МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторная работа № 8

тематика

Генерация текста на основе “Алисы в стране чудес”

Москва 2021

Цель

Рекуррентные нейронные сети также могут быть использованы в качестве генеративных моделей.
Это означает, что в дополнение к тому, что они используются для прогнозных моделей (создания прогнозов), они могут изучать последовательности проблемы, а затем генерировать совершенно новые вероятные последовательности для проблемной области.
Подобные генеративные модели полезны не только для изучения того, насколько хорошо модель выявила проблему, но и для того, чтобы узнать больше о самой проблемной области.

Задачи

- Ознакомиться с генерацией текста
- Ознакомиться с системой Callback в Keras

Выполнение работы

Описание проблемы: проект Гутенберг

Многие из классических текстов больше не защищены авторским правом.
Это означает, что вы можете скачать весь текст этих книг бесплатно и использовать их в экспериментах, например, при создании генеративных моделей. Возможно, лучшее место для получения доступа к бесплатным книгам, которые больше не защищены авторским правом, это Проект Гутенберг.
В данной лабораторной работе мы будем использовать в качестве набора данных Приключения Алисы в Стране Чудес Льюиса Кэрролла. Мы собираемся изучить зависимости между символами и условные вероятности символов в последовательностях, чтобы мы могли, в свою очередь, генерировать совершенно новые и оригинальные последовательности символов.
Эти эксперименты не ограничиваются текстом, вы также можете поэкспериментировать с другими данными ASCII, такими как размеченные документы в LaTeX, HTML или Markdown и другие.
Вы можете скачать полный текст в формате ASCII(Обычный текст UTF-8) для этой книги бесплатно и поместите ее в свой рабочий каталог с именем файла wonderland.txt
Теперь нам нужно подготовить набор данных к моделированию.
Project Gutenberg добавляет стандартный колонтитул к каждой книге, и это не является частью исходного текста. Откройте файл в текстовом редакторе и удалите верхний и нижний колонтитулы.
Необходимо удалить все до:

```
*** START OF THIS PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN
WONDERLAND ***
```

и после:

```
THE END
```

Начнем с импорта необходимых зависимостей для предварительной обработки данных и построения модели.

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
```

Затем нам нужно загрузить текст ASCII для книги в память и преобразовать все символы в нижний регистр, чтобы уменьшить словарный запас, который должна выучить сеть.

```
filename = "wonderland.txt"
raw_text = open(filename).read()
raw_text = raw_text.lower()
```

Теперь, когда книга загружена, мы должны подготовить данные для моделирования нейронной сетью. Мы не можем моделировать символы напрямую, вместо этого мы должны преобразовать символы в целые числа.

Мы можем сделать это легко, сначала создав набор всех отдельных символов в книге, а затем создав карту каждого символа с уникальным целым числом.

```
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
```

Теперь, когда книга загружена и карта подготовлена, мы можем суммировать набор данных.

```
n_chars = len(raw_text)
n_vocab = len(chars)
print "Total Characters: ", n_chars
print "Total Vocab: ", n_vocab
```

Мы видим, что книга содержит менее 150000 символов и что при преобразовании в строчные буквы в словаре есть только 47 различных символов для изучения сетью. Гораздо больше, чем 26 в алфавите.

Теперь нам нужно определить данные обучения для сети Существует большая гибкость в том, как вы решаете разбивать текст и выставлять его в сети во время обучения.

В этом уроке мы разделим текст книги на подпоследовательности с фиксированной длиной в 100 символов произвольной длины. Мы могли бы так же легко разделить данные по предложениям, дополнить более короткие последовательности и укоротить более длинные.

Каждый обучающий шаблон сети состоит из 100 временных шагов одного символа (X), за которыми следует один символьный вывод (y). При создании этих последовательностей мы перемещаем это окно по всей книге по одному символу за раз, позволяя каждому персонажу выучить шанс из 100 предшествующих ему символов (кроме, конечно, первых 100 символов).

Например, если длина последовательности равна 5 (для простоты), то первые два шаблона обучения будут следующими:

```
CHAPT -> E
HAPTE -> R
```

Разделяя книгу на эти последовательности, мы конвертируем символы в целые числа, используя нашу таблицу поиска, которую мы подготовили ранее.

```
seq_length = 100
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
        seq_in = raw_text[i:i + seq_length]
        seq_out = raw_text[i + seq_length]
        dataX.append([char_to_int[char] for char in seq_in])
        dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print "Total Patterns: ", n_patterns
```

Теперь, когда мы подготовили наши тренировочные данные, нам нужно преобразовать их так, чтобы они подходили для использования с Keras.

Сначала мы должны преобразовать список входных последовательностей в форму[образцы, временные шаги, особенности]ожидается сетью LSTM.
Затем нам нужно изменить масштаб целых чисел в диапазоне от 0 до 1, чтобы облегчить изучение шаблонов сетью LSTM, которая по умолчанию использует функцию активации сигмовидной кишки.
Наконец, нам нужно преобразовать выходные шаблоны (отдельные символы, преобразованные в целые числа) в одну кодировку. Это сделано для того, чтобы мы могли настроить сеть так, чтобы она предсказывала вероятность каждого из 47 различных символов в словаре (более простое представление), а не пыталась заставить ее предсказать точно следующий символ. Каждое значение у преобразуется в разреженный вектор длиной 47, полный нулей, за исключением 1 в столбце для буквы (целое число), которую представляет шаблон.
Мы можем реализовать эти шаги, как показано ниже.

```
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
```

Теперь мы можем определить нашу модель LSTM. Здесь мы определяем один скрытый слой LSTM с 256 единицами памяти. Сеть использует выпадение с вероятностью 20. Выходной уровень - это Плотный уровень, использующий функцию активации softmax для вывода прогнозирования вероятности для каждого из 47 символов в диапазоне от 0 до 1.
Эта проблема на самом деле представляет собой проблему классификации отдельных символов с 47 классами, и поэтому она определяется как оптимизация потерь (перекрестная энтропия) с использованием алгоритма оптимизации ADAM по скорости.

```
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer='adam')
```

Тестового набора данных нет. Мы моделируем весь обучающий набор данных, чтобы узнать вероятность каждого символа в последовательности.
Сеть работает медленно. Из-за медлительности и из-за наших требований по оптимизации мы будем использовать контрольные точки модели для записи всех сетевых весов, чтобы каждый раз регистрировать улучшение потерь в конце эпохи. Мы будем использовать лучший набор весов (наименьшая потеря), чтобы реализовать нашу генеративную модель в следующем разделе.

```
# define the checkpoint
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss',
verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]
```

Теперь осталось обучить модель

```
model.fit(X, y, epochs=20, batch_size=128,
callbacks=callbacks_list)
```

После запуска примера у вас должно быть несколько файлов контрольных точек веса в локальном каталоге. Вы можете удалить их все, кроме одного с наименьшим значением потери.

Генерация текста с помощью сети LSTM

Генерация текста с использованием обученной сети LSTM относительно проста.
Во-первых, мы загружаем данные и определяем сеть точно таким же образом, за исключением того, что веса сети загружаются из файла контрольных точек, и сеть не нуждается в обучении.

```
# load the network weights
filename = "weights-improvement-19-1.9435.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy',
optimizer='adam')
```

Кроме того, при подготовке сопоставления уникальных символов с целыми числами мы также должны создать обратное отображение, которое мы можем использовать для преобразования целых чисел обратно в символы, чтобы мы могли понять предсказания.

```
int_to_char = dict((i, c) for i, c in enumerate(chars))
```

Простейший способ использования модели Keras LSTM для прогнозирования - сначала начать с последовательности начальных чисел в качестве входных данных, сгенерировать следующий символ, затем обновить последовательность начальных чисел, чтобы добавить сгенерированный символ в конце, и обрезать первый символ. Этот процесс повторяется до тех пор, пока мы хотим предсказать новые символы (например, последовательность длиной 1000 символов).
Мы можем выбрать случайный шаблон ввода в качестве нашей начальной последовательности, а затем распечатать сгенерированные символы по мере их генерации.

```
# pick a random seed
start = numpy.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print "Seed:"
print "\"", ''.join([int_to_char[value] for value in
pattern]), "\""
# generate characters
for i in range(1000):
        x = numpy.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
        prediction = model.predict(x, verbose=0)
        index = numpy.argmax(prediction)
        result = int_to_char[index]
        seq_in = [int_to_char[value] for value in pattern]
        sys.stdout.write(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
print "\nDone."
```

Полный пример кода для генерации текста с использованием загруженной модели LSTM

```
# Load LSTM network and generate text
import sys
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
```

```
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
# load ascii text and covert to lowercase
filename = "wonderland.txt"
raw_text = open(filename).read()
raw_text = raw_text.lower()
# create mapping of unique chars to integers, and a reverse
mapping
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
int_to_char = dict((i, c) for i, c in enumerate(chars))
# summarize the loaded data
n_chars = len(raw_text)
n_vocab = len(chars)
print "Total Characters: ", n_chars
print "Total Vocab: ", n_vocab
# prepare the dataset of input to output pairs encoded as
integers
seq_length = 100
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
        seq_in = raw_text[i:i + seq_length]
        seq_out = raw_text[i + seq_length]
        dataX.append([char_to_int[char] for char in seq_in])
        dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print "Total Patterns: ", n_patterns
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# define the LSTM model
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
# load the network weights
filename = "weights-improvement-19-1.9435.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy',
optimizer='adam')
# pick a random seed
start = numpy.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print "Seed:"
print "\"", ''.join([int_to_char[value] for value in
pattern]), "\""
# generate characters
for i in range(1000):
```

```
        x = numpy.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
        prediction = model.predict(x, verbose=0)
        index = numpy.argmax(prediction)
        result = int_to_char[index]
        seq_in = [int_to_char[value] for value in pattern]
        sys.stdout.write(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
print "\nDone."
```

Требования

1. Реализовать модель ИНС, которая будет генерировать текст
2. Написать собственный CallBack, который будет показывать то как генерируется текст во время обучения (то есть раз в какое-то количество эпох генирировать и выводить текст у необученной модели)
3. Отследить процесс обучения при помощи TensorFlowCallBack (TensorBoard), в отчете привести результаты и их анализ