Липецкий государственный технический университет

Кафедра прикладной математики

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ МАТЕМАТИЧЕСКИХ ИССЛЕДОВАНИЙ

Лекция 2.8

Deep Learning в R

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

Липецк - 2019

Outline

8.1. Генерация последовательности данных

8.2. Реализация посимвольной генерации текста на основе LSTM

8.3. DeepDream

8.4. Нейронная передача стиля

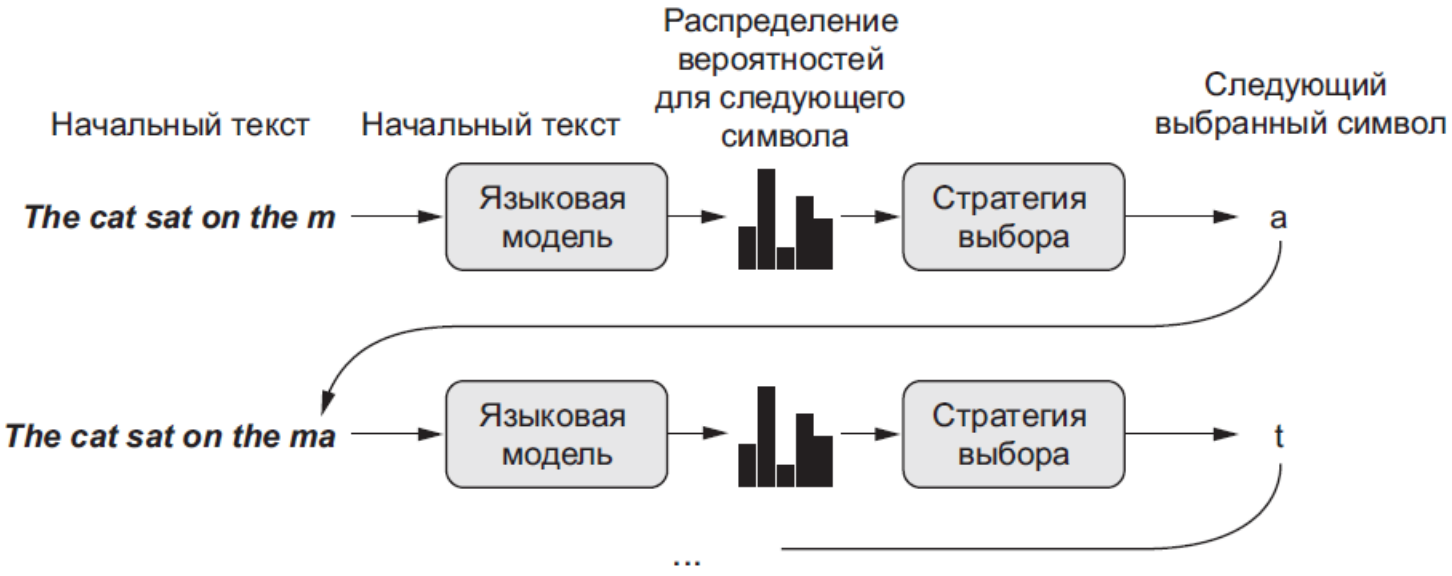
8.5. Выбор шаблонов из скрытых пространств изображений

8.6. Генеративно-состязательные сети

8.1. 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.



8.1. 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

8.2. 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)

8.2. 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

8.2. 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.

8.2. 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

8.2. 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

8.3. DeepDream

DeepDream — это метод художественной обработки изображений, основанный на использовании представлений, полученных сверточными нейронными сетями.

Загрузка предварительно обученной модели Inception V3

```
library(keras)

k_set_learning_phase(0)

model <- application_inception_v3(
  weights = "imagenet",
  include_top = FALSE,
)
```

Мы не будем обучать модель, поэтому выполним данную команду, чтобы запретить все операции, имеющие отношение к обучению

Конструирование сети Inception V3 без сверточной основы. Модель будет загружаться с весами, полученными в результате предварительного обучения на наборе ImageNet

Определение потерь для максимизации

Создается список, отображающий имена уровней в их экземпляры

```
layer_dict <- model$layers
names(layer_dict) <- lapply(layer_dict,
function(layer) layer$name)

loss <- k_variable(0)
for (layer_name in names(layer_contributions)) {
  coeff <- layer_contributions[[layer_name]]
  activation <- layer_dict[[layer_name]]$output
  scaling <- k_prod(k_cast(k_shape(activation), "float32"))
  loss <- loss + (coeff * k_sum(k_square(activation)) / scaling)
}
```

Величина потерь определяется добавлением вклада уровня в эту скалярную переменную

Получение результата уровня

Добавление L2-нормы признаков уровня к потерям

8.3. DeepDream

Процесс градиентного восхождения

```
dream <- model$input
grads <- k_gradients(loss, dream)[[1]]

grads <- grads / k_maximum(k_mean(k_abs(grads)), 1e-7)
outputs <- list(loss, grads)
fetch_loss_and_grads <- k_function(list(dream), outputs)
eval_loss_and_grads <- function(x) {
  outs <- fetch_loss_and_grads(list(x))
  loss_value <- outs[[1]]
  grad_values <- outs[[2]]
  list(loss_value, grad_values)
}
gradient_ascent <- function(x, iterations, step, max_loss = NULL) {
  for (i in 1:iterations) {
    c(loss_value, grad_values) %<-% eval_loss_and_grads(x)
    if (!is.null(max_loss) && loss_value > max_loss)
      break
    cat("...Loss value at", i, ":", loss_value, "\n")
    x <- x + (step * grad_values)
  }
  x
}
```

Этот тензор хранит сгенерированное изображение

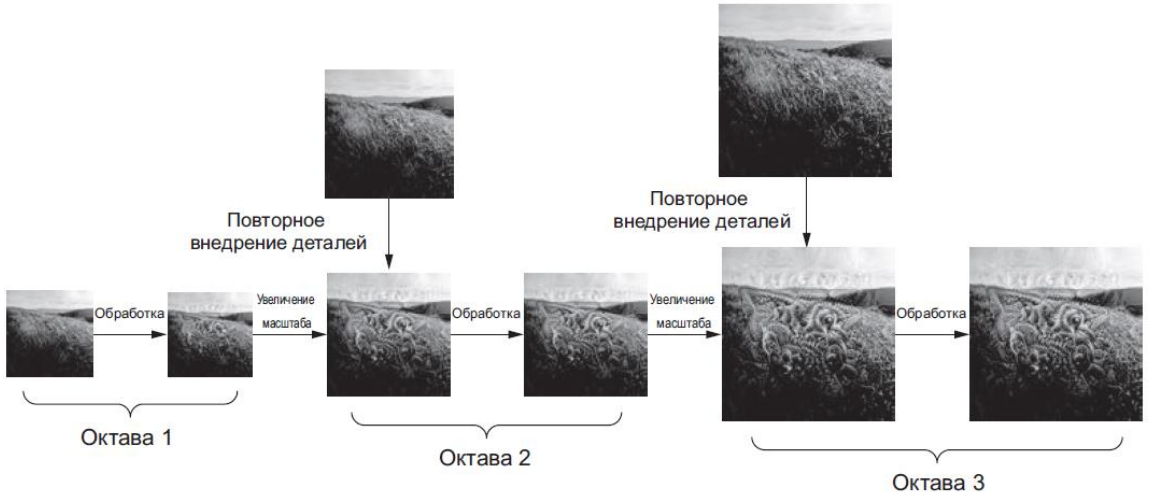
Вычисляет градиенты изображения с учетом потерь

Нормализация градиентов (важный шаг)

Настройка функции Keras для извлечения значения потерь и градиентов для заданного исходного изображения

Эта функция выполняет заданное число итераций градиентного восхождения

8.3. DeepDream



Выполнение градиентного восхождения через последовательность масштабов

```
step <- 0.01
num_octave <- 3
octave_scale <- 1.4
iterations <- 20

max_loss <- 10

base_image_path <- "..."

img <- preprocess_image(base_image_path)
```

Изменяя гиперпараметры, можно добиваться новых эффектов

← Размер шага градиентного восхождения

← Количество масштабов, на которых выполняется градиентное восхождение

← Отношение между соседними масштабами

← Число шагов восхождения для каждого масштаба

← Если величина потерь вырастет больше 10, мы должны прервать процесс градиентного восхождения, чтобы избежать появления безобразных артефактов

← Укажите здесь путь к файлу изображения

← Загрузка базового изображения в массив

8.3. DeepDream

```
original_shape <- dim(img)[-1]
successive_shapes <- list(original_shape)
for (i in 1:num_octave) {
  shape <- as.integer(original_shape / (octave_scale ^ i))
  successive_shapes[[length(successive_shapes) + 1]] <- shape
}

successive_shapes <- rev(successive_shapes)

original_img <- img
shrunk_original_img <- resize_img(img, successive_shapes[[1]])
for (shape in successive_shapes) {
  cat("Processsing image shape", shape, "\n")
  img <- resize_img(img, shape)
  img <- gradient_ascent(img,
                         iterations = iterations,
                         step = step,
                         max_loss = max_loss)

  upscaled_shrunk_original_img <- resize_img(shrunk_original_img, shape)
  same_size_original <- resize_img(original_img, shape)
  lost_detail <- same_size_original - upscaled_shrunk_original_img

  img <- img + lost_detail
  shrunk_original_img <- resize_img(original_img, shape)
  save_img(img, fname = sprintf("dream_at_scale_%s.png",
                                paste(shape, collapse = "x")))
}
```

Подготавливает список кортежей shape, определяющих разные масштабы для выполнения градиентного восхождения

Переворачивает список кортежей shape, чтобы они следовали в порядке возрастания масштаба

Уменьшает изображение в массиве до наименьшего масштаба

Увеличение масштаба изображения

Выполняет градиентное восхождение, изменяя изображение

Повторное внедрение деталей

Разница между двумя изображениями – это детали, утерянные в результате масштабирования

Вычисляет высококачественную версию исходного изображения с заданными размерами

8.3. DeepDream

```
resize_img <- function(img, size) {
  image_array_resize(img, size[[1]], size[[2]])
}

save_img <- function(img, fname) {
  img <- deprocess_image(img)
  image_array_save(img, fname)
}

preprocess_image <- function(image_path) {
  image_load(image_path) %>%
    image_to_array() %>%
    array_reshape(dim = c(1, dim(.))) %>%
    inception_v3_preprocess_input()
}

deprocess_image <- function(img) {
  img <- array_reshape(img, dim = c(dim(img)[[2]], dim(img)[[3]], 3))
  img <- img / 2
  img <- img + 0.5
  img <- img * 255

  dims <- dim(img)
  img <- pmax(0, pmin(img, 255))
  dim(img) <- dims
  img
}
```

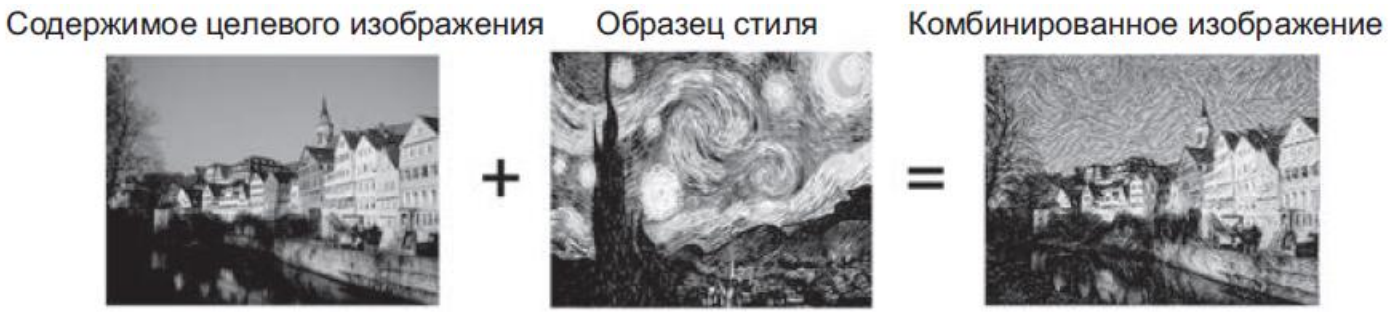
Вспомогательная функция, открывающая, изменяющая размер и преобразующая изображение в тензор для обработки в Inception V3

Вспомогательная функция, преобразующая тензор в допустимое изображение

Отменяет предварительную обработку, выполненную вызовом imagenet_preprocess_input

8.3. DeepDream

8.4. Нейронная передача стиля



Под стилем в основном подразумеваются *текстуры, цветовая палитра и визуальные шаблоны* в различных пространственных масштабах, а под содержимым — высокоуровневая макроструктура изображения.

В основе реализации передачи стиля лежит та же идея, что занимает центральное положение во всех алгоритмах глубокого обучения: задается функция потерь, чтобы определить цель для достижения, и она минимизируется. Определив математически содержимое и стиль, соответствующую функцию потерь для минимизации можно обозначить так:

```
loss <- distance(style(reference_image) - style(generated_image)) +
        distance(content(original_image) - content(generated_image))
```

8.4. Нейронная передача стиля

Предварительно обученную сверточную сеть можно использовать для определения потерь, которая будет:

- Сохранять содержимое, поддерживая сходство активаций верхнего слоя между содержимым целевого и сгенерированного изображений. Сверточная сеть должна «видеть» оба изображения — целевое и сгенерированное — как содержащие одно и то же.
- Сохранять стиль, поддерживая сходство корреляций в активациях всех, нижних и верхних, слоев. Корреляции признаков захватывают текстуры: изображение-образец и сгенерированное изображение должны обладать одинаковыми текстурами в разных пространственных масштабах.

Определение начальных переменных

```
library(keras)

target_image_path <- "img/portrait.png"

style_reference_image_path <- «img/transfer_style_reference.png»

img <- image_load(target_image_path)
width <- img$size[[1]]
height <- img$size[[2]]
img_nrows <- 400
img_ncols <- as.integer(width * img_nrows / height)
```

Путь к изображению, которое будет трансформироваться

Путь к изображению с образцом стиля

Вычисление размеров генерируемого изображения

8.4. Нейронная передача стиля

Вспомогательные функции

```
preprocess_image <- function(path) {
  img <- image_load(path, target_size = c(img_nrows, img_ncols)) %>%
    image_to_array() %>%
    array_reshape(c(1, dim(.)))
  imagenet_preprocess_input(img)
}

deprocess_image <- function(x) {
  x <- x[1,,,]
  x[,,1] <- x[,,1] + 103.939
  x[,,2] <- x[,,2] + 116.779
  x[,,3] <- x[,,3] + 123.68
  x <- x[,,c(3,2,1)]
  x[x > 255] <- 255
  x[x < 0] <- 0
  x[] <- as.integer(x)/255
  x
}
```

Нулевое центрирование путем удаления среднего значения пиксела из ImageNet. Это отменяет преобразование, выполненное vgg19.preprocess_input

Конвертирует изображения из BGR в RGB. Также является частью обратного порядка vgg19.preprocess_input

8.4. Нейронная передача стиля

Загрузка предварительно обученной сети VGG19 и применение ее к трем изображениям

```
target_image <- k_constant(preprocess_image(target_image_path))
style_reference_image <- k_constant(
  preprocess_image(style_reference_image_path)
)

combination_image <- k_placeholder(c(1, img_nrows, img_ncols, 3))

input_tensor <- k_concatenate(list(target_image, style_reference_image,
                                   combination_image), axis = 1)
model <- application_vgg19(input_tensor = input_tensor,
                           weights = "imagenet",
                           include_top = FALSE)

cat("Model loaded\n")
```

Объединение трех изображений в один пакет

Заготовка, куда будет помещено сгенерированное изображение

Конструирование сети VGG19 с пакетом из трех изображений на входе. В модель будут загружены веса, полученные в результате обучения на наборе ImageNet

8.4. Нейронная передача стиля

Функция потерь содержимого

```
content_loss <- function(base, combination) {
  k_sum(k_square(combination - base))
}
```

Функция потерь стиля

```
gram_matrix <- function(x) {
  features <- k_batch_flatten(k_permute_dimensions(x, c(3, 1, 2)))
  gram <- k_dot(features, k_transpose(features))
  gram
}

style_loss <- function(style, combination){
  S <- gram_matrix(style)
  C <- gram_matrix(combination)
  channels <- 3
  size <- img_nrows*img_ncols
  k_sum(k_square(S - C)) / (4 * channels^2 * size^2)
}
```

Функция общей потери вариации

```
total_variation_loss <- function(x) {
  y_ij  <- x[,1:(img_nrows - 1L), 1:(img_ncols - 1L),]
  y_i1j <- x[,2:(img_nrows), 1:(img_ncols - 1L),]
  y_ij1 <- x[,1:(img_nrows - 1L), 2:(img_ncols),]
  a <- k_square(y_ij - y_i1j)
  b <- k_square(y_ij - y_ij1)
  k_sum(k_pow(a + b, 1.25))
}
```

8.4. Нейронная передача стиля

Функция общей потери вариации

Именованный список, отображающий имена слоев в тензоры активаций

```
outputs_dict <- lapply(model$layers, `[[`, "output")
names(outputs_dict) <- lapply(model$layers, `[[`, "name")

content_layer <- "block5_conv2"
style_layers = c("block1_conv1", "block2_conv1",
                 "block3_conv1", "block4_conv1",
                 "block5_conv1")

total_variation_weight <- 1e-4
style_weight <- 1.0
content_weight <- 0.025

loss <- k_variable(0.0)
layer_features <- outputs_dict[[content_layer]]
target_image_features <- layer_features[1,,,]
combination_features <- layer_features[3,,,]
loss <- loss + content_weight * content_loss(target_image_features,
                                             combination_features)

for (layer_name in style_layers) {
  layer_features <- outputs_dict[[layer_name]]
  style_reference_features <- layer_features[2,,,]
  combination_features <- layer_features[3,,,]
  sl <- style_loss(style_reference_features, combination_features)
  loss <- loss + ((style_weight / length(style_layers)) * sl)
}

loss <- loss +
  (total_variation_weight * total_variation_loss(combination_image))
```

Слой, используемый для вычисления потерь содержимого

Слой, используемый для вычисления потерь стиля

Веса для вычисления среднего взвешенного по компонентам потерь

Величина потерь определяется сложением всех компонентов с этой переменной

Добавление потери содержимого

Добавление потери стиля для каждого целевого слоя

Добавление общей потери вариации

8.4. Нейронная передача стиля

Настройки процесса градиентного спуска

Получение градиентов сгенерированного изображения относительно потерь

```
grads <- k_gradients(loss, combination_image)[[1]]

fetch_loss_and_grads <-
      k_function(list(combination_image), list(loss, grads))

eval_loss_and_grads <- function(image) {
  image <- array_reshape(image, c(1, img_nrows, img_ncols, 3))
  outs <- fetch_loss_and_grads(list(image))
  list(
    loss_value = outs[[1]],
    grad_values = array_reshape(outs[[2]], dim = length(outs[[2]]))
  )
}

library(R6)
Evaluator <- R6Class("Evaluator",
  public = list(

    loss_value = NULL,
    grad_values = NULL,

    initialize = function() {
      self$loss_value <- NULL
      self$grad_values <- NULL
    },
```

Функция для получения значений текущих потерь и градиентов

Этот класс обертывает fetch_loss_and_grads и позволяет получать потери и градиенты вызовами двух отдельных методов, как того требует реализация оптимизатора

```
    loss = function(x) {
      loss_and_grad <- eval_loss_and_grads(x)
      self$loss_value <- loss_and_grad$loss_value
      self$grad_values <- loss_and_grad$grad_values
      self$loss_value
    },
    grads = function(x) {
      grad_values <- self$grad_values
      self$loss_value <- NULL
      self$grad_values <- NULL
      grad_values
    }
  )
)
evaluator <- Evaluator$new()
```

8.4. Нейронная передача стиля

Цикл передачи стиля

```
iterations <- 20
dms <- c(1, img_nrows, img_ncols, 3)
x <- preprocess_image(target_image_path)
x <- array_reshape(x, dim = length(x))
for (i in 1:iterations) {
  opt <- optim(
    array_reshape(x, dim = length(x)),
    fn = evaluator$loss,
    gr = evaluator$grads,
    method = "L-BFGS-B",
    control = list(maxit = 15)
  )
  cat("Loss:", opt$value, "\n")
  image <- x <- opt$par
  image <- array_reshape(image, dms)
  im <- deprocess_image(image)
  plot(as.raster(im))
}
```

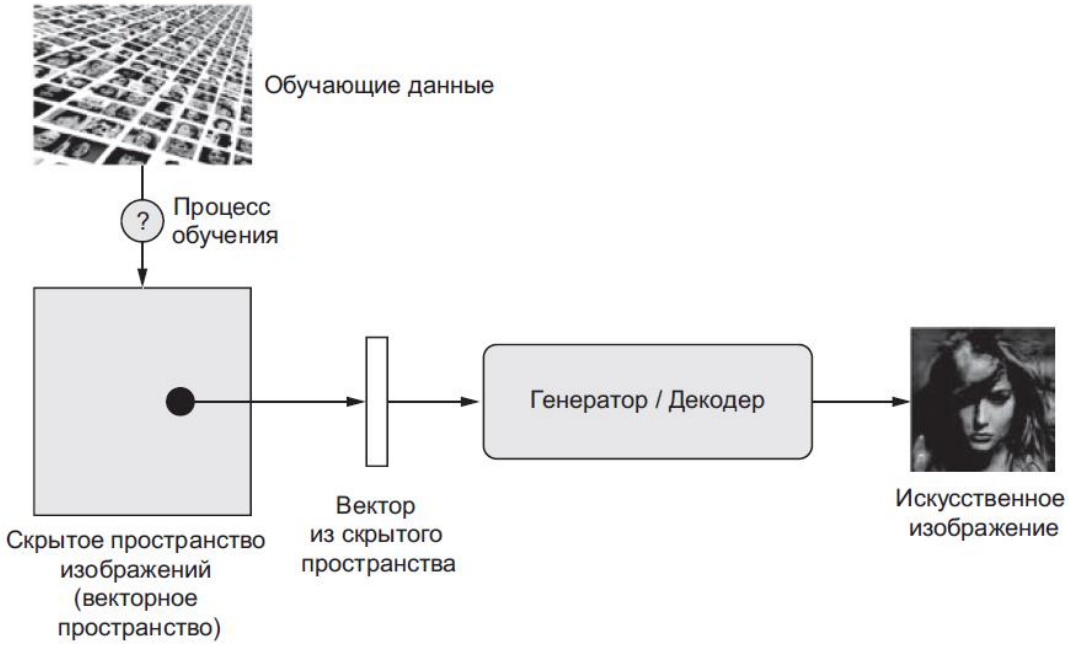
Начальное состояние: целевое изображение

Преобразование изображения в вектор, потому что функция optim способна обрабатывать только плоские векторы

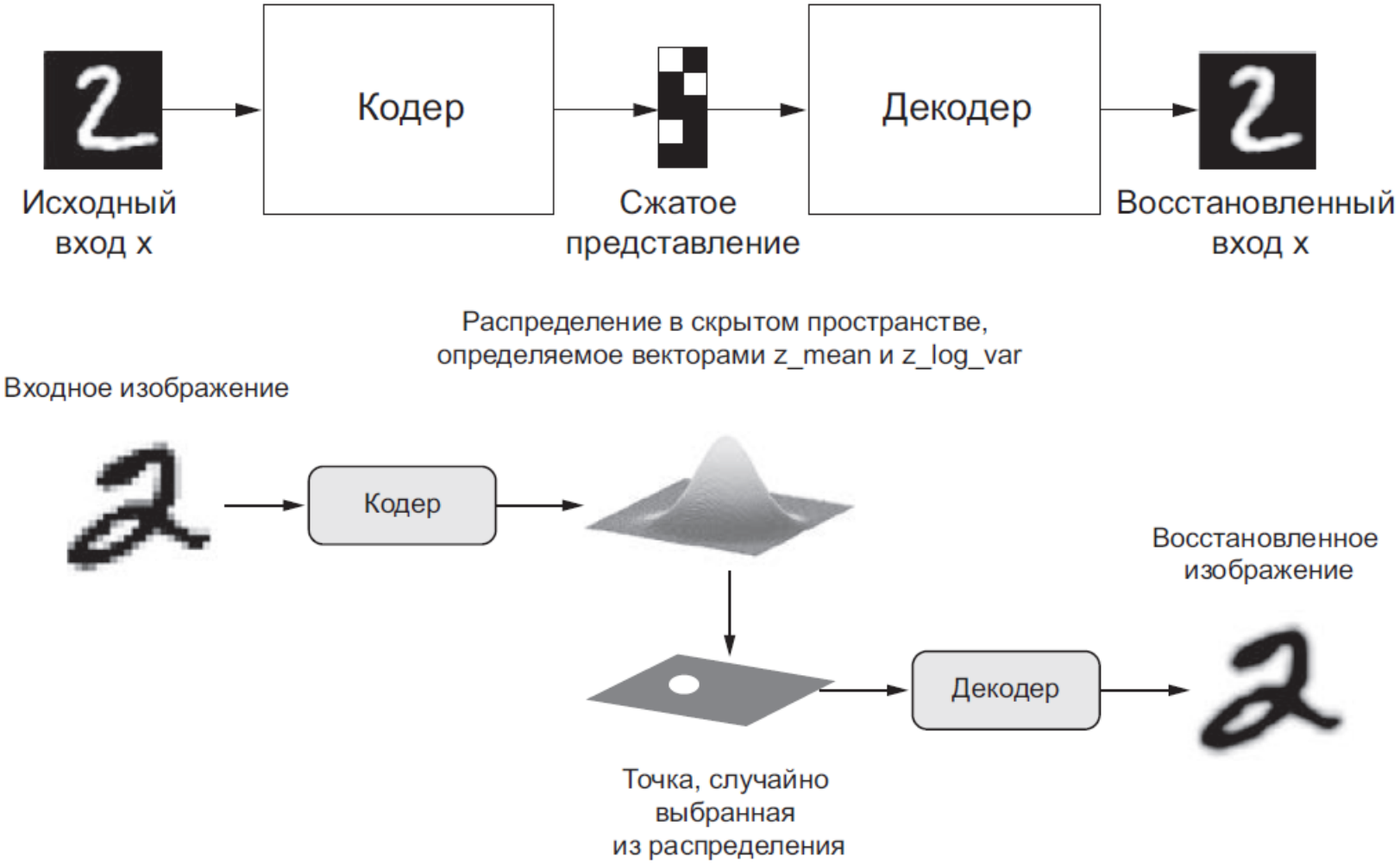
Применить алгоритм L-BFGS к пикселам сгенерированного изображения для минимизации потерь переноса стиля. Обратите внимание, что вы должны передать функцию, которая вычисляет градиенты как два отдельных аргумента

8.5. Выбор шаблонов из скрытых пространств изображений

Основная идея генерации изображений заключается в создании малоразмерного скрытого пространства представлений, любая точка которого может отображаться в реалистично выглядящее изображение. Модуль, способный реализовать это отображение, который принимает на входе скрытую точку и выводит изображение (сетку пикселов), называют генератором (в случае использования GAN) или декодером (если используется VAE).



8.5. Выбор шаблонов из скрытых пространств изображений



8.6. Генеративно-состязательные сети

Генеративно-состязательная сеть: состоит из двух сетей — выполняющей подделку и оценивающей эту подделку, — постепенно обучающих друг друга:

- сеть-генератор — получает на входе случайный вектор (случайную точку в скрытом пространстве) и декодирует его в искусственное изображение;
- сеть-дискриминатор (или противник) — получает изображение (настоящее или поддельное) и определяет, взято это изображение из обучающего набора или сгенерировано сетью-генератором.