10 Генеративное глубокое обучение

В этой главе

- Что такое генеративное глубокое обучение, области его использования и отличия от обсуждавшихся ранее задач глубокого обучения.
- Генерация текста с помощью RNN.
- Латентное пространство как основа для генерации новых изображений на примере вариационного автокодировщика.
- Основы генеративных состязательных сетей (GAN).

Среди наиболее впечатляющих результатов работы глубоких нейронных сетей — генерация правдоподобно выглядящих/звучащих изображений и звуков. В настоящее время глубокие нейронные сети способны создавать чрезвычайно реалистичные изображения человеческих лиц[1], синтезировать естественно звучащую речь[2], писать вполне связные тексты[3], и это далеко не все. Подобные *генеративные* (generative)[4]

---

1 *Karras T., Laine S., Aila T.* A Style-Based Generator Architecture for Generative Adversarial Networks // submitted 12 Dec. 2018. https://arxiv.org/abs/1812.04948. См. демонстрацию по адресу https://thispersondoesnotexist.com/.

2 *Oord A. van den, Dieleman S.* WaveNet: A Generative Model for Raw Audio // blog, 8 Sept. 2016. http://mng.bz/MOrn.

3 Better Language Models and Their Implications // OpenAI, 2019. https://openai.com/blog/better-language-models/.

4 В русскоязычной литературе часто называются также «порождающими». — *Примеч. пер.*

модели могут пригодиться для множества целей, включая помощь в художественном творчестве, модификацию существующего контента в зависимости от определенных условий и дополнение существующих наборов данных для решения других задач глубокого обучения[1].

Помимо чисто практических приложений, например наведения лоска на селфи потенциального покупателя косметики, генеративные модели заслуживают внимания и по теоретическим причинам. Генеративное и дискриминативное моделирование — два принципиально различных типа моделей в машинном обучении. Все встречавшиеся нам до сих пор в книге модели были *дискриминативными* (discriminative)[2]. Подобные модели предназначены для отображения входного сигнала в дискретное или непрерывное значение, вне зависимости от того, в ходе какого процесса этот входной сигнал был сгенерирован. Вспомните созданные нами классификаторы фишинговых сайтов, ирисов Фишера, цифр из набора MNIST и звуков речи, а также регрессор для цен на недвижимость. Генеративные модели, напротив, предназначены для математической имитации процесса генерации примеров данных различных классов. Но после усвоения генеративной моделью способа генерации она способна решать и дискриминативные задачи. Таким образом, можно считать, что генеративные модели лучше «понимают» данные, по сравнению с дискриминативными.

В этой главе мы рассмотрим основы глубоких генеративных моделей, предназначенных для текста и изображений. К концу главы вы узнаете, какие идеи лежат в основе использующих RNN моделей языка, автокодировщиков, ориентированных на обработку изображений, и генеративных состязательных сетей. Вы также познакомитесь с паттерном реализации подобных моделей в TensorFlow.js и при необходимости сможете применить их к своим наборам данных.

10.1. Генерация текста с помощью LSTM

Начнем с генерации текста. Для этого воспользуемся RNN, с которыми познакомили вас в предыдущей главе. И хотя описанная далее методика применяется здесь для генерации текста, она вовсе не ограничивается только этой предметной областью. Ее можно приспособить и для генерации прочих типов последовательностей, например музыкальных произведений, — достаточно обеспечить возможность представления музыкальных нот подходящим образом и найти подходящий обучающий набор данных[3]. Аналогичным образом можно приспособить их для генерации росчерков

---

[1] *Antoniou A., Storkey A., Edwards H.* Data Augmentation Generative Adversarial Networks // submitted 12 Nov. 2017. https://arxiv.org/abs/1711.04340.

[2] В русскоязычной литературе иногда встречается название «различающие модели». — *Примеч. пер.*

[3] Например, см. модель Performance-RNN из проекта Magenta компании Google: https://magenta.tensorflow.org/performance-rnn.

пера при рисовании, создавая прекрасные наброски[1] или даже реалистично выглядящие японские иероглифы[2].

10.1.1. Предсказание следующего символа: простой способ генерации текста

Во-первых, давайте сформулируем задачу генерации текста. Пусть в качестве входных обучающих данных дан корпус текстовых данных достаточного размера (хотя бы несколько мегабайт), например полное собрание сочинений Шекспира (в виде очень длинной строки). Необходимо обучить модель генерировать новые тексты, как можно более *похожие* на обучающие данные. Ключевое слово здесь, конечно, «похожие». Пока не будем утруждать себя точным определением этого слова. Его смысл прояснится после демонстрации метода и результатов его работы.

Попробуем сформулировать задачу в парадигме глубокого обучения. В рассмотренном в предыдущей главе примере преобразования форматов дат мы показали, как сгенерировать четко отформатированную выходную последовательность на основе небрежно отформатированной входной. У этой задачи преобразования текста в текст было четко заданное решение: правильная строка с датой в формате ISO-8601. Однако задача генерации текста этим требованиям не отвечает. Никакой явной входной последовательности нет, да и «правильный» выходной сигнал четко не определен: просто требуется сгенерировать нечто «правдоподобно выглядящее». Как же нам поступить?

Решение состоит в создании модели, которая бы предсказывала, какой символ следует за заданной последовательностью символов. Это называется *предсказанием следующего символа* (next-character prediction). Например, получив в качестве входной последовательности строку *Love looks not with the eyes, b*, модель, хорошо обученная на наборе данных текстов Шекспира, должна с высокой степенью вероятности предсказать символ *u*. Впрочем, она предсказывает только один символ. Как же сгенерировать с ее помощью последовательность символов? Очень просто: формируем новую входную последовательность той же длины, что и раньше, сдвигая предыдущую входную последовательность на один символ влево, отбрасывая первый символ и вставляя в конец последовательности только что сгенерированный символ *(u)*. В результате получаем новую входную последовательность для алгоритма предсказания следующего символа, а именно, *ove looks not with the eyes, bu*. По такой входной последовательности модель должна с высокой степенью вероятности предсказать символ *t*. Этот процесс, продемонстрированный на рис. 10.1, можно повторять столько раз, сколько требуется, для генерации последовательности нужной длины. Конечно, при этом требуется начальный фрагмент текста в качестве отправной точки, который можно просто выбрать случайным образом из корпуса текста.

---

[1] Например, см. модель Sketch-RNN, созданную Дэвидом Ха (David Ha) и Дугласом Экком (Douglas Eck): http://mng.bz/omyv.

[2] *Ha D.* Recurrent Net Dreams Up Fake Chinese Characters in Vector Format with TensorFlow // blog, 28 Dec. 2015. http://mng.bz/nvX4.

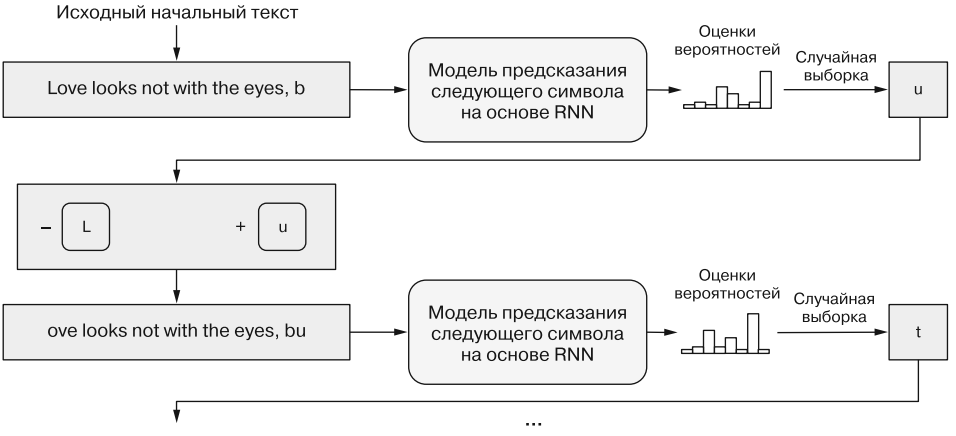


Рис. 10.1. Схематическая иллюстрация возможностей генерации текстовой последовательности по входному фрагменту текста в качестве начального значения с помощью модели предсказания очередного символа на основе RNN. На каждом шаге RNN предсказывает следующий символ по входному тексту. Далее производится конкатенация входного текста с предсказанным следующим символом, а первый символ текста отбрасывается. Полученный результат служит входным сигналом для очередного шага. На каждом шаге RNN выдает оценки вероятностей всех возможных символов из множества. Фактически следующий символ определяется на основе случайной выборки

Подобная формулировка превращает задачу генерации последовательности в задачу классификации на основе последовательности, аналогичную задаче анализа тональностей обзоров из IMDb в главе 9, в которой предсказывался бинарный класс по входному сигналу фиксированной длины. Модель генерации текста, по существу, делает то же самое, только речь идет о многоклассовой классификации с N возможными классами, где N — размер множества символов, а именно, количество всех уникальных символов в наборе текстов.

Подобная формулировка предсказания следующего символа далеко не нова в истории обработки естественного языка и теории вычислительной техники. Клод Шеннон, один из пионеров теории информации, провел эксперимент, в котором участников просили угадать следующую букву по короткому фрагменту текста на английском языке[1]. Благодаря этому эксперименту Шеннон смог оценить среднюю степень неопределенности каждой буквы типичного текста на английском языке для заданного контекста. Эта неопределенность, оказавшаяся равной 1,3 бита энтропии, отражает, какой в среднем объем информации несет каждая английская буква.

Результат в 1,3 бита меньше, чем неопределенность при совершенно случайном наборе из 26 символов: $\log_2(26) = 4{,}7$ бита. Интуитивно это понятно, поскольку в английском языке порядок букв случайным не бывает, буквы следуют определенным закономерностям. Если рассматривать на самом низком уровне, только определенные последовательности букв являются допустимыми английскими словами.

[1] Первоисточник — статью 1951 года — можно найти по адресу http://mng.bz/5AzB.

На более высоком уровне, только слова в определенном порядке удовлетворяют требованиям английской грамматики. А на еще более высоком уровне лишь небольшое подмножество грамматически правильных предложений является осмысленным.

Если задуматься, именно в этом и состоит наша задача генерации текста: усвоить закономерности на всех уровнях. По существу, наша модель обучается делать именно то, что делали участники исследования Шеннона, — угадывать следующий символ. А теперь взглянем на код этого примера и обсудим, как он работает. Запомните результат Шеннона (1,3 бита), поскольку мы еще вернемся к нему.

10.1.2. Пример lstm-text-generation

Пример lstm-text-generation из репозитория tfjs-examples включает обучение основанной на LSTM модели предсказания следующего символа и генерацию с ее помощью нового текста. Оба этапа — обучения и генерации — выполняются на JavaScript с помощью TensorFlow.js. Можете запустить этот пример как в браузере, так и в среде прикладной части с помощью Node.js. Первый вариант отличается более наглядным и интерактивным интерфейсом, но скорость обучения во втором — выше.

Для выполнения примера в браузере выполните следующие команды:

```
git clone https://github.com/tensorflow/tfjs-examples.git
cd tfjs-examples/lstm-text-generation
yarn && yarn watch
```

На открывшейся веб-странице можно выбрать и загрузить для обучения модели один из четырех предлагаемых текстовых наборов данных. В дальнейшем обсуждении используется набор данных текстов Шекспира. После загрузки данных можно создать для них модель, нажав кнопку Create Model (Создать модель). В текстовом поле ввода можно указать количество нейронов создаваемого LSTM-слоя. По умолчанию оно равно 128, но вы можете поэкспериментировать с другими значениями, например 64. Если ввести несколько чисел, разделенных запятыми (например, `128,128`), будет создана модель с несколькими LSTM-слоями, один поверх другого.

Для обучения в прикладной части с помощью модулей tfjs-node или tfjs-node-gpu воспользуйтесь командой `yarn train` вместо `yarn watch`:

```
yarn train shakespeare \
      --lstmLayerSize 128,128 \
      --epochs 120 \
      --savePath ./my-shakespeare-model
```

При наличии настроенного должным образом GPU с поддержкой CUDA можете добавить в эту команду флаг `--gpu`, чтобы выполнять обучение на GPU с существенно более высокой скоростью. Флаг `--lstmLayerSize` играет ту же роль, что и текстовое поле, куда нужно ввести размер LSTM в браузерной версии примера. Приведенная выше команда создает и обучает модель, состоящую из двух LSTM-слоев, каждый с 128 нейронами, размещенными один поверх другого.

У обучаемой здесь модели многоярусная LSTM-архитектура. Что означает размещение LSTM-слоев *друг поверх друга* (stacking)? По существу, подобно тому, как

размещение друг над другом плотных слоев в MLP расширяет разрешающие возможности модели, так и размещение друг над другом нескольких LSTM-слоев дает возможность многоэтапного преобразования представлений seq2seq входной последовательности, перед тем как она будет преобразована в завершающем LSTM-слое в итоговый результат регрессии или классификации. Данная архитектура схематически изображена на рис. 10.2. Важно отметить, что свойство `returnSequence` первого LSTM-слоя равно `true`, а потому он генерирует выходную последовательность, включающую выходной сигнал для всех отдельных элементов входной последовательности. Это позволяет подать выходной сигнал первого LSTM-слоя на вход второго, ведь LSTM-слой ожидает последовательные входные данные, а не входной сигнал из одного элемента.

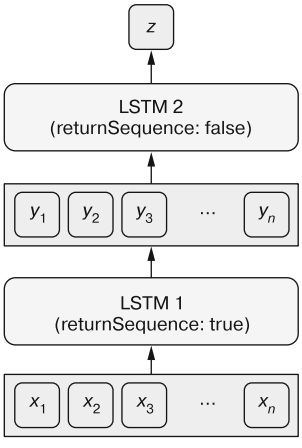
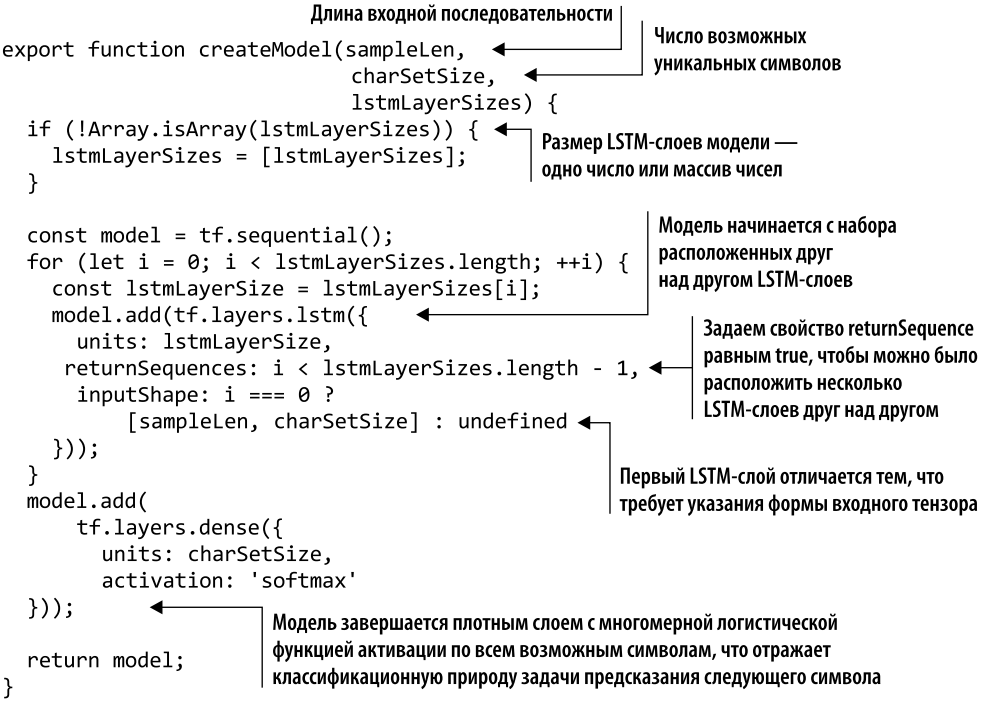


Рис. 10.2. Размещение в модели нескольких (в данном случае двух) LSTM-слоев друг над другом. Свойство returnSequence первого LSTM-слоя равно true, а потому он генерирует на выходе последовательность элементов. Последовательный выходной сигнал первого LSTM-слоя подается на вход второго. Второй LSTM-слой выдает на выходе не последовательность элементов, а один элемент, который может быть регрессионным предсказанием или массивом вероятностей из многомерной логистической функции. При этом формируется итоговый выходной сигнал модели

Листинг 10.1 содержит код, создающий модели предсказания следующего символа с архитектурой, приведенной на рис. 10.2 (отрывок из файла `lstm-text-generation/model.js`). Учтите, что, в отличие от схемы на рисунке, данный код включает плотный слой в качестве выходного слоя модели, с многомерной логистической функцией активации. Напомним, что многомерная логистическая функция активации нормализует выходные сигналы до значений от 0 до 1, в сумме равных 1, подобно распределению вероятности. Таким образом, выходной сигнал завершающего плотного слоя представляет собой предсказанные вероятности уникальных символов.

Аргумент `lstmLayerSize` функции `createModel()` служит для управления количеством LSTM-слоев и их размерами. Входную форму первого LSTM-слоя определяют настройки `sampleLen` (количество символов, получаемых моделью за один раз) и `charSetSize` (количество уникальных символов, содержащихся в текстовых данных). Для выполняемого в браузере примера `sampleLen` жестко задан равным 40; в обучающем сценарии на основе Node.js его можно задавать через флаг `--sampleLen`. Для набора данных текстов Шекспира `charSetSize` равен 71. Этот набор символов включает английские буквы в нижнем и верхнем регистрах, знаки препинания, пробел, разрыв строки и еще несколько специальных символов. При таких параметрах форма входного тензора модели, создаваемой функцией из листинга 10.1 — `[40, 71]` (не считая измерения батчей). Эта форма соответствует 40 символам в унитарном представлении. Форма выходного тензора модели — `[71]` (опять же не считая измерения батчей) — значение вероятностей многомерной логистической функции для 71 возможного варианта следующего символа.

Листинг 10.1. Создание многослойной LSTM-модели для предсказания следующего символа




```
export function createModel(sampleLen,
                             charSetSize,
                             lstmLayerSizes) {
  if (!Array.isArray(lstmLayerSizes)) {
    lstmLayerSizes = [lstmLayerSizes];
  }

  const model = tf.sequential();
  for (let i = 0; i < lstmLayerSizes.length; ++i) {
    const lstmLayerSize = lstmLayerSizes[i];
    model.add(tf.layers.lstm({
      units: lstmLayerSize,
     returnSequences: i < lstmLayerSizes.length - 1,
      inputShape: i === 0 ?
          [sampleLen, charSetSize] : undefined
    }));
  }
  model.add(
      tf.layers.dense({
        units: charSetSize,
        activation: 'softmax'
  }));

  return model;
}
```

Для подготовки модели к обучению скомпилируем ее, взяв в качестве функции потерь категориальную перекрестную энтропию, ведь наша модель, по существу, представляет собой классификатор с 71 классом. В качестве оптимизатора воспользуемся RMSProp, часто применяемым в рекуррентных моделях:

```
const optimizer = tf.train.rmsprop(learningRate);
model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});
```

Данные, на которых будет обучаться модель, состоят из пар входных отрывков текста и символов, следующих за ними, закодированных в унитарные векторы (см. рис. 10.1). Описанный в файле `lstm-text-generation/data.js` класс `TextData` включает логику генерации подобных тензоров данных на основе обучающего корпуса текстов. Код здесь довольно громоздкий, но сама идея проста, она состоит в случайной выборке из очень длинной строки фрагментов фиксированной длины и преобразовании их в унитарные тензорные представления.

Если вы используете веб-версию примера, можете подобрать гиперпараметры в разделе Model Training (Обучение модели) веб-страницы, а именно: количество эпох обучения, число примеров данных из расчета на эпоху, скорость обучения и т. д. Нажмите кнопку Train Model (Обучить модель) для запуска процесса обучения модели. При обучении же с помощью Node.js эти гиперпараметры настраиваются с помощью флагов командной строки. Подробнее вы можете узнать из справочных сообщений, набрав команду `yarn train --help`.

В зависимости от заданного вами количества эпох обучения и размера модели обучение займет от нескольких минут до нескольких часов. При обучении в Node.js после каждой эпохи автоматически выводится число примеров фрагментов текста, сгенерированных моделью (табл. 10.1). По мере обучения значение потерь должно неуклонно снижаться и уменьшиться от начального значения 3,2 примерно до 1,4–1,5. Примерно после 120 эпох по мере снижения значений потерь качество сгенерированного текста улучшится до такой степени, что к концу обучения он будет выглядеть *подобным на шекспировский*, а потери на проверочном наборе данных достигнут 1,5 — не так уж далеко от неопределенности 1,3 бита/символ из эксперимента Шеннона. Но учтите, что при такой парадигме обучения и разрешающих возможностях модели сгенерированный ею текст никогда не станет похож на настоящие тексты Шекспира.

Таблица 10.1. Примеры текста, сгенерированные основанной на LSTM моделью предсказания следующего символа. Для генерации текста сначала задается начальный текст: « in hourly synod about thy particular prosperity, and lo»[1]. Для сравнения приведем фактический текст, следующий в пьесе за нашим начальным текстом: «ve thee no worse than thy old father Menenius does! ...»

Эпох обучения	Потери на проверочном наборе	T = 0	T = 0.25	T = 0.5	T = 0.75
5	2,44	"rle the the the the the the the the the the the the the the the the the the the the the the the the the the the "	"te ans and and and and and warl torle an at an yawl and tand and an an ind an an in thall ang ind an tord and and and wa"	"te toll nlatese ant ann, tomdenl, teurteeinlndting fall ald antetetell linde ing thathere taod winld mlinl theens tord y"	"p, af ane me pfleh; fove this? Iretltard efidestind ants anl het insethou loellr ard,
25	1,96	"ve tray the stanter an truent to the stanter to the stanter to the stanter to the stanter to the stanter to the stanter "	"ve to the enter an truint to the surt an truin to me truent me the will tray mane but a bean to the stanter an trust tra"	"ve of marter at it not me shank to an him truece preater the beaty atweath and that marient shall me the manst on hath s"	"rd; not an an beilloters An bentest the like have bencest on it love gray to dreath avalace the lien I am sach me, m"

Продолжение ⇩

[1] Отрывок из пьесы «Кориолан» Шекспира, акт пятый, сцена вторая. Обратите внимание, что пример текста включает разрывы строк и обрывается посередине слова (love).

Таблица 10.1 *(продолжение)*

Эпох обучения	Потери на проверочном наборе	T = 0	T = 0.25	T = 0.5	T = 0.75
50	1,67	"rds the world the world the world the world the world the world the world the world the world the worl"	"ngs they are their shall the englents the world the world the stand the provicess their string shall the world I"	"nger of the hath the forgest as you for sear the device of thee shall, them at a hame, The now the would have bo"	"ngs, he coll, As heirs to me which upon to my light fronest prowirness foir. I be chall do vall twell. SIR C"
100	1,61	"nd the sough the sought That the more the man the forth and the strange as the sought That the more the man the "	"nd the sough as the sought In the consude the more of the princes and show her art the compont "	"rds as the manner. To the charit and the stranger and house a tarron. A tommern the bear you art this a contents, "	"nd their conswents That thou be three as me a thout thou do end, The longers and an heart and not strange. A G"
120	1,49	"ve the strike the strike the strike the strike the strikes the strike And the strike the strike the strike A"	"ve the fair brother, And this in the strike my sort the strike, The strike the sound in the dear strike And "	"ve the stratter for soul. Monty to digning him your poising. This for his brother be this did fool. A mock'd"	"ve of his trusdum him. poins thinks him where sudy's such then you; And soul they will I would from in my than s"

В табл. 10.1 показаны примеры текстов при четырех различных *значениях температуры* — параметра, определяющего степень случайности сгенерированного текста. Возможно, вы заметили в примерах сгенерированного текста, что при низких значениях температуры текст выглядит более однообразным и менее естественным, в то время как бóльшим значениям соответствует менее предсказуемый текст. По умолчанию максимальное значение температуры, демонстрируемое Node.js-сценарием обучения, равно 0,75, в результате чего иногда получаются последовательности символов, напоминающие английские слова, но на самом деле ими не являющиеся (например, *stratter* и *poins* в приведенных в таблице примерах). Далее вы узнаете, как температура работает и почему так называется.