**Московский государственный технический университет им. Н.Э. Баумана**

Кафедра «Системы обработки информации и управления»

**Лабораторная работа №6**

**по курсу «Проектирование интеллектуальных систем»**

**Выполнил:**

**Хотин П.Ю.**

**ИУ5-24М**

Москва, 2020 год

**Задание**

Требуется собрать рекуррентную нейронную сеть и обеспечить её работу на 3 наборах данных:

1. 2 связанных временных ряда (двумерный временной ряд).
2. Сигнал, являющийся суперпозицией двух синусоид с разным периодом и смещением.
3. Аналогичный сигнал с наложенным шумом.

**Импорт библиотек**

from IPython.display import set\_matplotlib\_formats  
import matplotlib.pyplot as plt  
import math  
import numpy as np  
import random  
import tensorflow as tf  
from tqdm import trange

tf.\_\_version\_\_

'1.15.0'

%matplotlib inline  
set\_matplotlib\_formats("retina")

**Детализация графиков**

**def** gr(X, Y, outputs, batch\_size):  
 output\_dim = X.shape[2]  
 **for** j in range(batch\_size):  
 plt.figure(figsize=(12, 3))  
   
 **for** k in range(output\_dim):  
 past = X[:,j,k]  
 expected = Y[:,j,k]  
 pred = outputs[:,j,k]  
  
 label1 = "Seen (past) values" **if** k==0 **else** "\_nolegend\_"  
 label2 = "True future values" **if** k==0 **else** "\_nolegend\_"  
 label3 = "Predictions" **if** k==0 **else** "\_nolegend\_"  
  
 plt.plot(range(len(past)), past , "o--b", label=label1)  
 plt.plot(range(len(past), len(expected)+len(past)),  
 expected, "x--b", label=label2)  
 plt.plot(range(len(past), len(pred)+len(past)),  
 pred, "o--y", label=label3)  
  
 plt.legend(loc='best')  
 plt.title("Predictions vs true values")  
 plt.show()

**Генерация данных**

Напишем функции генерации данных

**Упражнение 1**

Функция генерации 2 связанных временных рядов

**def** generate\_x\_y\_data\_v1(isTrain, batch\_size):  
 seq\_length = 10  
  
 batch\_x = []  
 batch\_y = []  
 **for** \_ in range(batch\_size):   
 *# Одна итерация цикла генерирует 1 пакет данных*   
 rand = random.random() \* 2 \* math.pi  
   
 *# Генерируем набор данных по заданному закону*  
 *# np . linspace*   
 *# генерирует набор точек, равномерно распредленных*  
 *# по заданному интервалу*   
 *# (границы интервала смещены на случайную величину)*  
 sig1 = np.sin(np.linspace(0.0 \* math.pi + rand,  
 3.0 \* math.pi + rand, seq\_length \* 2))  
 sig2 = np.cos(np.linspace(0.0 \* math.pi + rand,  
 3.0 \* math.pi + rand, seq\_length \* 2))  
   
 *# первую половину сигналов берем на обучающую выборку,*  
 *# вторую - на контрольную*  
 x1 = sig1[:seq\_length]  
 y1 = sig1[seq\_length:]  
 x2 = sig2[:seq\_length]  
 y2 = sig2[seq\_length:]  
   
 x\_ = np.array([x1, x2])  
 y\_ = np.array([y1, y2])  
 x\_, y\_ = x\_.T, y\_.T  
   
 batch\_x.append(x\_)  
 batch\_y.append(y\_)  
   
 batch\_x = np.array(batch\_x)  
 batch\_y = np.array(batch\_y)   
 *# размерность: (batch\_size , seq\_length , output\_dim)*  
 *# транспонируем, чтобы привести к нужной размерности*  
 batch\_x = np.array(batch\_x).transpose((1, 0, 2))  
 batch\_y = np.array(batch\_y).transpose((1, 0, 2))   
 *# размерность: (seq\_length , batch\_size , output\_dim)*  
   
 **return** batch\_x , batch\_y

**Упражнение 2**

Генерация данных с разными частотами

**def** generate\_x\_y\_data\_two\_freqs(isTrain, batch\_size, seq\_length):  
 batch\_x = []   
 batch\_y = []  
  
 **for** \_ in range(batch\_size):  
 offset\_rand = random.random() \* 2 \* math.pi  
 freq\_rand = (random.random() - 0.5) / 1.5 \* 15 + 0.5  
 amp\_rand = random.random() + 0.1  
   
 sig1 = amp\_rand \* np.sin(np.linspace(  
 seq\_length / 15.0 \* freq\_rand \* 0.0 \* math.pi + offset\_rand,  
 seq\_length / 15.0 \* freq\_rand \* 3.0 \* math.pi + offset\_rand,  
 seq\_length \* 2)  
 )  
   
 offset\_rand = random.random() \* 2 \* math.pi  
 freq\_rand = (random.random() - 0.5) / 1.5 \* 15 + 0.5  
 amp\_rand = random.random() \* 1.2  
 sig1 = amp\_rand \* np.cos(np.linspace(  
 seq\_length / 15.0 \* freq\_rand \* 0.0 \* math.pi + offset\_rand,  
 seq\_length / 15.0 \* freq\_rand \* 3.0 \* math.pi + offset\_rand,  
 seq\_length \* 2)  
 ) + sig1  
   
 x1 = sig1[:seq\_length]  
 y1 = sig1[seq\_length:]  
 x\_ = np.array([x1])  
 y\_ = np.array([y1])  
 x\_, y\_ = x\_.T, y\_.T  
   
 batch\_x.append(x\_)  
 batch\_y.append(y\_)  
   
 batch\_x = np.array(batch\_x)  
 batch\_y = np.array(batch\_y)  
 *# размерность: (batch\_size , seq\_length , output\_dim)*  
 batch\_x = np.array(batch\_x).transpose((1, 0, 2))  
 batch\_y = np.array(batch\_y).transpose((1, 0, 2))  
 *# размерность: (seq\_length , batch\_size , output\_dim)*  
  
 **return** batch\_x , batch\_y

**def** generate\_x\_y\_data\_v2(isTrain, batch\_size):  
 **return** generate\_x\_y\_data\_two\_freqs(isTrain, batch\_size, seq\_length=15)

**Упражнение 3**

Генерация данных с наложенным шумом

**def** generate\_x\_y\_data\_v3(isTrain, batch\_size):  
 seq\_length = 30  
 x, y = generate\_x\_y\_data\_two\_freqs(  
 isTrain, batch\_size, seq\_length=seq\_length)  
 noise\_amount = random.random() \* 0.15 + 0.10  
 x = x + noise\_amount \* np.random.randn(seq\_length, batch\_size, 1)  
  
 avg = np.average(x)  
 std = np.std(x) + 0.0001  
 x = x - avg  
 y = y - avg  
 x = x / std / 2.5  
 y = y / std / 2.5  
  
 **return** x, y

**Описание модели**

Создание сессии

sess = tf.InteractiveSession()

**Структура модели и граф вычислений**

**def** create\_model(input\_len, input\_dim=1,  
 output\_len=None, output\_dim=None,  
 hidden\_dim=12, layers\_stacked\_count=2,  
 learning\_rate=0.006, lr\_decay=0.91,  
 momentum=0.5, lambda\_l2\_reg=0.003):  
 **if** output\_len is None:  
 output\_len = input\_len  
 **if** output\_dim is None:  
 output\_dim = input\_dim  
   
 print(locals())  
   
 g = tf.Graph()  
 **with** g.as\_default():  
 **with** tf.variable\_scope('Seq2seq'):  
 enc\_inp = [  
 tf.placeholder(tf.float32, shape=(None, input\_dim),  
 name="inp\_{}".format(t))  
 **for** t in range(input\_len)  
 ]  
  
 expected\_sparse\_output = [  
 tf.placeholder(tf.float32 , shape=(None, output\_dim),  
 name="expected\_sparse\_output\_".format(t))  
 **for** t in range(output\_len)  
 ]  
  
 dec\_inp = [tf.zeros\_like(enc\_inp[0], dtype=np.float32,  
 name="GO")] + enc\_inp[:-1]  
  
 cells = []  
 **for** i in range(layers\_stacked\_count):  
 **with** tf.variable\_scope('RNN\_{}'.format(i)):  
 cells.append(tf.contrib.rnn.GRUCell(hidden\_dim))  
 cell = tf.contrib.rnn.MultiRNNCell(cells)  
  
 w\_in = tf.Variable(tf.random\_normal([input\_dim, hidden\_dim]))  
 b\_in = tf.Variable(tf.random\_normal([hidden\_dim], mean=1.0))  
 w\_out = tf.Variable(tf.random\_normal([hidden\_dim , output\_dim]))  
 b\_out = tf.Variable(tf.random\_normal([output\_dim]))  
  
 reshaped\_inputs = [tf.nn.relu(tf.matmul(i, w\_in) + b\_in)  
 **for** i in enc\_inp]  
  
 seq2seq = tf.contrib.legacy\_seq2seq.basic\_rnn\_seq2seq  
 dec\_outputs, dec\_memory = seq2seq(enc\_inp, dec\_inp, cell)  
  
 output\_scale\_factor = tf.Variable(1.0, name="Output\_ScaleFactor")  
 reshaped\_outputs = [output\_scale\_factor \* (tf.matmul(i, w\_out)  
 + b\_out) **for** i in dec\_outputs]  
  
 **with** tf.variable\_scope('Loss'):  
 *# L2 loss*  
 output\_loss = 0  
 **for** \_y, \_Y in zip(reshaped\_outputs, expected\_sparse\_output):  
 output\_loss += tf.reduce\_mean(tf.nn.l2\_loss(\_y - \_Y))  
  
 *# слагаемое, отвечающее за L2 регуляризацию*  
 reg\_loss = 0  
 **for** tf\_var in tf.trainable\_variables():  
 **if** not ("Bias" in tf\_var.name or "Output\_" in tf\_var.name):  
 reg\_loss += tf.reduce\_mean(tf.nn.l2\_loss(tf\_var))  
  
 loss = output\_loss + lambda\_l2\_reg \* reg\_loss  
  
 **with** tf.variable\_scope('Optimizer'):  
 optimizer = tf.train.RMSPropOptimizer(learning\_rate,  
 decay=lr\_decay,  
 momentum=momentum)  
 train\_op = optimizer.minimize(loss)  
 **return** (g, enc\_inp, expected\_sparse\_output, train\_op, loss,  
 reshaped\_outputs)

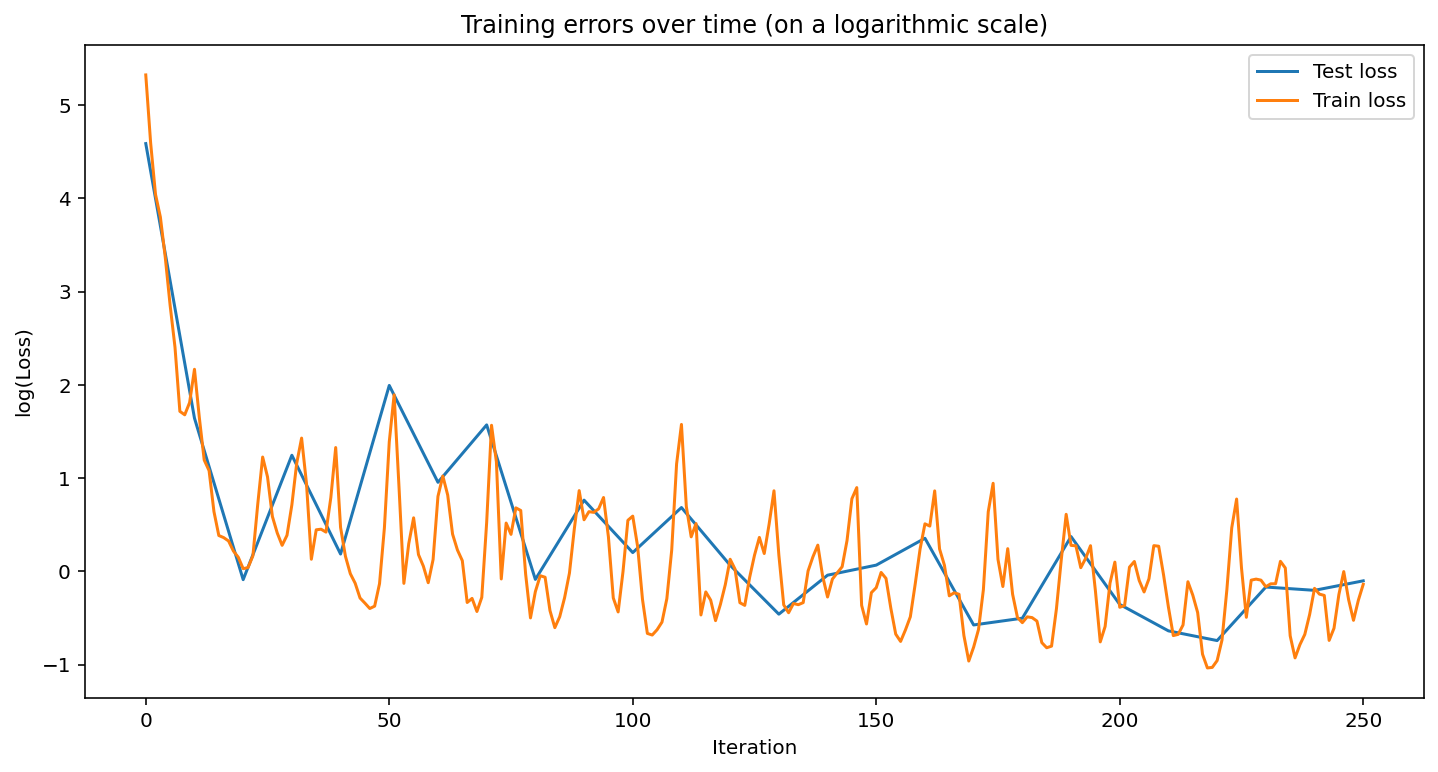
**Запуск модели с заданными дефолтными значениями в параметрах**

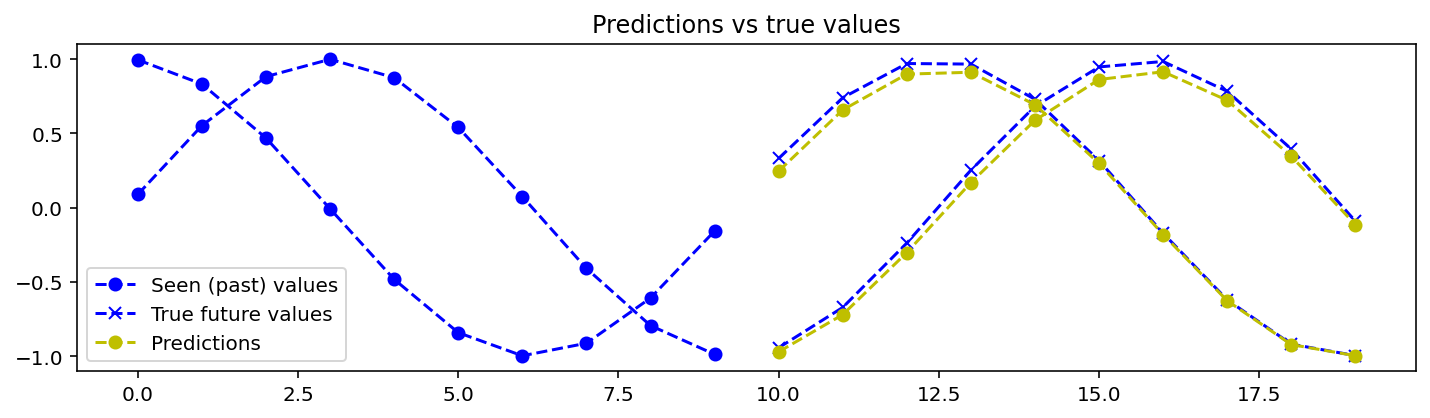
**def** processing\_model(model, generate\_X\_Y, batch\_size=32,  
 nb\_iters=150, nb\_predictions=1):  
 **if** nb\_iters % 10 != 0:  
 **raise** ValueError('nb\_iters does not divide on 10')  
   
 g, enc\_inp, expected\_sparse\_output, train\_op, loss, reshaped\_outputs = model  
   
 **def** train\_batch(batch\_size):  
 X, Y = generate\_X\_Y(isTrain=True, batch\_size=batch\_size)  
 feed\_dict = {enc\_inp[t]: X[t] **for** t in range(len(enc\_inp))}   
 feed\_dict.update({expected\_sparse\_output[t]: Y[t] **for** t in range(len(expected\_sparse\_output))})  
  
 \_, loss\_t = sess.run([train\_op, loss], feed\_dict)  
 **return** loss\_t  
  
 **def** test\_batch(batch\_size):  
 X, Y = generate\_X\_Y(isTrain=False, batch\_size=batch\_size)  
 feed\_dict = {enc\_inp[t]: X[t] **for** t in range(len(enc\_inp))}   
 feed\_dict.update({expected\_sparse\_output[t]: Y[t] **for** t in range(len(expected\_sparse\_output))})  
  
 loss\_t = sess.run(loss, feed\_dict)  
 **return** loss\_t  
  
 train\_losses = []  
 test\_losses = []  
  
 **with** g.as\_default():  
 **with** tf.Session() as sess:  
 sess.run(tf.global\_variables\_initializer())  
  
 test\_loss = -1  
 **with** trange(nb\_iters+1, ncols=70) as tq:  
 **for** t in tq:  
 train\_loss = train\_batch(batch\_size)  
 train\_losses.append(train\_loss)  
 tq.set\_postfix(train=train\_loss, test=test\_loss)  
  
 **if** t % 10 == 0:  
 test\_loss = test\_batch(batch\_size)  
 test\_losses.append(test\_loss)  
 tq.set\_postfix(train=train\_loss, test=test\_loss)  
  
 *# Show loss graph*  
 plt.figure(figsize=(12, 6))  
 plt.plot(  
 (np.array(range(0, len(test\_losses))) /   
 float(len(test\_losses)-1) \* (len(train\_losses)-1)),  
 np.log(test\_losses),  
 label="Test loss"  
 )  
 plt.plot(  
 np.log(train\_losses),  
 label="Train loss"  
 )  
 plt.title("Training errors over time (on a logarithmic scale)")  
 plt.xlabel('Iteration')  
 plt.ylabel('log(Loss)')  
 plt.legend(loc='best')  
 plt.show()  
   
 *# Show sample graphs*  
 X, Y = generate\_X\_Y(isTrain=False, batch\_size=nb\_predictions)  
 feed\_dict = {enc\_inp[t]: X[t] **for** t in range(len(enc\_inp))}  
 outputs = np.array(sess.run(reshaped\_outputs, feed\_dict))  
 gr(X, Y, outputs, nb\_predictions)

**Тренировка**

**Выполнение упражнения 1**

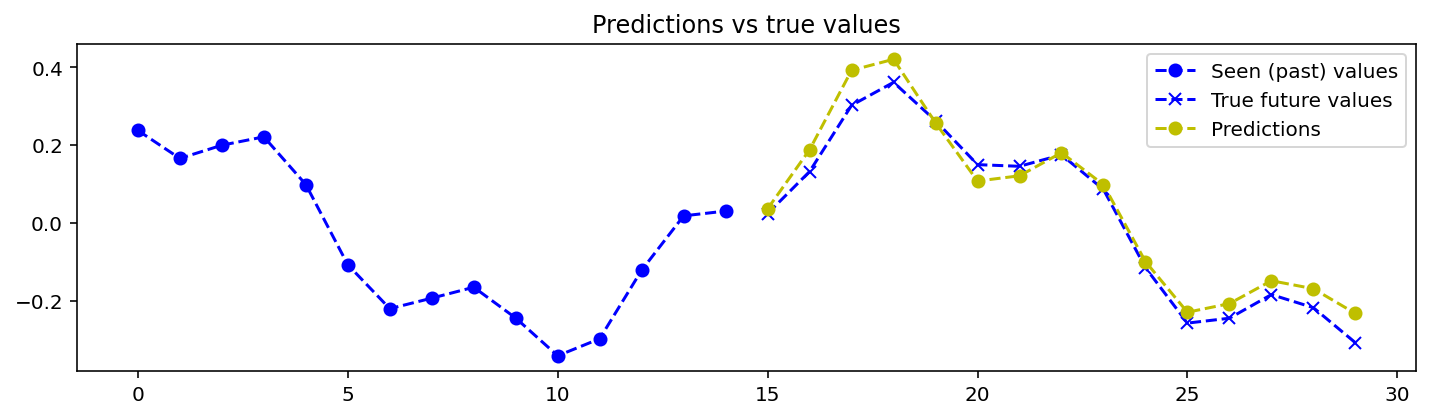
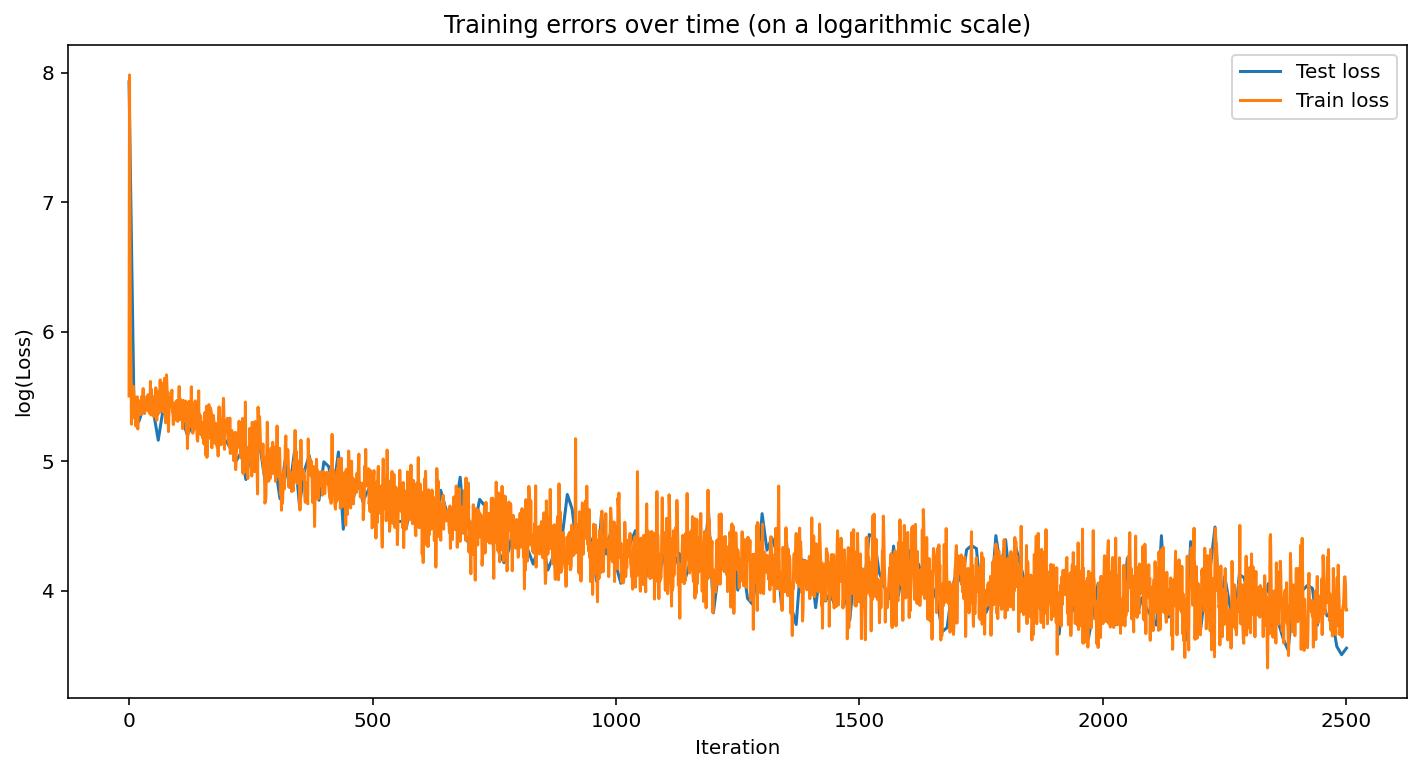
generate\_X\_Y = generate\_x\_y\_data\_v1  
x, \_ = generate\_X\_Y(True, 1)  
model = create\_model(x.shape[0], x.shape[2])  
processing\_model(model, generate\_X\_Y, nb\_iters=250)

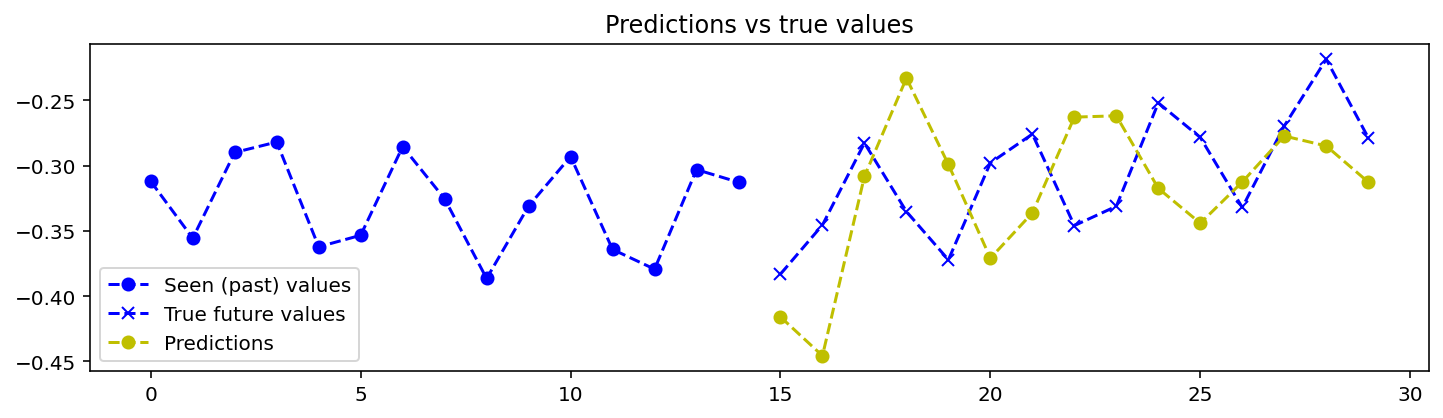
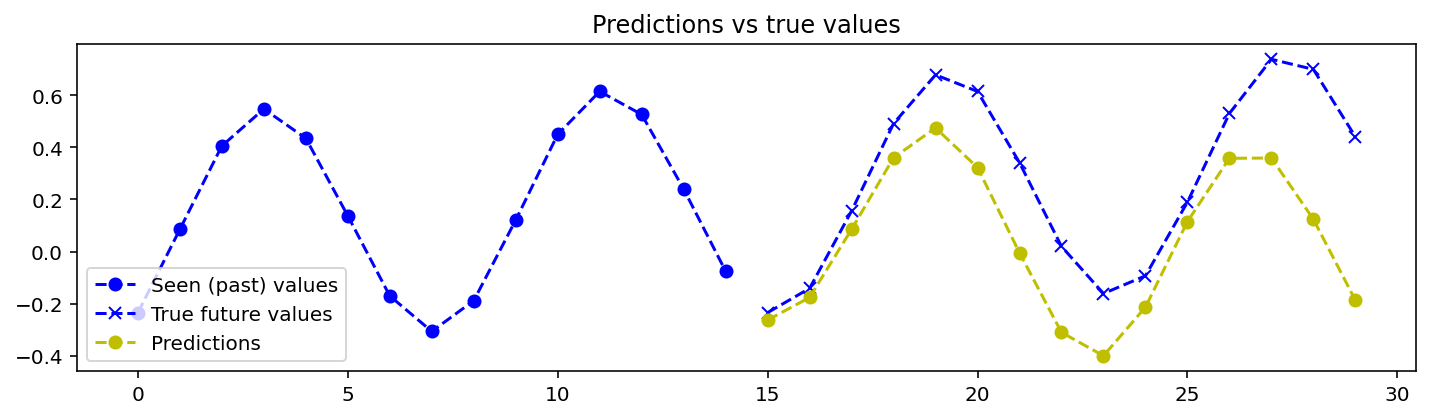
{'input\_len': 10, 'output\_len': 10, 'hidden\_dim': 12, 'layers\_stacked\_count': 2, 'learning\_rate': 0.006, 'lr\_decay': 0.91, 'momentum': 0.5, 'lambda\_l2\_reg': 0.003, 'input\_dim': 2, 'output\_dim': 2}  
  
  
100%|███████| 251/251 [00:04<00:00, 54.71it/s, test=0.903, train=0.87]



**Выполнение упражнения 2**

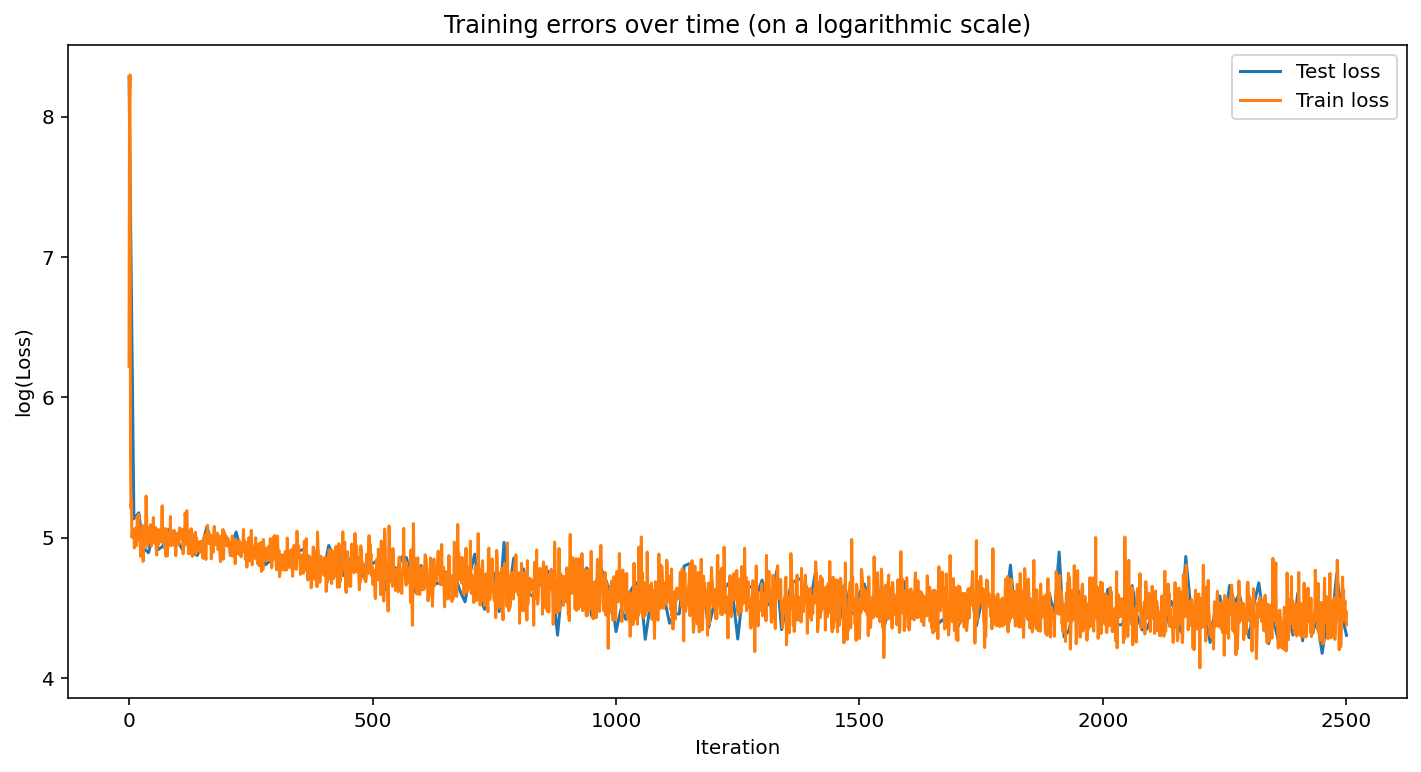
generate\_X\_Y = generate\_x\_y\_data\_v2  
x, \_ = generate\_X\_Y(True, 1)  
model = create\_model(x.shape[0], x.shape[2],  
 hidden\_dim=50)  
processing\_model(model, generate\_X\_Y, batch\_size=64,  
 nb\_iters=2500, nb\_predictions=3)

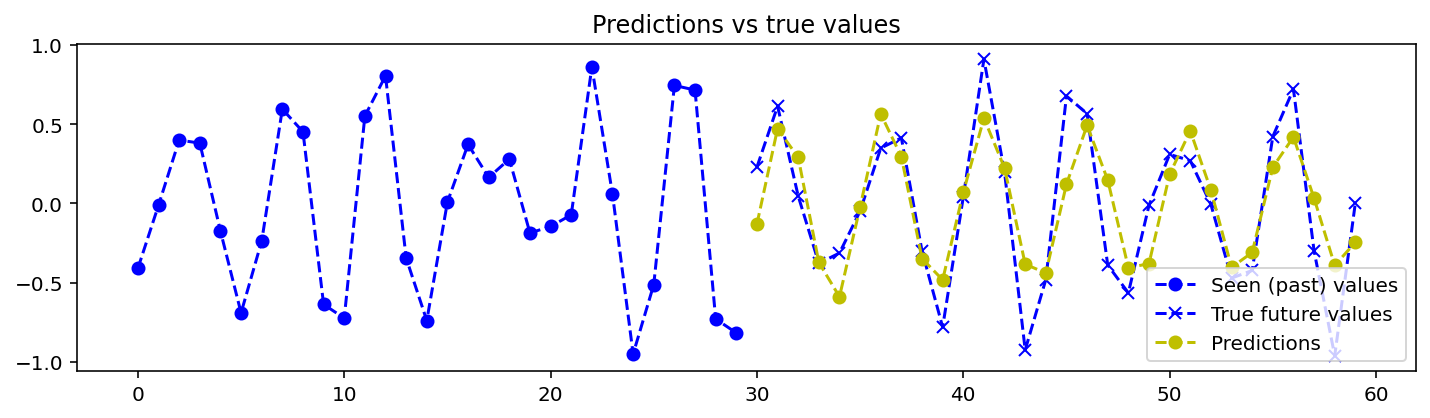
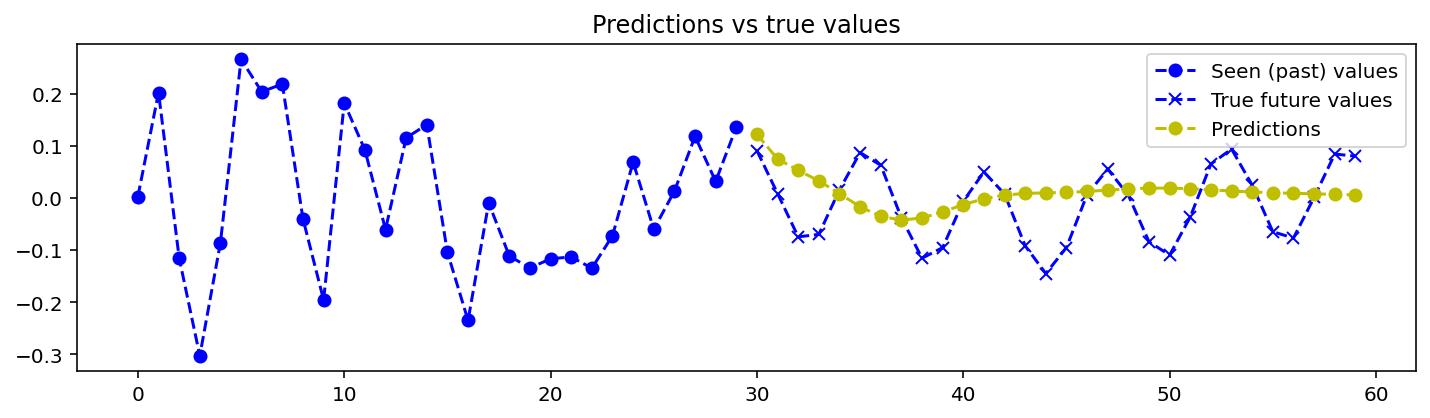
{'input\_len': 15, 'output\_len': 15, 'hidden\_dim': 50, 'layers\_stacked\_count': 2, 'learning\_rate': 0.006, 'lr\_decay': 0.91, 'momentum': 0.5, 'lambda\_l2\_reg': 0.003, 'input\_dim': 1, 'output\_dim': 1}  
  
  
100%|██████| 2501/2501 [00:53<00:00, 46.97it/s, test=35.1, train=47.1]

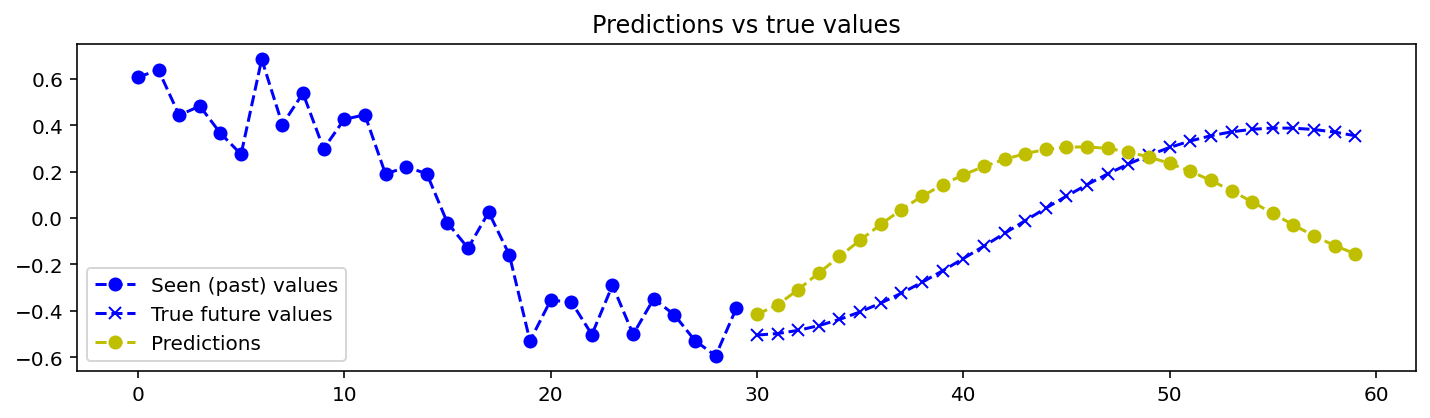


**Выполнение упражнения 3**

generate\_X\_Y = generate\_x\_y\_data\_v3  
x, \_ = generate\_X\_Y(True, 1)  
model = create\_model(x.shape[0], x.shape[2],  
 hidden\_dim=50)  
processing\_model(model, generate\_X\_Y, batch\_size=64,  
 nb\_iters=2500, nb\_predictions=3)

{'input\_len': 30, 'output\_len': 30, 'hidden\_dim': 50, 'layers\_stacked\_count': 2, 'learning\_rate': 0.006, 'lr\_decay': 0.91, 'momentum': 0.5, 'lambda\_l2\_reg': 0.003, 'input\_dim': 1, 'output\_dim': 1}  
  
  
100%|████████| 2501/2501 [02:04<00:00, 20.05it/s, test=74, train=79.8]





**Ответы на вопросы.**

1. **В чем преимущество рекуррентных нейронных сетей перед обычными персептронами?**  
   Рекуррентные нейронные сети содержат функционал цикличности, засчет этого они могут анализировать целую последовательность токенов, а не единичные отдельные экземпляры.
2. **Что такое регуляризация и зачем она?**  
   L1- и L2-регуляризация — эта два тесно связанных метода, которые можно применять в алгоритмах машинного обучения (machine learning, ML) для уменьшения степени переобучения модели (model overfitting).
3. **Что такое пакетный, мини-пакетный, онлайновый градиентный спуск.**

В пакетном градиентном спуске параметры веса обновляются один раз за эпоху.

В стохастическом градиентном спуске значения будут колебаться возле минимума, но никогда не достигнут его.   
Что касается мини-пакетного - выборка разбивается на более мелкие группы, по которым рассчитываются ошибки и обновляются веса.

**Список литературы**

[1] Google. Tensorflow. 2018. Apr. url - <https://www.tensorflow.org/api_docs/python/tf/train/Saver>.

[2] Google. TensorBoard. 2018. Apr. url - https://www.tensorflow.org/programmers\_guide/summaries\_and\_tensorboard.