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.

2. Решение

2.1. 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 ин
тервалу
        # (границы интервала смещены на случайную величину)
        sig1 = np.sin(np.linspace(0.0 * math.pi + rand , 3.0 * math.pi + r
and , seq_length * 2))
        sig2 = np.cos(np.linspace(0.0 * math.pi + rand , 3.0 * math.pi + r
and , 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 т
естовых данных)
seq_length = sample_x.shape[0]

# Размер пакета количество(тестовых примеров), по которому усредняется гра
диент
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 ра
сходиться 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

```
Step 0/1500, train loss: 399.58905029296875,    TEST loss: 315.69500732421875
Step 100/1500, train loss: 0.45104649662971497,         TEST loss: 0.5650931596755981
Step 200/1500, train loss: 0.4277403652667999,  TEST loss: 0.6213181018829346
Step 300/1500, train loss: 0.6972335577011108,  TEST loss: 0.5928292870521545
Step 400/1500, train loss: 0.09511750191450119,         TEST loss: 0.12446720153093338
Step 500/1500, train loss: 0.8414625525474548,  TEST loss: 0.7839953899383545
Step 600/1500, train loss: 0.06972821801900864,         TEST loss: 0.11182419955730438
Step 700/1500, train loss: 0.3685685396194458,  TEST loss: 0.3547897934913635
Step 800/1500, train loss: 0.20790152251720428,         TEST loss: 0.1486162543296814
Step 900/1500, train loss: 0.06917741149663925,         TEST loss: 0.04984629154205322
Step 1000/1500, train loss: 0.8158416748046875,         TEST loss: 0.3869292140007019
Step 1100/1500, train loss: 0.4972650706768036,         TEST loss: 0.38982197642326355
Step 1200/1500, train loss: 0.050588179379701614,       TEST loss: 0.05446721613407135
Step 1300/1500, train loss: 0.389155775308609,  TEST loss: 0.5287265181541443
Step 1400/1500, train loss: 0.031019527465105057,       TEST loss: 0.028560372069478035
```

Ошибка во время обучения

Полученные предсказания

2.2. 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

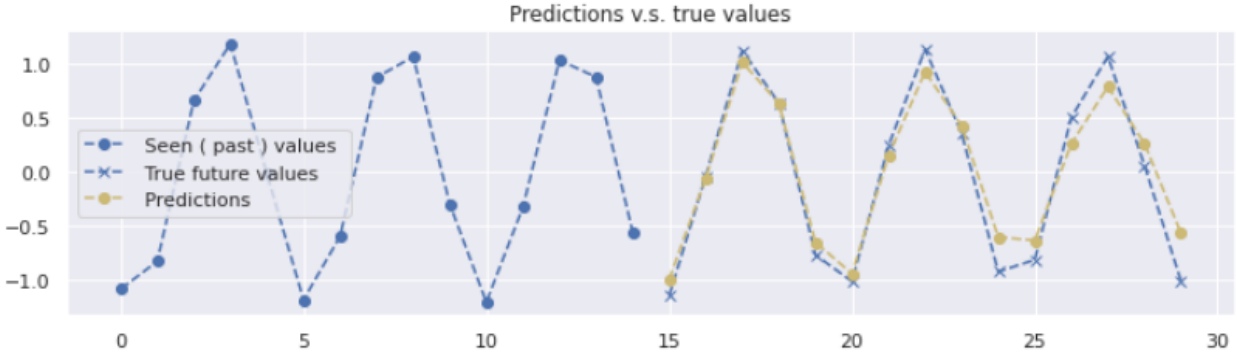
```
def generate_x_y_data_two_freqs(isTrain , batch_size , seq_length):
    batch_x = []
    batch_y = []

    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

```
Step 0/2500, train loss: 1080.309814453125,     TEST loss: 2660.6083984375
Step 100/2500, train loss: 653.9530029296875,   TEST loss: 584.1876220703125
Step 200/2500, train loss: 528.8779296875,      TEST loss: 588.0687255859375
Step 300/2500, train loss: 388.08514404296875,  TEST loss: 388.037353515625
Step 400/2500, train loss: 367.18402099609375,  TEST loss: 363.6763610839844
Step 500/2500, train loss: 407.16802978515625,  TEST loss: 371.5928955078125
Step 600/2500, train loss: 262.1539306640625,   TEST loss: 264.9698791503906
Step 700/2500, train loss: 261.9148864746094,   TEST loss: 246.21800231933594
Step 800/2500, train loss: 245.5906982421875,   TEST loss: 254.077880859375
Step 900/2500, train loss: 261.1959228515625,   TEST loss: 261.583740234375
Step 1000/2500, train loss: 216.74813842773438,         TEST loss: 212.20899963378906
Step 1100/2500, train loss: 283.7520751953125,  TEST loss: 266.4364013671875
Step 1200/2500, train loss: 172.85617065429688,         TEST loss: 215.8411102294922
Step 1300/2500, train loss: 147.30508422851562,         TEST loss: 162.9068145751953
Step 1400/2500, train loss: 209.6407012939453,  TEST loss: 262.1915283203125
Step 1500/2500, train loss: 233.4069366455078,  TEST loss: 182.8134002685547
Step 1600/2500, train loss: 272.6579895019531,  TEST loss: 192.37696838378906
Step 1700/2500, train loss: 164.44874572753906,         TEST loss: 189.24295043945312
Step 1800/2500, train loss: 148.04440307617188,         TEST loss: 132.8537139892578
Step 1900/2500, train loss: 225.56088256835938,         TEST loss: 178.19842529296875
Step 2000/2500, train loss: 138.3370361328125,  TEST loss: 142.2498321533203
Step 2100/2500, train loss: 149.03619384765625,         TEST loss: 137.21554565429688
Step 2200/2500, train loss: 172.17095947265625,         TEST loss: 171.08145141601562
Step 2300/2500, train loss: 151.39083862304688,         TEST loss: 169.27162170410156
Step 2400/2500, train loss: 139.5977020263672,  TEST loss: 149.91439819335938
Step 2500/2500, train loss: 192.97711181640625,         TEST loss: 197.6202392578125
Fin. train loss: 192.97711181640625,    TEST loss: 197.6202392578125
```

Ошибка во время обучения

Полученные предсказания



2.3. 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=se
q_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 т
естовых данных)
seq_length = sample_x.shape[0]

# Размер пакета количество(тестовых примеров), по которому усредняется гра
диент
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 ра
сходиться 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
```

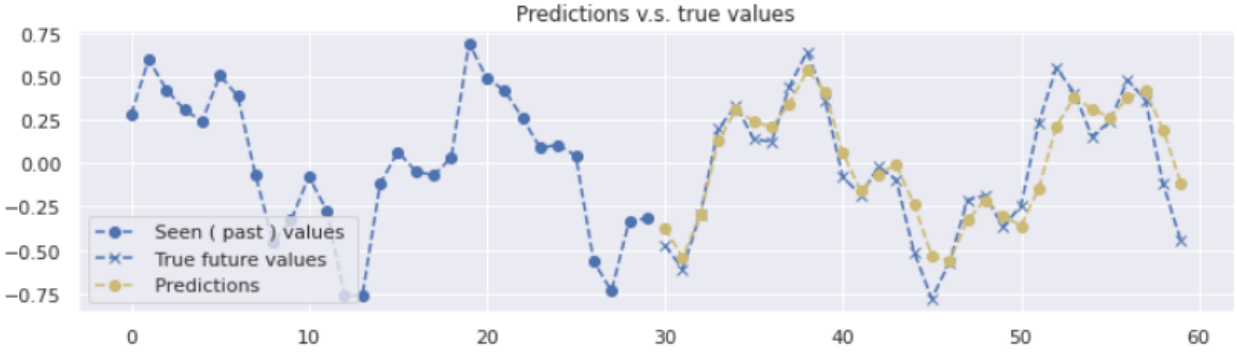
```
lambda_l2_reg = 1e-13
```

Значения функции потери во время обучения

```
Step 0/2500, train loss: 2941.4501953125,       TEST loss: 35303.234375
Step 100/2500, train loss: 456.996826171875,    TEST loss: 460.62554931640625
Step 200/2500, train loss: 435.25946044921875,  TEST loss: 450.9632568359375
Step 300/2500, train loss: 371.67681884765625,  TEST loss: 385.0115966796875
Step 400/2500, train loss: 322.2491149902344,   TEST loss: 355.0249938964844
Step 500/2500, train loss: 327.84564208984375,  TEST loss: 367.72601318359375
Step 600/2500, train loss: 306.7428894042969,   TEST loss: 340.6021728515625
Step 700/2500, train loss: 314.052001953125,    TEST loss: 263.91473388671875
Step 800/2500, train loss: 288.45074462890625,  TEST loss: 328.113525390625
Step 900/2500, train loss: 283.47772216796875,  TEST loss: 288.83795166015625
Step 1000/2500, train loss: 266.2341613769531,  TEST loss: 233.32936096191406
Step 1100/2500, train loss: 274.3446960449219,  TEST loss: 256.6833801269531
Step 1200/2500, train loss: 273.53240966796875,         TEST loss: 276.8822021484375
Step 1300/2500, train loss: 209.45684814453125,         TEST loss: 223.9257049560547
Step 1400/2500, train loss: 242.78933715820312,         TEST loss: 247.7176513671875
Step 1500/2500, train loss: 214.4210205078125,  TEST loss: 199.83477783203125
Step 1600/2500, train loss: 251.44122314453125,         TEST loss: 265.9028015136719
Step 1700/2500, train loss: 214.83604431152344,         TEST loss: 218.33554077148438
Step 1800/2500, train loss: 161.68453979492188,         TEST loss: 179.43814086914062
Step 1900/2500, train loss: 190.22085571289062,         TEST loss: 204.81837463378906
Step 2000/2500, train loss: 241.47715759277344,         TEST loss: 227.17340087890625
Step 2100/2500, train loss: 151.77320861816406,         TEST loss: 195.35992431640625
Step 2200/2500, train loss: 236.71694946289062,         TEST loss: 186.96890258789062
Step 2300/2500, train loss: 240.63507080078125,         TEST loss: 218.99537658691406
Step 2400/2500, train loss: 190.56961059570312,         TEST loss: 163.28785705566406
Step 2500/2500, train loss: 208.88470458984375,         TEST loss: 189.64105224609375
Fin. train loss: 208.88470458984375,    TEST loss: 189.64105224609375
```

Ошибка во время обучения

Полученные предсказания


Predictions v.s. true values
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
10
20
30
40
50
60
Seen (past) values
True future values
Predictions

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.

2. **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.

3. **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 №6 - М, 2020
2. TensorFlow [Электронный ресурс] - Режим доступа - https://www.tensorflow.org/