# Параллельные вычисления. Лабораторная работа 4

## Введение

Цель работы – ознакомиться с возможностями библиотеки Keras и научиться создавать генеративные нейронные сети с ее помощью.

Задание – сгенерировать текст с помощью нейронной сети, сравнить результаты работы нейронных сетей с разной глубиной.

## 1. Теоретическая часть

## 1.1 Ключевые особенности Keras

*«Being able to go from idea to result with the*

*least possible delay is key to doing good research»*

*(Возможность прийти от идеи к результату за*

*максимально короткий срок – ключ к хорошим исследованиям)*

Keras – это библиотека для глубокого обучения, написанная на языке Python. Она разработана как надстройка над Theano, TensorFlow или CNK для упрощения работы с ними. Keras создан для быстрого и простого прототипирования, так как процесс ознакомления с основами, необходимыми для проектирования и обучения нейронной сети, занимает очень мало времени. Большинство функций интуитивно понятны даже поверхностно знакомому с методами машинного обучения человеку.

Несмотря на внешнюю простоту, Keras является мощным и гибким инструментом. К его ключевым особенностям можно отнести:

1. возможность запуска кода с разными бэкендами;
2. можно запустить вычисления как на CPU, так и на GPU;
3. поддерживает сверточные и рекуррентные сети, а также их комбинации;
4. не требует отдельных конфигурационных файлов с декларативным описанием модели;
5. имеет встроенные инструменты для визуализации моделей.

Таким образом, с помощью Keras можно воплотить в жизнь практически любую архитектуру нейронной сети. Кроме того, Keras предоставляет инструменты и для предварительной обработки текста, изображений, последовательностей.

С общей теорией по нейронным сетям можно ознакомиться в этой статье: <https://habrahabr.ru/post/312450/> Она объясняет понятия: нейрон, предсказание, функция активации, обучение, итерация, эпоха, ошибка.

## 1.2 Генерация текста посредством нейронных сетей

После ознакомления с Keras, рассмотрим собственно задачу генерации текста посредством нейронных сетей. В данной лабораторной работе будет рассмотрена побуквенная генерация текста посредством LSTM нейронной сети.

LSTM-сети относятся к классу рекуррентных. Главная особенность рекуррентных сетей: в каждом шаге вычислений нейронной сети задействуется информация о предыдущем шаге, то есть, присутствует обратная связь. Условно, каждый нейрон «запоминает» информацию о своем предыдущем состоянии и при вычислении функции активации учитывает и прежнее состояние, и новые данные.

LSTM (Long Short Term Memory) – или иначе, сети долгой краткосрочной памяти, также содержат обратные связи. Каждый нейрон имеет память о состоянии, однако способ обновления этого состояния более сложный. Изменение состояния нейрона происходит с помощью фильтров, они регулируют какую часть входной информации следует сохранить, а какую – отбросить.

Таким образом, LSTM-сеть может сохранять информацию о контексте предложения или целого абзаца, так как собирает частицы информации о каждом из значимых предыдущих состояний, а не только о последнем. Это решает проблему потери контекста в длинных текстах и улучшает результаты анализа. Подробнее с LSTM можно ознакомиться по следующей ссылке: <https://habrahabr.ru/company/wunderfund/blog/331310/>

Чтобы сгенерировать текст с помощью нейронной сети, нужно обучить ее зависимостям между символами и условным вероятностям встречаемости букв в последовательности. Для обучения сети нужно большое количество текстовых примеров, поэтому в качестве источника обучающих данных будет использоваться текст книги «Алиса в Стране чудес» на английском языке, так как в нем меньше словоформ, чем в русском, и результат генерации должен быть лучше. Количество различных символов в тексте без очистки пунктуации: 45.

Определим формат обучающей выборки таким образом: входные данные: последовательность символов длиной 100 символов, выходные данные: следующий символ (101й).

Таким образом, исходный текст будет обрабатываться скользящим окном размером 100 с шагом 1. Так мы получим большое количество данных для обучения. Эти обучающие данные научат сеть предсказывать следующий символ по 100 предыдущим.

Собственно генерация текста с помощью обученной сети производится следующим образом: берем случайную последовательность из текста и используем ее последние 100 символов для предсказания следующего символа. Добавляем к ее концу предсказанный символ. И далее повторяем этот процесс столько раз, сколько символов требуется предсказать.

## 1.3 Принципы создания нейронной сети

Популярность Keras среди разработчиков обеспечена простотой создания, расширения и модификации архитектуры нейронной сети. Рассмотрим процесс создания нейронной сети на примере генеративной LSTM-сети.

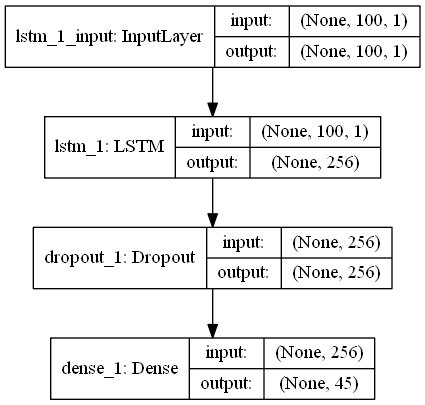


Рисунок 1 – Архитектура генеративной LSTM сети

Приведенная на рисунке нейронная сеть состоит из четырех полносвязных последовательно соединенных слоев: входнаой (100 нейронов, подаются по 1), LSTM (256 нейронов), слой регуляризации (256 нейронов), выходной (45 нейронов). При этом входной слой просто определяет форму входных данных и не выполняет вычислений. Слой регуляризации (Dropout) необходим, чтобы избежать переобучения сети, когда она слишком точно запоминает входной контекст и не может генерировать новую информацию.

Нейронная сеть в Keras представляется в виде модели – структуры для организации слоев. Различают два типа моделей:

1. Sequential – последовательная, представляет собой линейную последовательность слоев, подходит для большинства экспериментов;
2. Model (functional API) – модель для более сложных структур, позволяет организовать слои в виде произвольного графа с несколькими входными и выходными слоями.

В нашем случае слои соединяются последовательно, поэтому воспользуемся первым типом. Для создания пустой структуры последовательной модели необходимо запустить следующий код:

model = Sequential()

Чтобы построить требуемую архитектуру сети необходимо просто последовательно добавить в пустую модель слои. Четыре требуемых слоя будут добавлены следующим образом:

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'))

Входные данные традиционно именуются переменной X, заключение – y. Важно заметить, что входной слой, задача которого просто определить количество входных элементов, отдельно не добавляется – его параметр указывается в первом вычислительном слое. Функция add принимает в качестве аргумента объект типа Слой (Layer). В Keras существуют следующие часто используемые типы слоев:

1. Сore – основные, включает Dense – полносвязные слои, Dropout – слои регуляризации;
2. Convolutional – сверточные слои;
3. Pooling – слои объединения;
4. Locally-connected – не полносвязные;
5. Recurrent – рекуррентные слои, включает LSTM.

Существуют и другие типы слоев, предназначенные для определенных задач, с их подробными описаниями можно подробнее ознакомиться здесь: <https://keras.io/layers/>

Аргументы конструкторов слоев в данном случае:

1. units (256) – количество нейронов в слое;
2. activation – функция активации (softmax, relu, tanh, sigmoid, linear и др.);
3. input\_dim – количество входных нейронов сети, необходимо указывать для входного слоя;
4. rate (0.2) – вероятность отключения нейрона при регуляризации.

Существует и иной способ добавления слоев, они полностью идентичны:

model = Sequential([

LSTM(256, input\_shape=(X.shape[1], X.shape[2]))

Dropout(0.2),

Dense(y.shape[1], activation='softmax')

])

Для перехода к обучению модели нужно произвести ее компиляцию. Это необходимо, чтобы определить:

1. оптимизатор вычислений;
2. функцию потерь;
3. метрики.

Так как рассматриваемая сеть имеет 45 выходов, она является мульти-классификатором. Для задачи мульти-классификации рекомендуется использовать: функцию потерь «категориальная кросс-энтропия», оптимизатор – Adam, в качестве метрики эффективности нейронной сети укажем accuracy (точность, правильность). Код будет выглядеть таким образом:

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

Теперь можно приступать непосредственно к обучению. Оно производится с помощью функции:

model.fit(X, y, epochs=20, batch\_size=128)

где:

1. X – входные данные;
2. y – ожидаемые значения на выходе для каждого экземпляра входных данных;
3. epochs – количество эпох (сколько раз все входные данные будут проходить через сеть);
4. batch\_size – (размер партии) количество экземпляров входных данных, обрабатываемых за одну итерацию.

Batch size необходимо подбирать исходя из доступной вычислительной мощности устройства. Количество эпох определяется по результатам обучения (если сеть недообучена или переобучена, количество эпох может быть изменено).

После обучения модели, можно приступать к ее использованию:

char = model.predict(x\_test)

Predict делает предсказание о выходном значении по входным данным.

В данной работе мы не заинтересованы в максимальной точности, так как это даст модель, которая просто повторяет тренировочные выборки, не генерируя новые. Поэтому успешность обучения будем отслеживать по убыванию функции потерь.

Чтобы получить текст длиной 100 с помощью обученной модели, нужно будет 100 раз запустить предсказание, так как за одну итерацию модель предсказывает один символ.

## 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Использование простой LSTM сети для генерации текста

## 2.1.1 Код для обучения сети (простая модель)

Необходимые материалы для выполнения лабораторной работы находятся в директории (labs/lab4).

Для генерации текста будем использовать модель с вышеприведенной архитектурой (рисунок 1). Эта модель довольно простая, так как помимо регуляризации содержит только один скрытый слой.

Исходный код для обучения данной модели находится в: labs/lab4/simple\_model.py. Рассмотрим его подробнее.

*# Импорт всего необходимого***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  
**import** io  
*# Приводим текст к нижнему регистру, чтобы сократить*

*# будущий словарь символов, которые сети предстоит выучить*filename = **"wonderland.txt"**raw\_text = io.open(filename, encoding=**'utf-8'**).read()  
raw\_text = raw\_text.lower()

*# Составляем словарь уникальных символов на основе текста,*

*# так как нейросеть работает только с числами –*

*# создаем словарь соответствий «символ – число»*chars = sorted(list(set(raw\_text)))  
char\_to\_int = dict((c, i) **for** i, c **in** enumerate(chars))

*# Подсчитываем исходные данные*n\_chars = len(raw\_text)  
n\_vocab = len(chars)  
print(**"Total Characters: "**, n\_chars)  
print(**"Total Vocab: "**, n\_vocab)

*# Подготавливаем обучающую выборку из образцов по 100 символов*

*# на входе и одного символа на выходе. С помощью словаря*

*# преобразуем последовательности символов в числовые*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)

*# Приводим входные данные (X) к форме:*

*# [образцы, длина последовательности, символы]*X = numpy.reshape(dataX, (n\_patterns, seq\_length, 1))

*# Нормализируем – приводим числа к интервалу от 0 до 1, т.к.*

*# Функция активации LSTM – сигмоида – работает только с этим*

*# диапазоном данных*X = X / float(n\_vocab)

*# Кодируем каждый символ в one-hot: нулевой вектор длины словаря*

*# где единственная 1 находится на позиции закодированного символа*

*# к примеру, [0,…0,1] для z (выходной символ)*y = np\_utils.to\_categorical(dataY)

*# Создаем и компилируем модель*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'**)

*# Так как нейросеть обучается долго, создаем контрольные точки,*

*# чтобы веса сохранялись каждую эпоху (только если было улучшение)*filepath=**"weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"**checkpoint = ModelCheckpoint(filepath, monitor=**'loss'**, verbose=1, save\_best\_only=**True**, mode=**'min'**)  
callbacks\_list = [checkpoint]

*# Запускаем обучение*model.fit(X, y, epochs=20, batch\_size=128, callbacks=callbacks\_list)

В итоге получаем 144271 образец для обучения, 45 символов в словаре (не очищена пунктуация). Нейронная сеть обучается на кластере полтора часа, поэтому в лабораторной работе используются **предобученные веса.**

## 2.1.2 Код для генерации текста (простая модель)

Исходный код для обучения данной модели находится в: labs/lab4/test\_simple.py. Рассмотрим его фрагмент подробнее.

# Загружаем предобученные веса. Для этого скопируйте

# из директории название файла весов с **минимальной** ошибкой

# В данном случае минимальная ошибка была на 19 эпохе = 1.9588  
filename = **"weights-improvement-19-1.9588.hdf5"**model.load\_weights(filename)  
model.compile(loss=**'categorical\_crossentropy'**, optimizer=**'adam'**)

# Берем случайные 100 символов из книги в качестве входной строки

# которую модель должна продолжить  
start = numpy.random.randint(0, len(dataX)-1)  
pattern = dataX[start]  
print(**"Seed:"**)  
print(**"\""**, **''**.join([int\_to\_char[value] **for** value **in** pattern]), **"\""**)

# Предсказываем на их основе следующие 1000 символов по одному за # итерацию цикла  
**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."**)

## 2.1.3 Практические задания

Все скрипты необходимо запускать на 3 питоне.

1. Сгенерировать несколько текстовых последовательностей длиной 1000 символов, оценить результат. Привести в отчете примеры последовательностей.
2. Выделить в сгенерированных примерах слова, которые были хорошо усвоены нейронной сетью (сгенерированы без ошибок). Привести результаты в отчете.
3. Замерить время генерации текстовой последовательности длиной 1000 на CPU и GPU, оценить ускорение. Результаты усреднить по не менее чем 10 прогонам. Привести результаты в отчете.
4. Сгенерировать несколько текстовых последовательностей меньшей длины (128 +- 40), оценить результат. Привести в отчете примеры последовательностей.

## 2.2 Использование улучшенной LSTM сети для генерации текста

## 2.2.1 Код для обучения сети

Исходный код для данной модели находится в: labs/lab4/bigger\_model.py. Рассмотрим его подробнее.

Чтобы получить более осмысленный результат, улучшим архитектуру нейронной сети. Увеличим ее глубину, добавив еще одну пару слоев LSTM – Dropout с такими же параметрами (рисунок 2). Это должно помочь нейронной сети обрабатывать текст на более абстрактном уровне, а значит, генерировать слова более точно.

Изменения в коде обучения будут незначительны. Необходимо только указать, что первый слой LSTM должен возвращать последовательности, так как слой, следующий за ним, также работает с последовательностями:

model.add(LSTM(256, input\_shape=(X.shape[1], X.shape[2]),

**return\_sequences=True**))

Добавить два новых слоя:

**model.add(LSTM(256))**

**model.add(Dropout(0.2))**

Также значительно уменьшим размер batch, чтобы сеть чаще обновляла свои веса и имела большие шансы на качественное обучение. Вместе с этим увеличим количество эпох до 50:

model.fit(X, y, epochs=**50**, batch\_size=**64**,

callbacks=callbacks\_list)

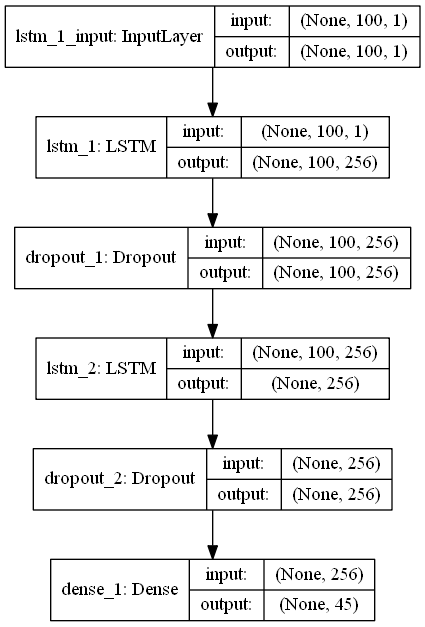


Рисунок 2 – Архитектура улучшенной сети для генерации текста

## 2.2.2 Код для генерации текста

Код для генерации текста останется таким же, как и в разделе 3.2, поменяется только раздел создания модели: в него будут добавлены две новые строки и новый атрибут:

model = Sequential()

model.add(LSTM(256, input\_shape=(X.shape[1], X.shape[2]), **return\_sequences=True**))

model.add(Dropout(0.2))

**model.add(LSTM(256))**

**model.add(Dropout(0.2))**

model.add(Dense(y.shape[1], activation='softmax'))

Также изменится название файла предобученных весов: к нему в конце добавлена пометка –bigger, например:

weights-improvement-50-1.1974-bigger.hdf5

Данная модель была обучена как для генерации текста на английском (использовалась книга «Алиса в Стране чудес»), так и на русском («Убийство в Восточном экспрессе»). Первоначальный код подготовлен для генерации слов английского языка. Для изменения языка генерации нужно:

1. заменить название файла, из которого берем фразы для примера, которые модель должна продолжить:

filename = "express.txt" # Для английского – wonderland.txt

1. заменить файл предобученных весов на:

rus-weights-improvement-50-…-bigger.hdf5

Для улучшения результатов обучения, модель для генерации русских слов обучалась на примерах с заранее очищенной пунктуацией.

## 2.2.3 Практические задания

1. Сгенерировать несколько текстовых последовательностей длиной 1000 символов (на русском или английском), оценить результат. Привести в отчете примеры последовательностей.
2. Выделить в сгенерированных примерах осмысленные словосочетания. Привести результаты в отчете.
3. Замерить время генерации текстовой последовательности длиной 1000 символов на CPU и GPU, оценить ускорение. Результаты усреднить по не менее чем 10 прогонам. Привести результаты в отчете.
4. Сгенерировать несколько текстовых последовательностей меньшей длины (128 +- 40), оценить результат. Привести результаты в отчете.
5. Используя предобученные веса, сравнить текстовые последовательности длиной 200, генерируемые нейронной сетью, если ее обучать: 1, 20, 40, 50 эпох (файлы весов, соответственно: weights-improvement-**1**-…-bigger, weights-improvement-**20**-…-bigger, weights-improvement-**40**-…-bigger …) Привести результаты в отчете.
6. Изменить программу таким образом, чтобы она продолжала строку, длиной >100 символов, введенную пользователем, сгенерировать пример. Привести результаты в отчете.