Липецкий государственный технический университет

Кафедра прикладной математики

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ МАТЕМАТИЧЕСКИХ ИССЛЕДОВАНИЙ

Лекция 14

Deep Learning в R

Составитель - Сысоев А.С., к.т.н., доцент

Липецк - 2022

Outline

16.1. Модели глубокого обучения для обработки текста

16.2. Рекуррентные нейронные сети

16.3. Генерация последовательности данных

16.4. Реализация посимвольной генерации текста на основе LSTM

16.1. Модели глубокого обучения для обработки текста

Как любые другие нейронные сети, модели глубокого обучения не могут принимать на входе простой текст: они работают только с числовыми тензорами. Векторизация текста — это процесс преобразования текста в числовые тензоры.

Способы:

- Разбить текст на слова и преобразовать каждое слово в вектор.
- Разбить текст на символы и преобразовать каждый символ в вектор.
- Извлечь n-граммы из слов или символов и преобразовать каждую n-грамму в вектор. N-граммы — это перекрывающиеся группы из нескольких последовательных слов или символов.

Собирательно разные единицы, на которые можно разбить текст (слова, символы или n-граммы) называют токенами, а разбиение текста на такие токены называют токенизацией.

16.1. Модели глубокого обучения для обработки текста

Прямое кодирование токенов.
Векторное кодирование токенов.

```
{"The", "The cat", "cat", "cat sat", "sat",
"sat on", "on", "on the", "the", "the mat", "mat"}
```

```
{"The", "The cat", "cat", "cat sat", "The cat sat",
"sat", "sat on", "on", "cat sat on", "on the", "the",
"sat on the", "the mat", "mat", "on the mat"}
```

16.1. Модели глубокого обучения для обработки текста

Прямое кодирование на уровне слов (упрощенный пример)

Создание индекса всех лексем в данных

Исходные данные: один элемент – один образец (в данном случае образцы представляют по одному предложению, но точно так же это могли бы быть целые документы)

Токенизация образцов с помощью функции strsplit. В действующем приложении также желательно было бы удалить знаки пунктуации и специальные символы

Присваивает уникальный индекс каждому уникальному слову. Обратите внимание, что индекс со значением 1 не используется

```
samples <- c("The cat sat on the mat.",
"The dog ate my homework.")

token_index <- list()
for (sample in samples)
  for (word in strsplit(sample, " ")[[1]])
    if (!word %in% names(token_index))
      token_index[[word]] <- length(token_index) + 2

max_length <- 10

results <- array(0, dim = c(length(samples),
                            max_length,
                            max(as.integer(token_index))))

for (i in 1:length(samples)) {
  sample <- samples[[i]]
  words <- head(strsplit(sample, " ")[[1]], n = max_length)
  for (j in 1:length(words)) {
    index <- token_index[[words[[j]]]]
    results[[i, j, index]] <- 1
  }
}
```

Векторизация образцов. В каждом образце рассматриваются только первые max_length слов

Здесь сохраняются результаты

16.1. Модели глубокого обучения для обработки текста

Прямое кодирование на уровне символов (упрощенный пример)

```
samples <- c("The cat sat on the mat.", "The dog ate my homework.")

ascii_tokens <- c("", sapply(as.raw(c(32:126)), rawToChar))
token_index <- c(1:(length(ascii_tokens)))
names(token_index) <- ascii_tokens

max_length <- 50

results <- array(0, dim = c(length(samples), max_length, length(token_
index)))

for (i in 1:length(samples)) {
  sample <- samples[[i]]
  characters <- strsplit(sample, "")[[1]]
  for (j in 1:length(characters)) {
    character <- characters[[j]]
    results[i, j, token_index[[character]]] <- 1
  }
}
```

- Прием прямого кодирования имеет разновидность — так называемое прямое хеширование признаков (one-hot hashing trick), — которое можно использовать, когда словарь содержит слишком большое количество токенов, чтобы его можно было использовать явно.
- Вместо явного присваивания индекса каждому слову и сохранения ссылок на эти индексы в словаре можно хешировать слова в векторы фиксированного размера.
- **Главное достоинство** этого метода — отсутствие необходимости хранить индексы слов, что позволяет сэкономить память и кодировать данные по мере необходимости (векторы токенов можно генерировать сразу же, по мере их обхода, до просмотра всех имеющихся данных).
- **Единственный недостаток** — этот метод восприимчив к хеш-коллизиям: два разных слова могут получить одинаковые хеш-значения, и впоследствии любая модель машинного обучения не сможет различить эти слова. *Вероятность хеш-коллизий снижается, когда размер пространства хеширования намного больше общего количества уникальных токенов, подвергаемых хешированию*.

16.1. Модели глубокого обучения для обработки текста

Прямое кодирование на уровне слов с использованием хеширования (упрощенный пример)

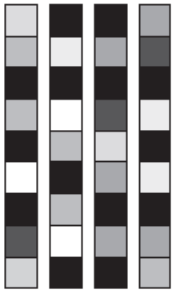
```
library(hashFunction)

samples <- c("The cat sat on the mat.", "The dog ate my homework.")

dimensionality <- 1000

max_length <- 10

results <- array(0, dim = c(length(samples), max_length, dimensionality))

for (i in 1:length(samples)) {
  sample <- samples[[i]]
  words <- head(strsplit(sample, " ")[[1]], n = max_length)
  for (j in 1:length(words)) {
    index <- abs(spooky.32(words[[i]])) %% dimensionality
    results[[i, j, index]] <- 1
  }
}
```

Слова будут сохраняться как векторы с размером 1000. Если число слов близко к 1000 (или даже больше), вы увидите множество хеш-коллизий, снижающих точность этого метода кодирования

Хеширование слов в случайные целочисленные индексы между 0 и 1000

ИСПОЛЬЗОВАНИЕ ВЕКТОРНОГО ПРЕДСТАВЛЕНИЯ СЛОВ

Векторы, полученные прямым кодированием:
— разреженные
— с большим числом размерностей
— негибкие



Векторные представления:
— плотные
— малоразмерные
— конструируются на основе данных

- В отличие от векторов, полученных прямым кодированием, векторные представления слов конструируются из данных.
- Векторное представление слов позволяет уместить больший объем информации в меньшее число измерений.

Получить векторные представления слов можно двумя способами:

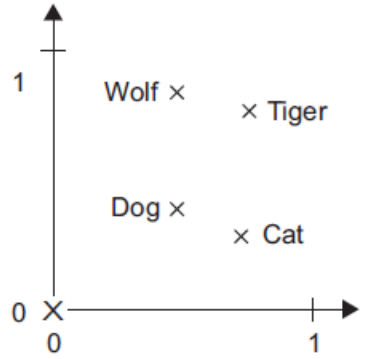
- *Конструировать векторные представления в процессе решения основной задачи* (такой, как классификация документа или определение эмоциональной окраски). В этом случае изначально создаются случайные векторы слов, которые затем постепенно конструируются (обучаются), как это происходит с весами нейронной сети.
- *Загрузить в модель векторные представления, полученные с использованием другой задачи машинного обучения, отличной от решаемой.* Такие представления называют предварительно обученными векторными представлениями слов.

16.1. Модели глубокого обучения для обработки текста

КОНСТРУИРОВАНИЕ ВЕКТОРНЫХ ПРЕДСТАВЛЕНИЙ СЛОВ С ПОМОЩЬЮ СЛОЯ ОТОБРАЖЕНИЯ В ВЕКТОРЫ

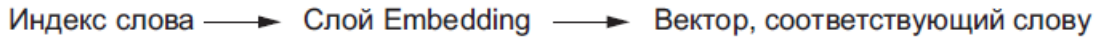
Простейший способ связать плотный вектор со словом — выбрать случайный вектор. Однако могут возникать проблемы:

- слова *accurate* и *exact* могут в конечном счете получить совершенно разные векторные представления, даже при том, что в большинстве случаев они взаимозаменяемы;
- глубокой нейронной сети трудно будет понять такое искаженное, неструктурированное пространство векторов;
- геометрические отношения между векторами слов должны отражать семантические связи между соответствующими им словами.



Создание уровня отображения в векторы

```
embedding_layer <- layer_embedding(input_dim = 1000, output_dim = 64)
```



Уровень отображения в векторы получает на входе двумерный тензор с целыми числами и с формой (`образцы, длина_последовательности`), каждый элемент которого является последовательностью целых чисел. Он может работать с последовательностями разной длины.

Этот уровень возвращает трехмерный тензор с вещественными числами и с формой (`образцы, длина_последовательности, размерность_векторного_представления`).

При создании уровня отображения в векторы его веса (внутренний словарь векторов лексем) инициализируются случайными значениями, как в случае с любым другим уровнем. *В процессе обучения векторы слов постепенно корректируются посредством обратного распространения ошибки, и пространство превращается в структурированную модель, пригодную к использованию.* После полного обучения пространство векторов приобретет законченную структуру, специализированную для решения конкретной задачи.

16.1. Модели глубокого обучения для обработки текста

Загрузка данных из IMDB для передачи в слой отображения в векторы

```
max_features <- 10000
maxlen <- 20

imdb <- dataset_imdb(num_words = max_features)
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb

x_train <- pad_sequences(x_train, maxlen = maxlen)
x_test <- pad_sequences(x_test, maxlen = maxlen)
```

Количество слов, рассматриваемых как признаки

Обрезать текст после этого количества слов (в числе max_features самых распространенных слов)

Загрузить данные как списки целых чисел

Преобразовать списки целых чисел в двумерный тензор с целыми числами и с формой (образцы, максимальная_длина)

Использование уровня отображения в векторы и классификатора данных из IMDB

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 8,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

summary(model)

history <- model %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 32,
  validation_split = 0.2
)
```

Добавляет классификатор сверху

Преобразование трехмерного тензора векторов в двумерный тензор с формой (образцы, максимальная_длина * 8)

Определяет максимальную длину входа для уровня отображения в векторы, чтобы потом можно было уменьшить размерность. После уровня отображения в векторы активация имеет форму (образцы, максимальная_длина, 8)

16.1. Модели глубокого обучения для обработки текста

ИСПОЛЬЗОВАНИЕ ПРЕДВАРИТЕЛЬНО ОБУЧЕННЫХ ВЕКТОРНЫХ ПРЕДСТАВЛЕНИЙ СЛОВ

Вместо обучения векторного представления совместно с решением задачи можно загрузить предварительно сформированные векторные представления, хорошо организованные и обладающие полезными свойствами. Есть смысл повторно использовать признаки, выделенные в ходе решения другой задачи.

ПРИМЕР с IMDB

Загрузка и обработка меток

```
imdb_dir <- "~/Downloads/aclImdb"
train_dir <- file.path(imdb_dir, "train")

labels <- c()
texts <- c()

for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(train_dir, label_type)
  for (fname in list.files(dir_name, pattern = glob2rx("*.txt"),
                           full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}
```

16.1. Модели глубокого обучения для обработки текста

```
library(keras)
maxlen <- 100                 # Отсекать остаток отзывов после 100-го слова
training_samples <- 200       # Размер обучающей выборки 200 образцов
validation_samples <- 10000   # Размер проверочной выборки 10 000 образцов
max_words <- 10000            # Рассматривать только 10 000 наиболее часто используемых слов

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)
  sequences <- texts_to_sequences(tokenizer, texts)

  word_index = tokenizer$word_index
  cat(«Found», length(word_index), "unique tokens.\n")

  data <- pad_sequences(sequences, maxlen = maxlen)

  labels <- as.array(labels)
  cat("Shape of data tensor:", dim(data), "\n")
  cat('Shape of label tensor:', dim(labels), "\n")

  indices <- sample(1:nrow(data))
  training_indices <- indices[1:training_samples]
  validation_indices <- indices[(training_samples + 1):
                                (training_samples + validation_samples)]
  x_train <- data[training_indices,]
  y_train <- labels[training_indices]
  x_val <- data[validation_indices,]
  y_val <- labels[validation_indices]
```

Разбивает данные на обучающую и проверочную выборки, но перед этим перемешивает их, потому что отзывы в исходном наборе упорядочены (сначала следуют отрицательные, а потом положительные)

16.1. Модели глубокого обучения для обработки текста

Предварительная обработка векторных представлений

```
glove_dir = "~/Downloads/glove.6B"
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt"))

embeddings_index <- new.env(hash = TRUE, parent = emptyenv())
for (i in 1:length(lines)) {
  line <- lines[[i]]
  values <- strsplit(line, " ")[[1]]
  word <- values[[1]]
  embeddings_index[[word]] <- as.double(values[-1])
}

cat("Found", length(embeddings_index), "word vectors.\n")

embedding_dim <- 100

embedding_matrix <- array(0, c(max_words, embedding_dim))

for (word in names(word_index)) {
  index <- word_index[[word]]
  if (index < max_words) {
    embedding_vector <- embeddings_index[[word]]
    if (!is.null(embedding_vector))
      embedding_matrix[index+1,] <- embedding_vector
  }
}
```

Словам, отсутствующим в индексе представлений, будут соответствовать векторы с нулевыми значениями

16.1. Модели глубокого обучения для обработки текста

Определение модели

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

summary(model)
```

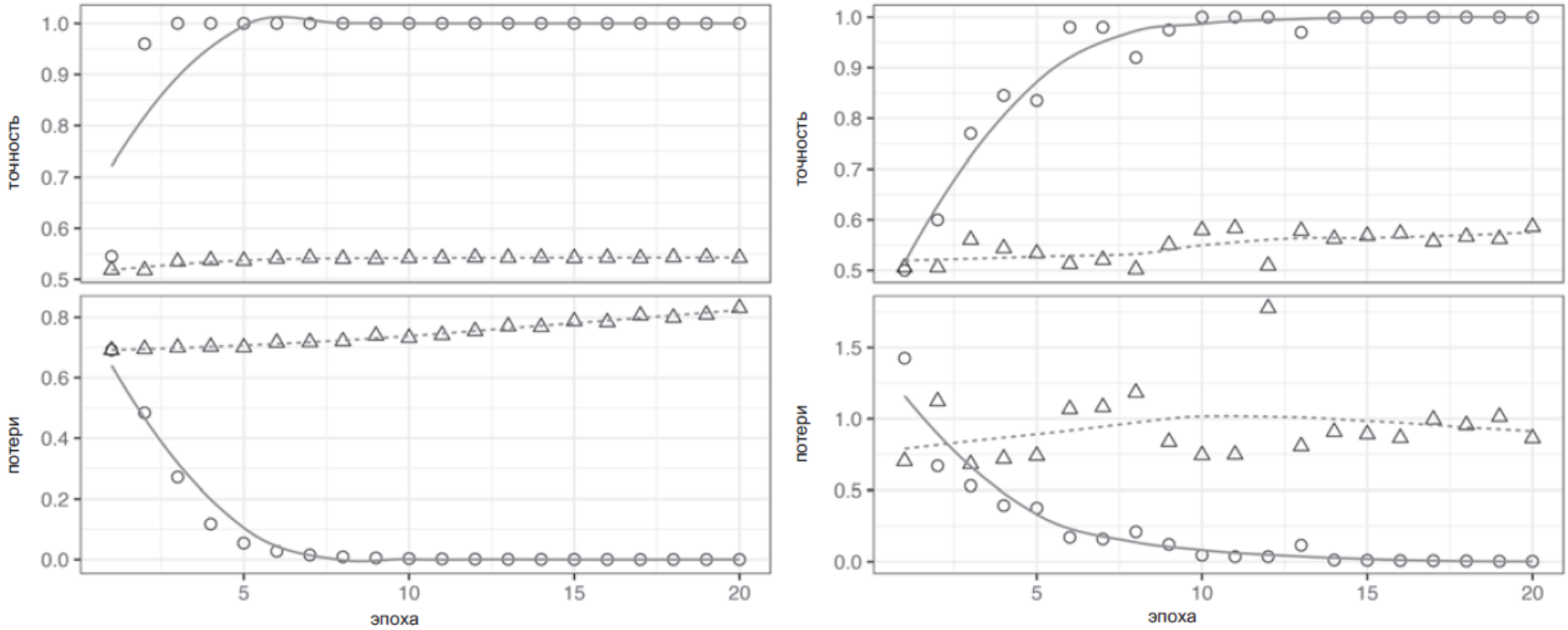
Загрузка предварительно обученных векторных представлений слов в слой отображения в векторы

```
get_layer(model, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()
```

Обучение и оценка

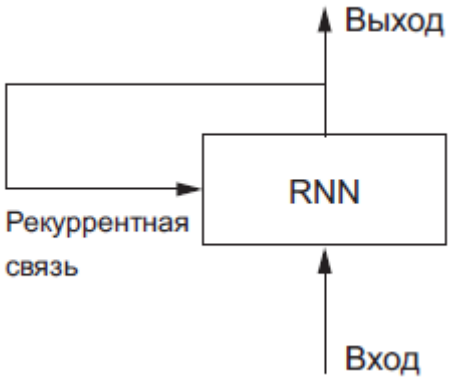
```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
save_model_weights_hdf5(model, "pre_trained_glove_model.h5")
```

16.1. Модели глубокого обучения для обработки текста



16.2. Рекуррентные нейронные сети

- Рекуррентная нейронная сеть (Recurrent Neural Network, RNN) обрабатывает последовательность, перебирая ее элементы и сохраняя состояние, полученное при обработке предыдущих элементов.
- Фактически RNN — это разновидность нейронной сети, имеющей внутренний цикл.
- Сеть RNN сбрасывает состояние между обработкой двух разных независимых последовательностей, поэтому одна последовательность все еще интерпретируется как единый блок данных — единственный входной пакет.
- Блок данных обрабатывается не за один шаг; сеть выполняет внутренний цикл, перебирая последовательность элементов.




```
state_t <- 0
for (input_t in input_sequence) {
  output_t <- activation(dot(W, input_t) + dot(U, state_t) + b)
  state_t <- output_t
}
```

16.2. Рекуррентные нейронные сети

```
timesteps <- 100 ← Число временных интервалов во входной последовательности
input_features <- 32 ← Размерность пространства входных признаков
output_features <- 64 ← Размерность пространства выходных признаков

random_array <- function(dim) {
  array(runif(prod(dim)), dim = dim)
}

inputs <- random_array(dim = c(timesteps, input_features))
state_t <- rep_len(0, length = c(output_features))

W <- random_array(dim = c(output_features, input_features))
U <- random_array(dim = c(output_features, output_features))
b <- random_array(dim = c(output_features, 1))
output_sequence <- array(0, dim = c(timesteps, output_features))
for (i in 1:nrow(inputs)) {
  input_t <- inputs[i,] ← input_t – вектор с формой (входные_признаки)
  output_t <- tanh(as.numeric((W %*% input_t) + (U %*% state_t) + b))
  output_sequence[i,] <- as.numeric(output_t)
  state_t <- output_t
```

Входные данные: случайный шум для простоты примера

Начальное состояние: вектор с нулевыми значениями элементов

Создание матриц со случайными весами

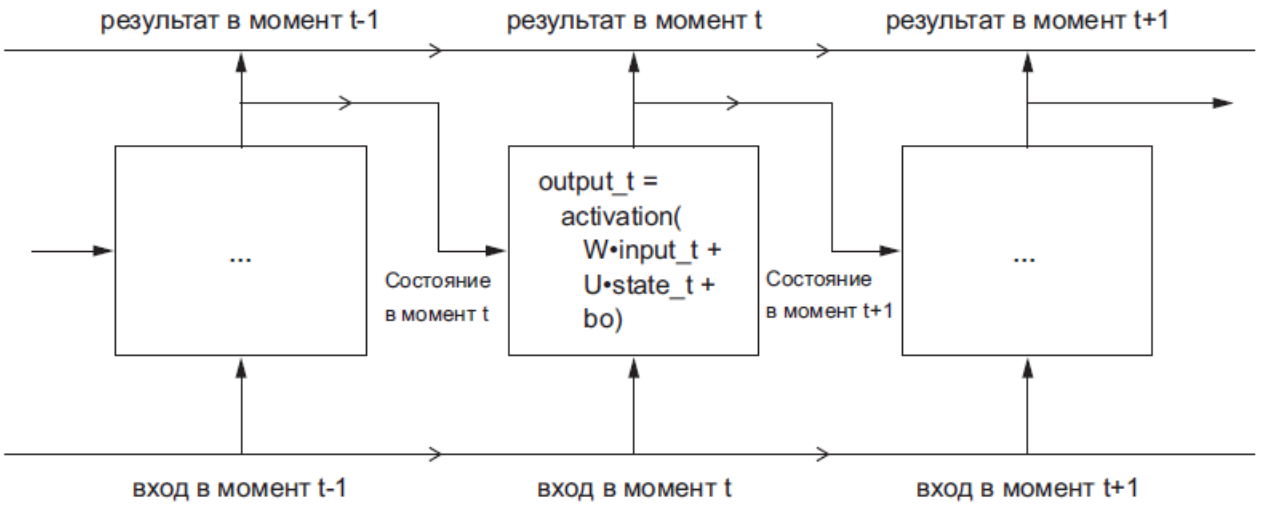
Объединяет входные данные с текущим состоянием (выходными данными на предыдущем шаге)

Обновление матрицы с результатами

Обновление текущего состояния сети как подготовка к обработке следующего временного интервала

16.2. Рекуррентные нейронные сети

```
output_t <- tanh(as.numeric((W %*% input_t) + (U %*% state_t) + b))
```



16.2. Рекуррентные нейронные сети

РЕКУРРЕНТНЫЙ УРОВЕНЬ В KERAS

```
library(keras)
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 32) %>%
  layer_simple_rnn(units = 32)

> summary(model)
```

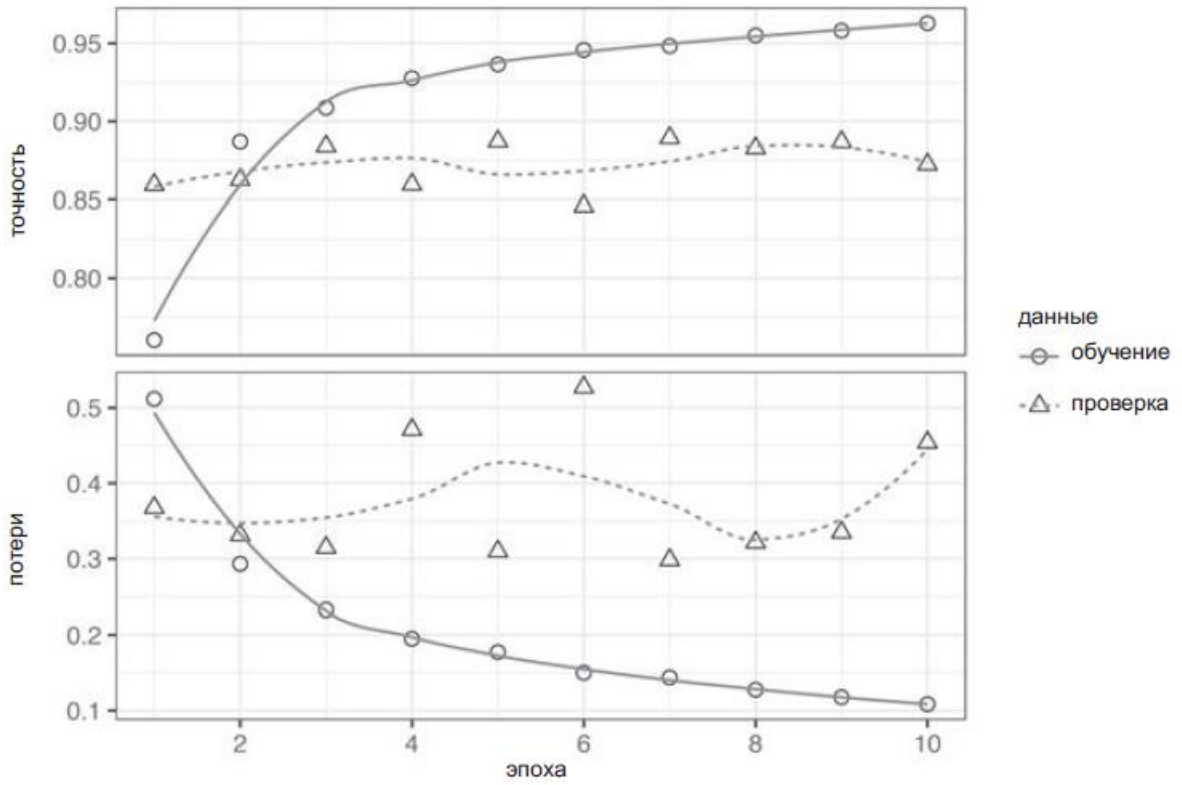
```
_____________________________________________________________________
Layer (type)                 Output Shape              Param #
=====================================================================
embedding_22 (Embedding)     (None, None, 32)          320000
_____________________________________________________________________
simplernn_10 (SimpleRNN)     (None, 32)                2080
=====================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
```

16.2. Рекуррентные нейронные сети

```
library(keras)

max_features <- 10000 ◄── Количество слов, рассматриваемых как признаки
maxlen <- 500 ◄── Обрезать текст после этого количества слов
batch_size <- 32          (в числе max_features самых распространенных слов)

cat("Loading data...\n")
imdb <- dataset_imdb(num_words = max_features)
c(c(input_train, y_train), c(input_test, y_test)) %<-% imdb
cat(length(input_train), "train sequences\n")
cat(length(input_test), "test sequences")

cat("Pad sequences (samples x time)\n")
input_train <- pad_sequences(input_train, maxlen = maxlen)
input_test <- pad_sequences(input_test, maxlen = maxlen)
cat("input_train shape:", dim(input_train), "\n")
cat("input_test shape:", dim(input_test), "\n")

model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_simple_rnn(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history <- model %>% fit(
  input_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2
)
```

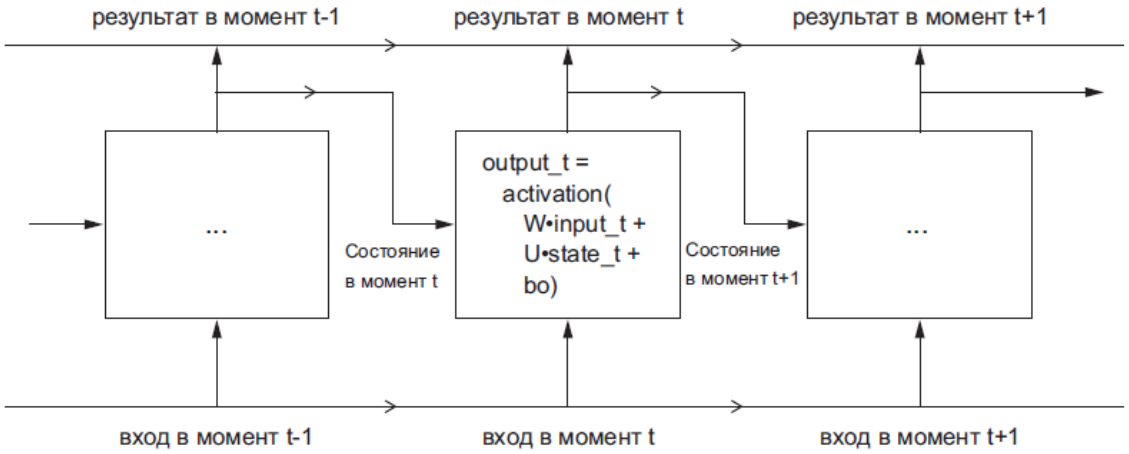
16.2. Рекуррентные нейронные сети

```
plot(history)
```

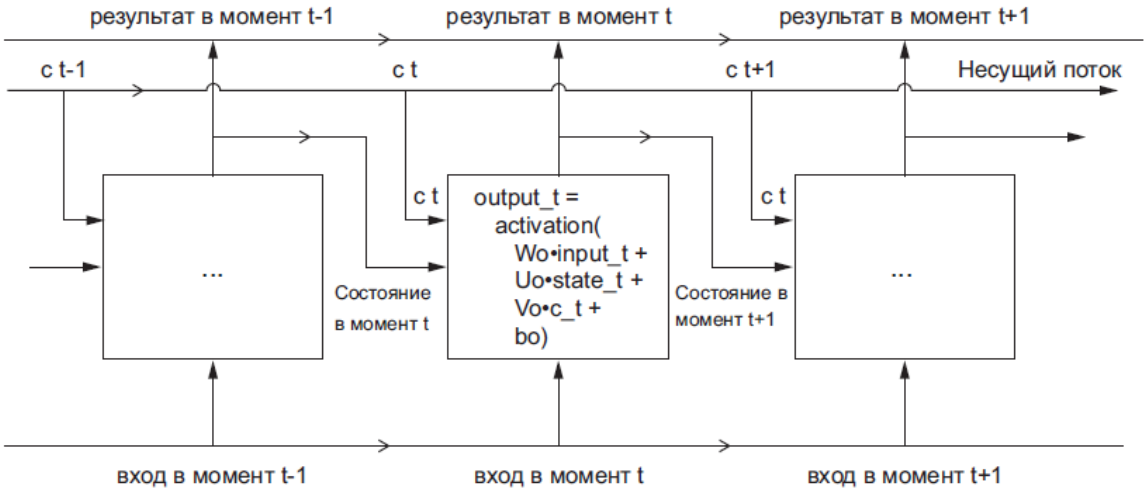


16.2. Рекуррентные нейронные сети

СЛОИ LSTM И GRU



Начальная точка слоя LSTM: слой SimpleRNN

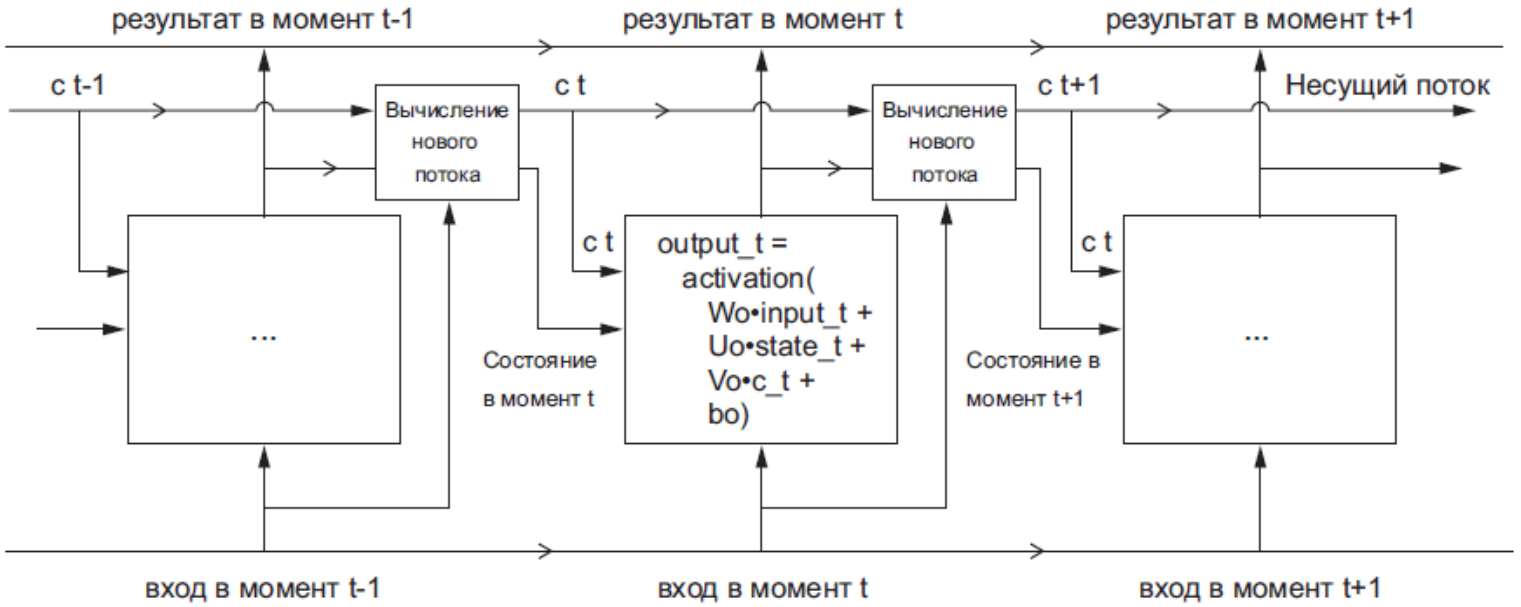


Переход от SimpleRNN к LSTM: добавление несущего потока

```
y = activation(dot(state_t, U) + dot(input_t, W) + b)
```

16.2. Рекуррентные нейронные сети

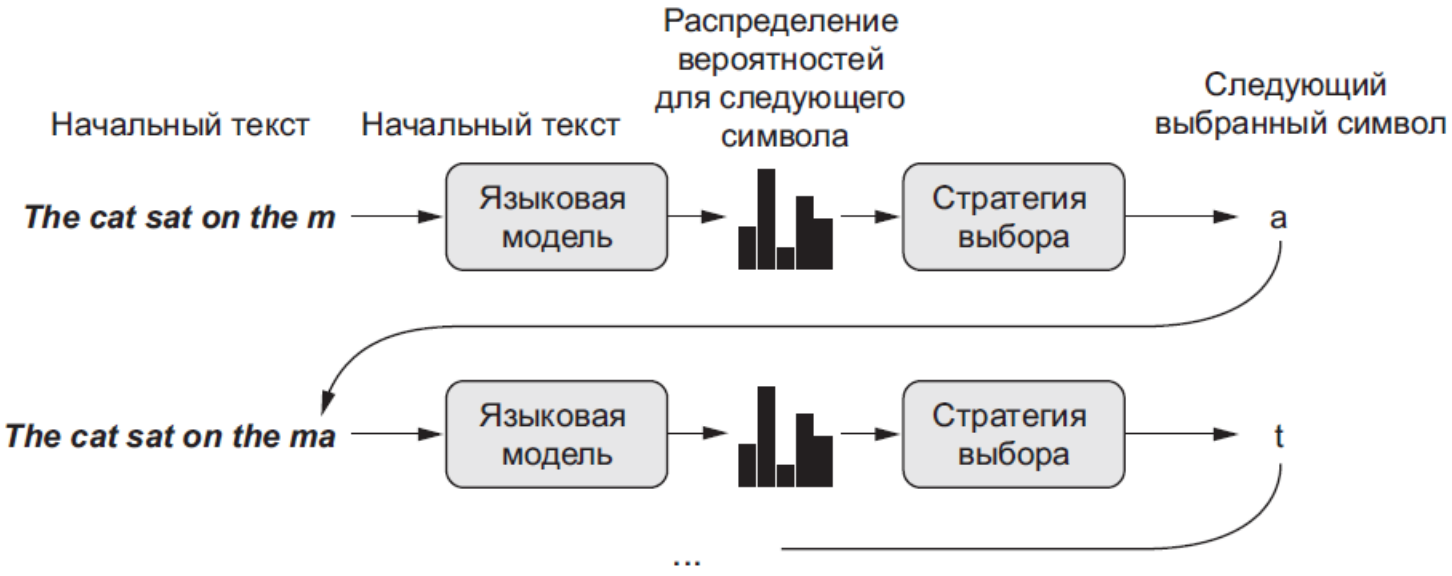
```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(C_t, Vo) + bo)

i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)

c_t+1 = i_t * k_t + c_t * f_t
```



16.3. Генерация последовательности данных

Универсальный способ генерации последовательностей данных с применением методов глубокого обучения заключается в обучении сети (обычно рекуррентной или сверточной сети) для прогнозирования следующего токена или следующих нескольких токенов в последовательности, опираясь на предыдущие токены.

Сеть, моделирующая вероятность появления следующего токена на основе предыдущих, называется языковой моделью. Языковая модель фиксирует скрытое пространство языка: его статистическую структуру.



16.3. Генерация последовательности данных

СТРАТЕГИИ ВЫБОРА

- Жадный выбор
- Стохастический выбор

Температура softmax – параметр, характеризующий энтропию распределения вероятностей, используемую для выбора: определяет степень необычности или предсказуемости выбора следующего символа. С учетом значения temperature и на основе оригинального распределения вероятностей (результата функции softmax модели) будет вычисляться новое распределение путем взвешивания вероятностей.

original_distribution – это вектор значений вероятностей, сумма которых должна быть равна 1. temperature – это фактор, количественно определяющий энтропию выходного распределения

```
reweight_distribution <- function(original_distribution,
                                  temperature = 0.5) {
  distribution <- log(original_distribution) / temperature
  distribution <- exp(distribution)
  distribution / sum(distribution)
}
```

Возвращает новую, взвешенную версию оригинального распределения. Сумма вероятностей в новом распределении может получиться больше 1, поэтому разделим элементы вектора на сумму, чтобы получить новое распределение

16.4. Реализация посимвольной генерации текста на основе LSTM

```
library(keras)
library(stringr)

path <- get_file(
  "nietzsche.txt",
  origin = "https://s3.amazonaws.com/text-datasets/nietzsche.txt"
)
text <- tolower(readChar(path, file.info(path)$size))
cat("Corpus length:", nchar(text), "\n")

maxlen <- 60

step <- 3

text_indexes <- seq(1, nchar(text) - maxlen, by = step)
sentences <- str_sub(text, text_indexes, text_indexes + maxlen - 1)
next_chars <- str_sub(text, text_indexes + maxlen,
text_indexes + maxlen)

cat("Number of sequences: ", length(sentences), "\n")
```

Извлекаются последовательности по 60 символов

Новые последовательности выбираются через каждые 3 символа

Для хранения извлеченных последовательностей

Для хранения целей (символов, следующих за последовательностями)

16.4. Реализация посимвольной генерации текста на основе LSTM

```
chars <- unique(sort(strsplit(text, "")[[1]]))
cat("Unique characters:", length(chars), "\n")
char_indices <- 1:length(chars)
names(char_indices) <- chars

cat("Vectorization...\n")
x <- array(0L, dim = c(length(sentences), maxlen, length(chars)))
y <- array(0L, dim = c(length(sentences), length(chars)))
for (i in 1:length(sentences)) {
  sentence <- strsplit(sentences[[i]], "")[[1]]
  for (t in 1:length(sentence)) {
    char <- sentence[[t]]
    x[i, t, char_indices[[char]]] <- 1
  }
  next_char <- next_chars[[i]]
  y[i, char_indices[[next_char]]] <- 1
}
```

Список уникальных символов в корпусе

Именованный список, отображающий уникальные символы в их индексы

Прямое кодирование символов в бинарные массивы

16.4. Реализация посимвольной генерации текста на основе LSTM

КОНСТРУИРОВАНИЕ СЕТИ

```
model <- keras_model_sequential() %>%
  layer_lstm(units = 128, input_shape = c(maxlen, length(chars))) %>%
  layer_dense(units = length(chars), activation = "softmax")

optimizer <- optimizer_rmsprop(lr = 0.01)

model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer
)
```

ОБУЧЕНИЕ МОДЕЛИ И ИЗВЛЕЧЕНИЕ ОБРАЗЦОВ ИЗ НЕЕ

1. Извлечь из модели распределение вероятностей следующего символа для имеющегося на данный момент сгенерированного текста.

2. Выполнить взвешивание распределения с заданной температурой.

3. Выбрать следующий символ в соответствии с вновь взвешенным распределением вероятностей.

4. Добавить новый символ в конец текста.

16.4. Реализация посимвольной генерации текста на основе LSTM

Функция выборки следующего символа с учетом прогнозов модели

```
sample_next_char <- function(preds, temperature = 1.0) {
  preds <- as.numeric(preds)
  preds <- log(preds) / temperature
  exp_preds <- exp(preds)
  preds <- exp_preds / sum(exp_preds)
  which.max(t(rmultinom(1, 1, preds)))
}
```

Цикл генерации текста

```
for (epoch in 1:60) {

  cat("epoch", epoch, "\n")

  model %>% fit(x, y, batch_size = 128, epochs = 1)

  start_index <- sample(1:(nchar(text) - maxlen - 1), 1)
  seed_text <- str_sub(text, start_index, start_index + maxlen - 1)

  cat(«--- Generating with seed:», seed_text, "\n\n")
```

Обучение модели в течение 60 эпох

Выполнить одну итерацию обучения

Выбрать случайный начальный текст

16.4. Реализация посимвольной генерации текста на основе LSTM

```
for (temperature in c(0.2, 0.5, 1.0, 1.2)) {

  cat("------ temperature:", temperature, "\n")
  cat(seed_text, "\n")

  generated_text <- seed_text

  for (i in 1:400) {

    sampled <- array(0, dim = c(1, maxlen, length(chars)))
    generated_chars <- strsplit(generated_text, "")[[1]]
    for (t in 1:length(generated_chars)) {
      char <- generated_chars[[t]]
      sampled[1, t, char_indices[[char]]] <- 1
    }
    preds <- model %>% predict(sampled, verbose = 0)
    next_index <- sample_next_char(preds[1,], temperature)
    next_char <- chars[[next_index]]

    generated_text <- paste0(generated_text, next_char)
    generated_text <- substring(generated_text, 2)

    cat(next_char)
  }
  cat("\n\n")
 }
}
```

Сгенерировать текст для разных температур

Сгенерировать 400 символов, начиная с начального текста

Прямое кодирование символов, сгенерированных до сих пор

Выбор следующего символа