Введение в искусственные нейронные сети

Рекуррентные нейронные сети

На этом уроке

1. Познакомимся с рекуррентными нейронными сетями
2. Изучим их архитектуру
3. Применим знания на практике

Оглавление

Что такое рекуррентные нейронные сети

Нейронные сети, разобранные нами ранее, относятся к классу feed forward нейронных сетей или сетей прямого распространения. Выходной сигнал слоя в этих нейронных сетях передавался напрямую в следующий слой. Однако есть задачи, в которых нужно обучать нейронную сеть не на единичных экземплярах (наподобие изображений), а на наборах последовательностей, например, последовательностей слов.

В рекуррентной нейронной сети выходной сигнал внутренних слоёв циркулирует в этих слоях некоторое время. При обучении такой нейронной сети, прежние выходные сигналы используются как дополнительные input'ы. Можно сказать, что эти дополнительные input'ы конкатенируют с "нормальными" input'ами предыдущего слоя.

Рекуррентные нейронные сети используются, например, для того, чтобы научить компьютерные системы «понимать» человеческих язык, для генерации текста. Также нейронные сети с подобной архитектурой могут использоваться для любых задач, где осуществляется работа с некоторыми последовательностями значений, например, с биржевыми котировками. Разновидности рекуррентных нейронных сетей также используются для построения ИИ (подобных тем, что обыграли человека в компьютерную игру Dota 2). В отличие от свёрточных нейронных сетей, рекуррентные обычно содержат небольшое количество слоёв. Так, рекуррентная нейронная сеть в несколько десятков слоёв будет считаться большой.

Архитектура рекуррентных нейронных сетей

Несмотря на то, что RNN могут хорошо справляться со своими задачами, они не могут работать с длинными последовательностями. Эффективно они работают только с последовательностями, состоящими из 3-4 элементов. Например, для анализа текста отзывов на предмет того, положительный он или нет, этого будет недостаточно. Здесь может понадобиться анализ нескольких десятков слов, чтобы сделать корректный вывод. Давайте обсудим, почему обычной RNN не удается анализировать длинные последовательности.

Vanishing gradient problem

Из материалов по свёрточным нейронным сетям нам известна проблема исчезающего градиента. В случае с большим количеством слоёв значение градиента при последовательном их обновлении становиться всё меньше и может стать настолько маленьким, что не будет в состоянии существенно изменить поведение нейронов. В рекуррентных нейронных сетях из-за сигнала, циркулирующего

внутри слоёв, эта проблема становится ещё острее. Причём градиент может стать не только очень маленьким, но и крайне большим.

Long Short Term Memory(LSTM)

Решить проблему исчезающего градиента призвана разновидность RNN под названием LSTM.

Long short-term memory (LSTM) юниты — это блоки, из которых состоят слои одной из разновидностей рекуррентной нейронной сети(RNN). RNN, состоящая из LSTM юнитов, иногда называется просто LSTM. Обычно LSTM юнит представляет собой ячейку, состоящую из input gate, output gate и forget gate. Эти ячейки ответственны за запоминание значений на определённые промежутки времени.

Каждый из этих элементов можно представить как типичный искусственный нейрон в многослойной нейронной сети, они вычисляют активацию (используя функцию активации) как взвешенную сумму. Их работа сводиться к регуляции потока значений через блок LSTM, поэтому они и называются воротами или затворами (gate). Понятие долгой памяти в названии возникло из-за того, что они могут запоминать информацию на более длинный период времени, чем обычная RNN. LSTM хорошо подходит для классификации процессов и предсказания временных последовательностей неизвестного размера и неизвестных промежутков между важными событиями. С технической точки зрения это достигается за счёт ликвидации проблем, связанных с exploding и vanishinggradient'ами.

Компоненты LSTM

Ниже приведен список компонентов, из которых состоит ячейка LSTM:

Работа затворов в LSTM

Во-первых, LSTM ячейка берёт предыдущее состояние памяти Ct-1 и умножает на значение в forget gate(f), чтобы определить, присутствует ли состояние памяти Ct. Если forget gate значение равно 0, то предыдущее состояние памяти полностью забывается, если же f forget gate значение равно 1, то предыдущее значение состояния памяти полностью проходит через ячейку (помните, что f gate даёт значение между 0 и 1).

*Ct = Ct-1 * ft*

Вычисляем новое состояние памяти:

*Ct = Ct + (It * C`t)*

Теперь, вычисляем выходное значение:

Ht = tanh(Ct)

GRU

GRU — новое поколение рекуррентных нейронных сетей. Оно во многом похоже на LSTM, но есть определённая разница. В GRU не используется состояние ячейки и используется скрытое состояние для передачи информации. В GRU также есть два затвора: reset gate и update gate.

Update Gate обновляет затворы, действуя подобно forget и input gate, которые используются в LSTM. Он решает, какая информация будет отброшена, а какая новая информация будет добавлена. Reset Gate — другой затвор, использующийся для принятия решения о том, как много прошлой информации будет забыто.

В этих особенностях заключается архитектура GRU. GRU имеет меньше тензорных операций и, соответственно, тренируется быстрее, чем LSTM. Нельзя сказать точно, какая архитектура лучше, исследователи и инженеры определяют это для каждого конкретного случая. Обычно GRU может подойти в случаях, когда скорость важнее, чем точность, а LSTM — наоборот.

Практика

Попробуем сделать простую рекуррентную нейронную сеть, которая будет учиться складывать числа. Для этих целей мы не будем пользоваться фреймворками для Deep Learning (чтобы посмотреть, как она работает внутри).

```
# впервую очередь подключим numpy и библиотеку copy, которая понадобиться, чтобы сделать deepcopy ряда элементов


import copy, numpy as np

np.random.seed(0)


# вычислим сигмоиду

def sigmoid(x):

    output = 1/(1+np.exp(-x))

    return output


# конвертируем значение функции сигмоиды в ее производную.

def sigmoid_output_to_derivative(output):

    return output*(1-output)


# генерация тренировочного датасета

int2binary = {}

binary_dim = 8
```

```
largest_number = pow(2,binary_dim)
binary = np.unpackbits(
    np.array([list(range(largest_number))],dtype=np.uint8).T,axis=1)
for i in range(largest_number):
    int2binary[i] = binary[i]


# входные переменные
alpha = 0.1
input_dim = 2
hidden_dim = 16
output_dim = 1



# инициализация весов нейронной сети
synapse_0 = 2*np.random.random((input_dim,hidden_dim)) - 1
synapse_1 = 2*np.random.random((hidden_dim,output_dim)) - 1
synapse_h = 2*np.random.random((hidden_dim,hidden_dim)) - 1

synapse_0_update = np.zeros_like(synapse_0)
synapse_1_update = np.zeros_like(synapse_1)
synapse_h_update = np.zeros_like(synapse_h)

# тренировочная логика
for j in range(10000):

    # генерация простой проблемы сложения (a + b = c)
    a_int = np.random.randint(largest_number/2) # int version
    a = int2binary[a_int] # бинарное кодирование

    b_int = np.random.randint(largest_number/2) # int version
    b = int2binary[b_int] # бинарное кодирование

    # правильный ответ
    c_int = a_int + b_int
    c = int2binary[c_int]

    # место где мы располагаем наши лучше результаты (бинарно закодированные)
```

```
d = np.zeros_like(c)


overallError = 0


layer_2_deltas = list()

layer_1_values = list()

layer_1_values.append(np.zeros(hidden_dim))


# движение вдоль позиций бинарной кодировки

for position in range(binary_dim):


    # генерация input и output

    X = np.array([[a[binary_dim - position - 1],b[binary_dim - position - 1]]])

    y = np.array([[c[binary_dim - position - 1]]]).T


    # внутренний слой (input ~+ предыдущий внутренний)

    layer_1 = sigmoid(np.dot(X,synapse_0) + np.dot(layer_1_values[-1],synapse_h))


    # output layer (новое бинарное представление)

    layer_2 = sigmoid(np.dot(layer_1,synapse_1))


    # проверка упустили ли мы что-то и если да, то как много

    layer_2_error = y - layer_2

    layer_2_deltas.append((layer_2_error)*sigmoid_output_to_derivative(layer_2))

    overallError += np.abs(layer_2_error[0])


    # декодируем оценку чтобы мы могли ее вывести на экран

    d[binary_dim - position - 1] = np.round(layer_2[0][0])


    # сохраняем внутренний слой, чтобы мы могли его использовать в след. timestep

    layer_1_values.append(copy.deepcopy(layer_1))


future_layer_1_delta = np.zeros(hidden_dim)


for position in range(binary_dim):


    X = np.array([[a[position],b[position]]])

    layer_1 = layer_1_values[-position-1]
```

```
        prev_layer_1 = layer_1_values[-position-2]


        # величина ошибки в output layer
        layer_2_delta = layer_2_deltas[-position-1]
        # величина ошибки в hidden layer
        layer_1_delta = (future_layer_1_delta.dot(synapse_h.T) + layer_2_delta.dot(synapse_1.T)) *
sigmoid_output_to_derivative(layer_1)


        # обновление всех весов и пробуем заново
        synapse_1_update += np.atleast_2d(layer_1).T.dot(layer_2_delta)
        synapse_h_update += np.atleast_2d(prev_layer_1).T.dot(layer_1_delta)
        synapse_0_update += X.T.dot(layer_1_delta)


        future_layer_1_delta = layer_1_delta




    synapse_0 += synapse_0_update * alpha
    synapse_1 += synapse_1_update * alpha
    synapse_h += synapse_h_update * alpha


    synapse_0_update *= 0
    synapse_1_update *= 0
    synapse_h_update *= 0


    # вывод на экран процесса обучения
    if(j % 1000 == 0):
        print("Error:" + str(overallError))
        print("Pred:" + str(d))
        print("True:" + str(c))
        out = 0
        for index,x in enumerate(reversed(d)):
            out += x*pow(2,index)
        print(str(a_int) + " + " + str(b_int) + " = " + str(out))
        print("------------")
Error:[3.45638663]
Pred:[0 0 0 0 0 0 0 1]
True:[0 1 0 0 0 1 0 1]
9 + 60 = 1
```

```
------------

Error:[3.63389116]

Pred:[1 1 1 1 1 1 1 1]

True:[0 0 1 1 1 1 1 1]

28 + 35 = 255

------------

Error:[3.91366595]

Pred:[0 1 0 0 1 0 0 0]

True:[1 0 1 0 0 0 0 0]

116 + 44 = 72

------------

Error:[3.72191702]

Pred:[1 1 0 1 1 1 1 1]

True:[0 1 0 0 1 1 0 1]

4 + 73 = 223

------------

Error:[3.5852713]

Pred:[0 0 0 0 1 0 0 0]

True:[0 1 0 1 0 0 1 0]

71 + 11 = 8

------------

Error:[2.53352328]

Pred:[1 0 1 0 0 0 1 0]

True:[1 1 0 0 0 0 1 0]

81 + 113 = 162

------------

Error:[0.57691441]

Pred:[0 1 0 1 0 0 0 1]

True:[0 1 0 1 0 0 0 1]

81 + 0 = 81

------------

Error:[1.42589952]

Pred:[1 0 0 0 0 0 0 1]

True:[1 0 0 0 0 0 0 1]

4 + 125 = 129

------------

Error:[0.47477457]

Pred:[0 0 1 1 1 0 0 0]
```

```
True:[0 0 1 1 1 0 0 0]

39 + 17 = 56

------------

Error:[0.21595037]

Pred:[0 0 0 0 1 1 1 0]

True:[0 0 0 0 1 1 1 0]

11 + 3 = 14

------------
```

Теперь попробуем с помощью Keras построить LSTM нейронную сеть для оценки настроений отзывов на IMD.

Данный датасет слишком мал для того, чтобы проявить преимущества LSTM, однако в учебных целях его будет достаточно.

В тренировке рекуррентных нейронных сетей важную роль играет размер batch, но ещё большую роль играет выбор функций loss и optimizer.

```
from __future__ import print_function


from keras.preprocessing import sequence

from keras.models import Sequential

from keras.layers import Dense, Embedding

from keras.layers import LSTM

from keras.datasets import imdb


max_features = 20000


# обрезание текстов после данного количества слов (среди top max_features наиболее используемые слова)

maxlen = 80

batch_size = 50 # увеличьте значение для ускорения обучения


print('Загрузка данных...')

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

print(len(x_train), 'тренировочные последовательности')

print(len(x_test), 'тестовые последовательности')


print('Pad последовательности (примеров в x единицу времени)')
```

```
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)

x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

print('x_train shape:', x_train.shape)

print('x_test shape:', x_test.shape)


print('Построение модели...')

model = Sequential()

model.add(Embedding(max_features, 128))

model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))

model.add(Dense(1, activation='sigmoid'))


# стоит попробовать использовать другие оптимайзер и другие конфигурации оптимайзеров

model.compile(loss='binary_crossentropy',

              optimizer='adam',

              metrics=['accuracy'])


print('Процесс обучения...')

model.fit(x_train, y_train,

          batch_size=batch_size,

          epochs=1, # увеличьте при необходимости

          validation_data=(x_test, y_test))

score, acc = model.evaluate(x_test, y_test,

                            batch_size=batch_size)

print('Результат при тестировании:', score)

print('Тестовая точность:', acc)

Загрузка данных...

25000 тренировочные последовательности

25000 тестовые последовательности

Pad последовательности (примеров в x единицу времени)

x_train shape: (25000, 80)

x_test shape: (25000, 80)

Построение модели...

Процесс обучения...

/usr/local/lib/python3.7/dist-packages/tensorflow_core/python/framework/indexed_slices.py:433: UserWarning:
Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.

  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 25000 samples, validate on 25000 samples

Epoch 1/1

25000/25000 [==============================] - 24s 966us/step - loss: 0.4608 - accuracy: 0.7844 - val_loss:
```

```
0.3825 - val_accuracy: 0.8336

25000/25000 [==============================] - 6s 232us/step

Результат при тестировании: 0.3825102120637894

Тестовая точность: 0.8335599899291992
```

Также рассмотрим пример, в котором будет использоваться другой класс задач: генерация текста на основе тренировочного текста. Задача нейросети: обучиться на тексте «Алиса в стране чудес» и начать генерировать текст, похожий на тот, что можно встретить в книге. В примере будет использоваться GRU.

```
import numpy as np

from keras.layers import Dense, Activation

from keras.layers.recurrent import SimpleRNN, LSTM, GRU

from keras.models import Sequential



# построчное чтение из примера с текстом

with open("alice_in_wonderland.txt", 'rb') as _in:

    lines = []

    for line in _in:

        line = line.strip().lower().decode("ascii", "ignore")

        if len(line) == 0:

            continue

        lines.append(line)

text = " ".join(lines)

chars = set([c for c in text])

nb_chars = len(chars)



# создание индекса символов и reverse mapping чтобы передвигаться между значениями numerical

# ID and a specific character. The numerical ID will correspond to a column

# ID и определенный символ. Numerical ID будет соответсвовать колонке

# число при использовании one-hot кодировки для представление входов символов

char2index = {c: i for i, c in enumerate(chars)}

index2char = {i: c for i, c in enumerate(chars)}



# для удобства выберете фиксированную длину последовательность 10 символов
```

```
SEQLEN, STEP = 10, 1

input_chars, label_chars = [], []


# конвертация data в серии разных SEQLEN-length субпоследовательностей

for i in range(0, len(text) - SEQLEN, STEP):

    input_chars.append(text[i: i + SEQLEN])

    label_chars.append(text[i + SEQLEN])




# Вычисление one-hot encoding входных последовательностей X и следующего символа (the label) y


X = np.zeros((len(input_chars), SEQLEN, nb_chars), dtype=np.bool)

y = np.zeros((len(input_chars), nb_chars), dtype=np.bool)

for i, input_char in enumerate(input_chars):

    for j, ch in enumerate(input_char):

        X[i, j, char2index[ch]] = 1

    y[i, char2index[label_chars[i]]] = 1




# установка ряда метапамертров  для нейронной сети и процесса тренировки

BATCH_SIZE, HIDDEN_SIZE = 128, 128

NUM_ITERATIONS = 1 # 25 должно быть достаточно

NUM_EPOCHS_PER_ITERATION = 1

NUM_PREDS_PER_EPOCH = 100




# Create a super simple recurrent neural network. There is one recurrent

# layer that produces an embedding of size HIDDEN_SIZE from the one-hot

# encoded input layer. This is followed by a Dense fully-connected layer

# across the set of possible next characters, which is converted to a

# probability score via a standard softmax activation with a multi-class

# cross-entropy loss function linking the prediction to the one-hot

# encoding character label.


'''

Создание очень простой рекуррентной нейронной сети. В ней будет один реккурентный закодированный входной слой.
За ним последует полносвязный слой связанный с набором возможных следующих символов, которые конвертированы в
вероятностные результаты через стандартную softmax активацию с multi-class cross-encoding loss функцию
ссылающуются на предсказание one-hot encoding лейбл символа
```

```
'''

model = Sequential()
model.add(
    GRU(  # вы можете изменить эту часть на LSTM или SimpleRNN, чтобы попробовать альтернативы
        HIDDEN_SIZE,
        return_sequences=False,
        input_shape=(SEQLEN, nb_chars),
        unroll=True
    )
)
model.add(Dense(nb_chars))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy", optimizer="rmsprop")



# выполнение серий тренировочных и демонстрационных итераций
for iteration in range(NUM_ITERATIONS):

    # для каждой итерации запуск передачи данных в модель
    print("=" * 50)
    print("Итерация #: %d" % (iteration))
    model.fit(X, y, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS_PER_ITERATION)

    # Select a random example input sequence.
    test_idx = np.random.randint(len(input_chars))
    test_chars = input_chars[test_idx]

    # для числа шагов предсказаний использование текущей тренируемой модели
    # конструирование one-hot encoding для тестирования input и добавление предсказания.
    print("Генерация из посева: %s" % (test_chars))
    print(test_chars, end="")
    for i in range(NUM_PREDS_PER_EPOCH):

        # здесь one-hot encoding.
        X_test = np.zeros((1, SEQLEN, nb_chars))
        for j, ch in enumerate(test_chars):
            X_test[0, j, char2index[ch]] = 1
```

```
        # осуществление предсказания с помощью текущей модели.
        pred = model.predict(X_test, verbose=0)[0]
        y_pred = index2char[np.argmax(pred)]

        # вывод предсказания добавленного к тестовому примеру
        print(y_pred, end="")

        # инкрементация тестового примера содержащего предсказание
        test_chars = test_chars[1:] + y_pred
print()
==================================================
Итерация #: 0
Epoch 1/1
158773/158773 [==============================] - 14s 89us/step - loss: 2.3163
Генерация из посева: ry: ill tr
ry: ill treand the was the was the was the was the was the was the was the was the was the was the was
```

Используемые источники

1. https://www.kaggle.com/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru
2. Шакла Н. — Машинное обучение и TensorFlow 2019
3. Николенко, Кадурин, Архангельская: Глубокое обучение. Погружение в мир нейронных сетей 2018
4. Aurélien Géron — Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2019
5. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
6. https://github.com/llSourcell/recurrent_neural_net_demo