Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Лабораторная работа №6 по дисциплине**

«Проектирование интеллектуальных систем»

**ИСПОЛНИТЕЛЬ:**

Чечнев А.А.

Группа ИУ5-23М

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Москва 2020

1. **Задание**

**Цель работы:** научить работать с рекуррентными нейронными сетями в tensorflow. В материалах рассмотрены создание модели рекуррентной нейронной сети, её компиляция и обучение. Описано применение полученной модели для прогнозирования временного ряда. Приведены методы генерации тестовых наборов данных. В заключительной части рассмотрена визуализация полученных результатов.

**Упражнение 1**. Два временных ряда, которые связанны вместе (двумерный временной ряд).

**Упражнение 2**. Один сигнал, являющийся суперпозицией двух синусоид

**Упражнение 3**. Один сигнал, являющийся суперпозицией двух синусоид с шумами.

1. **Решение**
   1. **Упражнение 1**

**Генерация входных данных**

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

        # 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

**Значения гиперпараметров**

# Данные имеют размерность (seq\_length , batch\_size , output\_dim)

sample\_x , sample\_y = generate\_x\_y\_data\_v1(isTrain=True , batch\_size=3)

# Длина последовательности (в данных примерах одинаковая для обучающих и тестовых данных)

seq\_length = sample\_x.shape[0]

# Размер пакета количество(тестовых примеров), по которому усредняется градиент

batch\_size = 40

# Размерность выходных данных

output\_dim = input\_dim = sample\_x.shape[-1]

# Количество скрытых нейронов в каждой ячейке

hidden\_dim = 20

# Количество ячеек рекуррентной сети (в глубину)

layers\_stacked\_count = 2

# Параметры оптимизатора

# Скорость обучения маленькая (скорость обучения позволяет алгоритму не расходиться во время обучения)

learning\_rate = 0.001

# Количество итераций по обучающей выборке

nb\_iters = 1500

# Дополнительные параметры алгоритма оптимизации

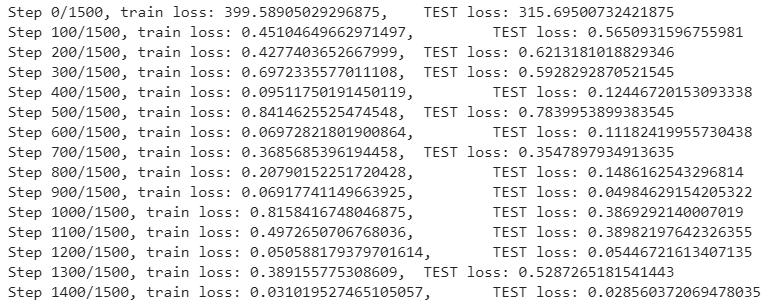
lr\_decay = 0.95

momentum = 0.1

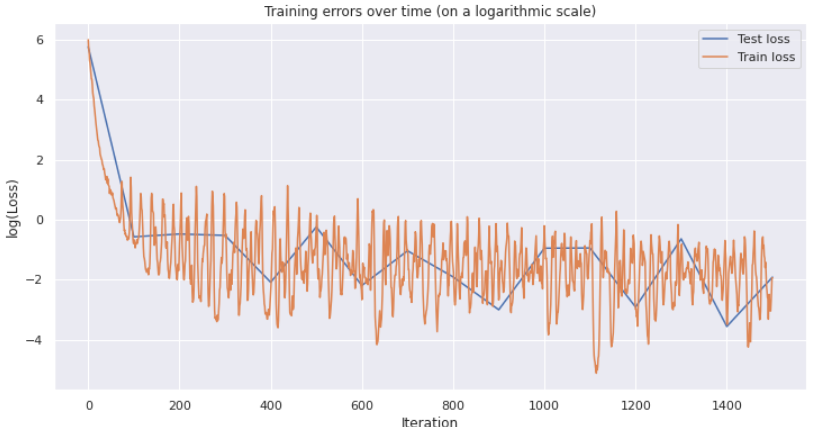
# Коэффициент L2 регуляризации

lambda\_l2\_reg = 1e-14

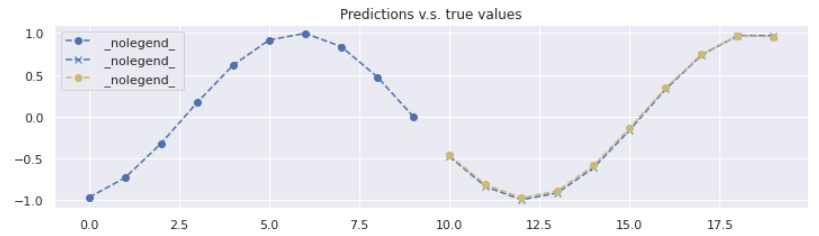
**Значения функции потери во время обучения**



**Ошибка во время обучения**



**Полученные предсказания**



* 1. **Упражнение 2**

**Генерация входных данных**

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

**Значения гиперпараметров**

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)

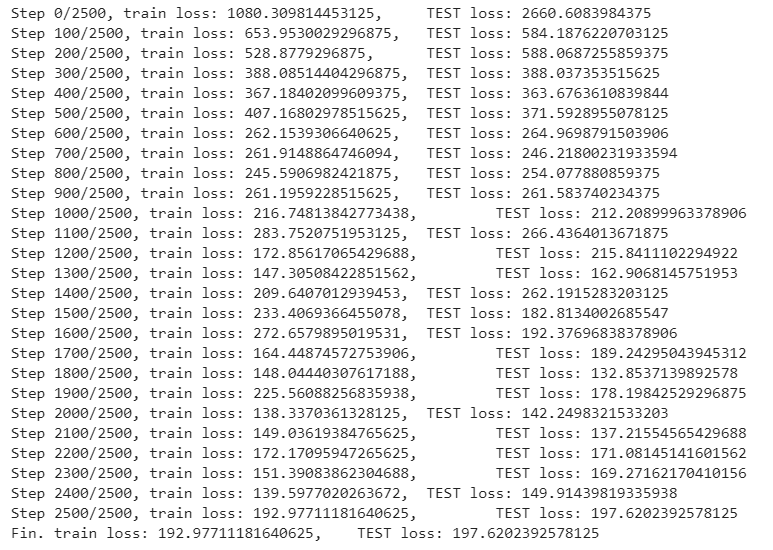
    batch\_x = np.array(batch\_x).transpose((1, 0, 2))

    batch\_y = np.array(batch\_y).transpose((1, 0, 2))

    # размерность: (seq\_length , batch\_size , output\_dim)

    return batch\_x , batch\_y

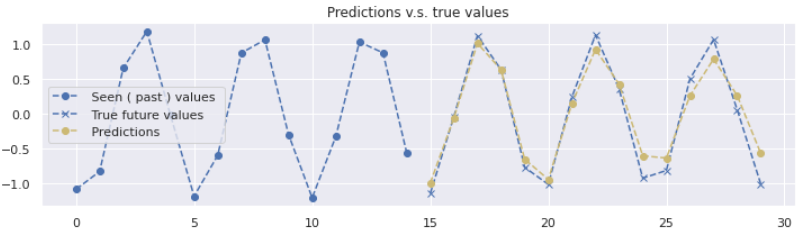
**Значения функции потери во время обучения**



**Ошибка во время обучения**



**Полученные предсказания**



* 1. **Упражнение 3**

**Генерация входных данных**

**#Предварительно запустить функцию из упражнения 2**

def generate\_x\_y\_data(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

**Значения гиперпараметров**

# Данные имеют размерность (seq\_length , batch\_size , output\_dim)

sample\_x , sample\_y = generate\_x\_y\_data(isTrain=True , batch\_size=3)

# Длина последовательности (в данных примерах одинаковая для обучающих и тестовых данных)

seq\_length = sample\_x.shape[0]

# Размер пакета количество(тестовых примеров), по которому усредняется градиент

batch\_size = 100

# Размерность выходных данных

output\_dim = input\_dim = sample\_x.shape[-1]

# Количество скрытых нейронов в каждой ячейке

hidden\_dim = 50

# Количество ячеек рекуррентной сети (в глубину)

layers\_stacked\_count = 1

# Параметры оптимизатора

# Скорость обучения маленькая (скорость обучения позволяет алгоритму не расходиться во время обучения)

learning\_rate = 0.01

# Количество итераций по обучающей выборке

nb\_iters = 2000

# Дополнительные параметры алгоритма оптимизации

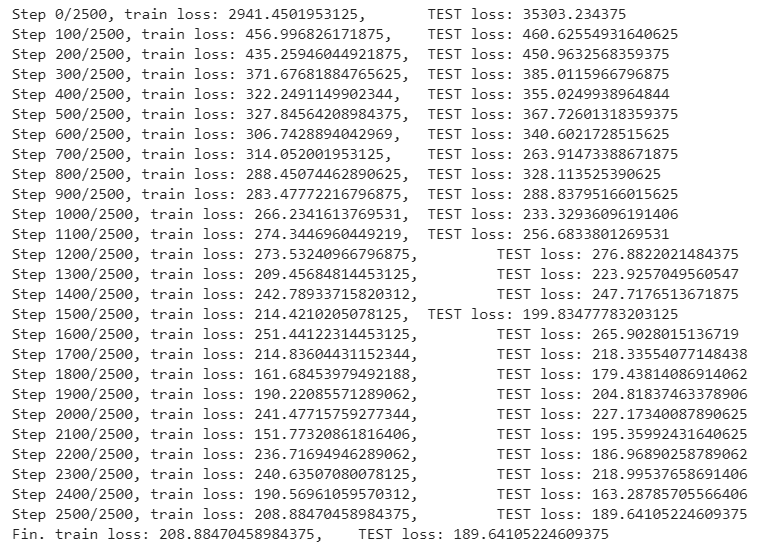
lr\_decay = 0.91

momentum = 0.3

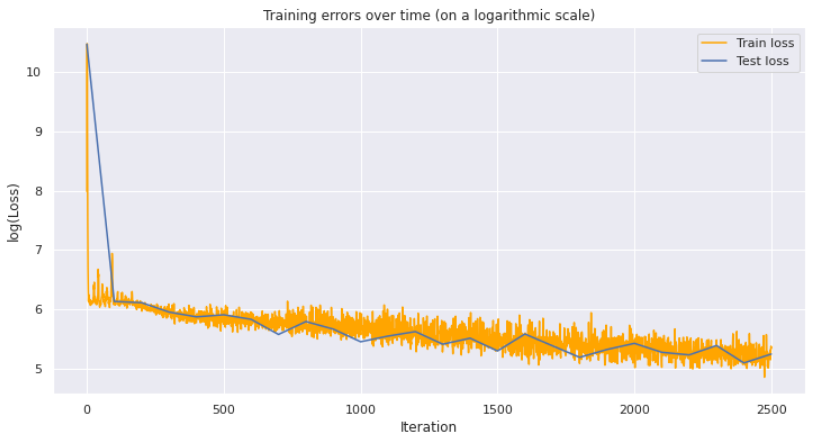
# Коэффициент L2 регуляризации

lambda\_l2\_reg = 1e-13

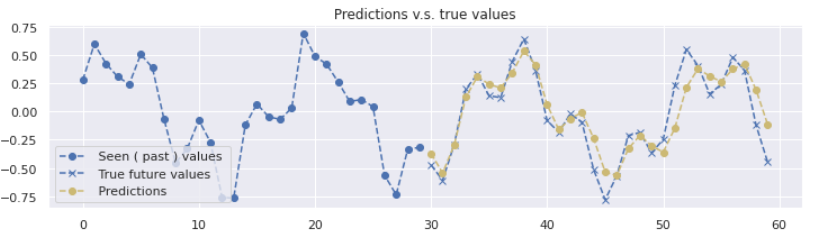
**Значения функции потери во время обучения**



**Ошибка во время обучения**



**Полученные предсказания**



**Контрольные вопросы**

1. **В чем преимущество рекурентных нейронных сетей по сравнению с обычными персептронами?**

В отличие от полносвязных сетей, реккурентные нейронные связаны с предыдущими итерациями обучения. Каждый выход нейросети является входом для одного и более следующих нейросетей. Таким образом при функционировании учитываются значения предыдущей итерации, тем самым 'запоминая' предыдущее состояние. Показывают себя хорошо для временных рядов.

1. **Что такое регуляризация и зачем она нужна?**

Регуляризация — метод добавления ограничений на сложность решения, путем добавления штрафа к весам в процессе обучения. Помогает избавиться или как минимум уменьшить вероятность явления переобучения.

1. **Что такое пакетный, мини-пакетный и онлайновый градиентный спуск?**

Пакетный градиентный спуск —вычисляет ошибку для каждого примера тренировочного датасета и обновляет модель после оценки всех примеров.

Мини-пакетный градиентный спуск — разбивает датасет на подвыборки (батчи) и для каждого считает ошибку и применяет изменение модели

Онлайновый градиентный спуск — разновидность алгоритма градиентного спуска, также называемая стохастическим градиентным спуском. СГС вычисляет ошибку и обновляет модель для каждого примера в наборе обучающих данных.

# **Список литературы**

1. Терехов В.И., Черненький И.М., Методические указания к лабораторной работе №2 - М, 2020
2. TensorFlow [Электронный ресурс] - Режим доступа - <https://www.tensorflow.org/>