

In [88]:

**def** moving\_average(series, n):

*"""*

*Вычислить среднее за последние n наблюдений*

*Calculate average of last n observations*

*"""*

**return** np.average(series[-n:])

a=scaler.inverse\_transform(moving\_average(X, 500000).reshape(-1,1))

a

Out[88]:

array([[870.3]])

In [89]:

*#893.16-*

**Вывод**

Из приведенных моделей наиболее адекватные результаты дает последняя из рассмотренных моделей. Эту модель предлагается использовать для дальнейшей реализации.