

**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ  
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ  
им. проф. М. А. БОНЧ-БРУЕВИЧА»  
(СПбГУТ)**

---

**Ф.В. Филиппов**

# **НЕЙРОСЕТЕВЫЕ ТЕХНОЛОГИИ**

**ЛАБОРАТОРНЫЙ  
ПРАКТИКУМ**

**СПб ГУТ)))**

**САНКТ-ПЕТЕРБУРГ  
2021**

УДК 004.31(004.42)  
ББК 32.973-018.2я73

Рецензент  
кандидат технических наук, доцент кафедры конструирования  
и производства радиоэлектронных средств  
*T.B. Матюхина*

*Утверждено редакционно-издательским советом СПбГУТ  
в качестве лабораторного практикума*

**Филиппов, Ф.В.**

Нейросетевые технологии: лабораторный практикум / Ф. В. Филиппов;  
СПбГУТ, – СПб., 2021. – 48 с.

Приведены методические материалы для выполнения лабораторных работ. Сформулированы задания к лабораторным работам и практическим занятиям. Изложение материала ведется с примерами, реализованными в среде *RStudio* с использованием широкого круга библиотек.

Пособие предназначено для бакалавров направления 09.03.02 Информационные системы и технологии и будет полезно при изучении дисциплин «Нейросетевые технологии» и «Технологии обработки информации».

© Филиппов Ф.В., 2021

©Федеральное государственное образовательное бюджетное учреждение высшего образования  
«Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича»,  
2021

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
ЛАБОРАТОРНЫЕ РАБОТЫ .....	5
1. ПЕРСЕПТРОНЫ .....	5
Аппроксимация функций .....	6
Лабораторная работа № 1 .....	9
Прогнозирование временных рядов .....	10
Лабораторная работа № 2 .....	14
Классификация объектов .....	14
Лабораторная работа № 3 .....	17
Построение автокодировщиков .....	18
Лабораторная работа № 4 .....	20
2. ОГРАНИЧЕННЫЕ МАШИНЫ БОЛЬЦМАНА .....	21
Лабораторная работа № 5 .....	25
3. СЕТИ КОХОНЕНА .....	26
Лабораторная работа № 6 .....	27
4. СВЕРТОЧНЫЕ СЕТИ .....	28
Лабораторная работа № 7 .....	31
5. ВАРИАЦИОННЫЕ АВТОКОДИРОВЩИКИ .....	32
Лабораторная работа № 8 .....	35
6. РЕКУРРЕНТНЫЕ СЕТИ НА БАЗЕ LSTM .....	36
Лабораторная работа № 9 .....	39
ЗАДАНИЯ ДЛЯ ПРАКТИКИ .....	40
Задание № 1: Моделирование логической функции .....	40
Задание № 2: Классификация входных данных .....	40
Задание № 3: Построение карты Кохонена .....	41
Задание № 4: Распознавание и классификация объектов .....	42
Задание № 5: Сжатие данных .....	43
Задание № 6: Составление прогноза .....	43
Задание № 7: Классификация объектов .....	44
Задание № 8: Распознавание изображений .....	45
Задание № 9: Классификация изображений .....	46
Задание № 10: Прогнозирование аварий сетевого оборудования .....	47
ЗАКЛЮЧЕНИЕ .....	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	48

## ВВЕДЕНИЕ

Для решения широкого круга задач, когда не удается найти алгоритмического решения используются нейронные сети. Способность нейронных сетей обучаться позволяет переложить сложности поиска решения на их плечи. Успех качественного обучения нейронных сетей зависит от многих факторов, но главными из них являются два – формирование хорошей обучающей выборки и выбор модели сети адекватной для конкретной задачи. Если подбор и подготовка данных для обучения является общей процедурой, зависящей от целей обучения, то выбор надлежащей архитектуры сети, во многом зависит от характера решаемой задачи.

За период от начала практического использования нейронных сетей до настоящего времени сложились обоснованные предпочтения выбора конкретной архитектуры для задач определенного класса. Тем не менее процесс создания и апробирования новых моделей продолжается. Создано большое число библиотек, позволяющих легко экспериментировать и находить наилучшие решения.

Лабораторный практикум предназначен для получения навыков работы

с моделями нейронных сетей при решении широкого класса задач. Его содержимое будет предельно понятно, если предварительно ознакомиться с одноименным учебным пособием [1]. Практикум содержит лабораторные работы и задания для самостоятельной практики.

Изучение относительно простых архитектур нейронных сетей основано на использовании классических моделей персепtronов, ограниченных машин Больцмана и сетей Кохонена. Все эти модели реализованы в виде функций, которые позволяют варьировать большое число параметров, необходимых для их адаптации к решаемой задачи и получения адекватных результатов.

Для построения более сложных архитектур с различным числом и назначением слоев, а также с введением специфических модулей, типа *LSTM*, здесь используется библиотека *keras*.

При выполнении лабораторных работ рекомендуется детально изучить листинги, предваряющие задание, и выполнить их в среде *RStudio*. Задания для самостоятельной практики взяты из веб-ресурсов, с которыми возможно следует познакомиться. Выполнение этих заданий носит характер небольшого исследования.

Поскольку предлагаемые здесь примеры и задания используют язык программирования *R*, полезно также получить некоторые навыки его использования, например, изучив базовые структуры и функции из [2]. Для тех, кто знаком с языком *Python*, предлагается выполнять представленные здесь задания в среде *Colab* [3].

# ЛАБОРАТОРНЫЕ РАБОТЫ

## 1. ПЕРСЕПТРОНЫ

Главной особенностью моделей нейронных сетей, объединенных под названием персептрон, являются полно связные слои. В них нейрон предыдущего слоя связан со всеми нейронами следующего слоя. В среде *RStudio* используется три модели персептрана, реализованные в виде функций *nnet()*, *neuralnet()* и *mlp()* [4]. Каждая из этих функций имеет много параметров, большинству из которых присвоены значения по умолчанию:

```
nnet(x, y, weights, size, Wts, mask, linout = FALSE, entropy = FALSE, softmax = FALSE,
      censored = FALSE, skip = FALSE, rang = 0.7, decay = 0, maxit = 100, Hess = FALSE,
      trace = TRUE, MaxNWts = 1000, abstol = 1.0e-4, reltol = 1.0e-8)

neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax = 1e+05, rep = 1,
          startweights = NULL, learningrate.limit = NULL, learningrate=NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2), lifesign = «none»,
          lifesign.step = 1000, algorithm = «rprop+», err.fct = «sse», act.fct = «logistic»,
          linear.output = TRUE, exclude = NULL, constant.weights = NULL,
          likelihood = FALSE)

mlp(x, y, size = c(5), maxit = 100, initFunc = «Randomize_Weights»,
     initFuncParams = c(-0.3, 0.3), learnFunc = «Std_Backpropagation»,
     learnFuncParams = c(0.2, 0), updateFunc = «Topological_Order»,
     updateFuncParams = c(0), hiddenActFunc = «Act_Logistic», shufflePatterns = T,
     linOut = FALSE, inputsTest = NULL, targetsTest = NULL, pruneFunc = NULL,
     pruneFuncParams = NULL)
```

Обилие параметров предоставляет широкие возможности по настройке или, как часто говорят, по подгонке выбранной модели для оптимального решения задачи.

Известно, что персептроны удобно применять в решении практических задач, связанных с классификацией и аппроксимацией (или регрессией):

- сеть с одним скрытым слоем, содержащим  $N$  нейронов со ступенчатой функцией активации, способна осуществить произвольную классификацию  $Nd$  точек  $d$ -мерного пространства (то есть классифицировать  $Nd$  примеров);
- одного скрытого слоя нейронов с сигмоидной функцией активации достаточно для аппроксимации любой функции со сколь угодно высокой точностью.

Рассмотрим примеры использования различных моделей персептронов для решения различных задач.

## Аппроксимация функций

Рассмотрим простой пример аппроксимации функции  $y = 2\cos x + 8$  на отрезке  $[0, 8]$ . В листинге 1 приведен код для решения этой задачи с использованием модели *neuralnet()*.

Сначала формируется фрейм данных *learn* для обучения сети (строка 6). Затем загружается библиотека *neuralnet* и формируется сеть *netcos*.

В функции *neuralnet()* мы определяем минимальное число параметров: задаем формулу  $y \sim x$ , определяем число нейронов в скрытом слое *hidden = 4* и указываем имя фрейма *learn* с набором данных для обучения. Остальные параметры заданы по умолчанию (строка 10). Запуск функции на исполнение осуществляет обучение сети *netcos*.

**Листинг 1.** Аппроксимация функции  $y = 2\cos x + 8$

```
1 # Формирование данных learn для обучения
2 set.seed(9)
3 x <- runif(50,0,8)
4 y <- 2*cos(x) + 8
5 plot(x, y, pch = 16, cex = 1.5, col = 'green')
6 learn=data.frame(x,y)
7
8 # Загрузка библиотеки и обучение сети netcos
9 library(neuralnet)
10 netcos <- neuralnet(y~x, hidden = 4,learn)
11
12 # Формирование тестовых данных и проверка сети netcos
13 set.seed(6)
14 x <- runif(5000,0,8)
15 y <- 0
16 test=data.frame(x,y)
17 yp <- predict(netcos,test)
18 points(x, yp, col = 'black', pch = '.', cex = 1.5)
19
20 # Архитектура сети netcos
21 plot(netcos)
22
23 # Функции на выходах нейронов скрытого слоя
24 y1=1/(1 + exp(-32.79766*x - 35.59392))
25 y2=1/(1 + exp(-1.54957*x + 7.37296))
26 y3=1/(1 + exp(1.49435*x - 11.54414))
27 y4=1/(1 + exp(1.72479*x - 2.87867))
28
29 # Графики функций на выходах нейронов скрытого слоя
30 set.seed(6)
31 x <- runif(5000,0,8)
32 plot(x,y2,col="blue")
33 points(x,y1,col='red')
34 points(x,y3,col='green')
35 points(x,y4,col='yellow')
36
37 # Функция и график на выходе сети netcos
38 y=y1*(-0.17364)+y2*5.81777+y3*5.43294+y4*5.07481-0.03697
39 plot(x,y)
```

Для проверки работы обученной сети создаем фрейм *test* (строка 16) с тестовыми данными. Функция *predict()* вычисляет реакцию обученной сети *netcos* в переменной *yp*. На рис. 1 приведен график моделируемой функции *y*, представленной 50 зелеными точками, и функции *yp*, сформированной сетью и представленной на графике 5000 черных точек. График демонстрирует полное совпадение функций.

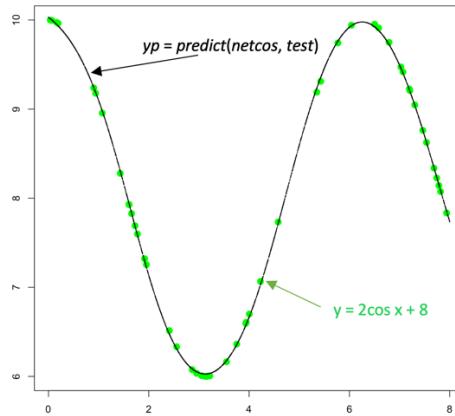


Рис. 1. Моделируемая и результирующая функция

С помощью функции *plot(netcos)* получаем архитектуру полученной сети (рис. 2). Полную информацию о построенной сети можно получить с помощью функции *print(netcos)*.

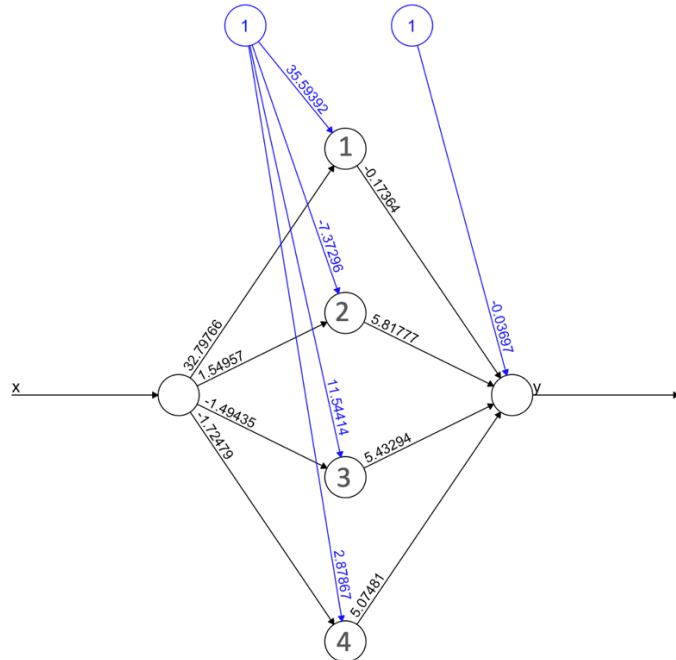


Рис. 2. Архитектура сети *netcos*

Интересно посмотреть аналитическое представление функции реализованной сетью *netcos* для исходной моделируемой функции. Для этого представим функции, реализуемые нейронами скрытого слоя (строка 24 – 27, листинг 1) и построим их графики (рис.3).

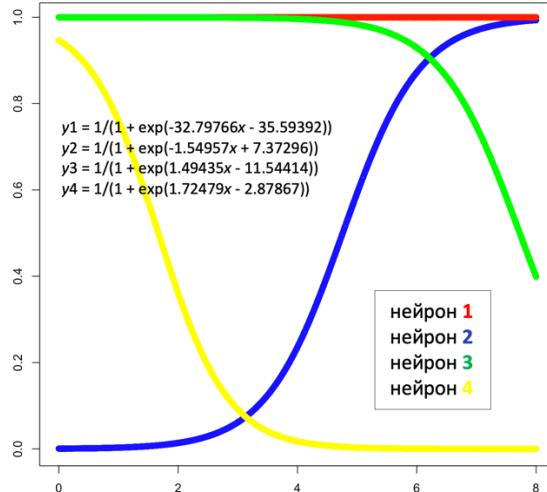


Рис. 3. Функции, реализуемые нейронами скрытого слоя

Функцию  $y$ , реализованную сетью *netcos*, получаем из этих функций следующим образом (округляем значения весов до сотых долей):

$$y = \frac{-0.17}{1 + e^{-32.80x-35.60}} + \frac{5.82}{1 + e^{-1.55x+7.37}} + \frac{5.43}{1 + e^{1.49x-11.54}} + \frac{5.07}{1 + e^{1.72x-2.88}} - 0.04$$

Постройте график  $y$  (строка 39, листинг 1) и убедитесь в том, что на отрезке  $[0, 8]$  эта функция практически тождественна исходной функции  $y = 2\cos x + 8$ .

Обратите внимание на использование в листинге функций *set.seed()*, которые задают начальную фазу генератора случайных чисел. Их необходимо использовать для повторной воспроизводимости полученных результатов.

Из многочисленного числа параметров для решения задач приведенного ниже задания достаточно будет правильно сформировать обучающие и проверочные данные, подобрать структуру сети и число повторений (эпох) обучения.

Данные для обучения и тестирования должны быть сформированы в виде фреймов.

Для правильного подбора структуры необходимо подобрать число скрытых слоев и число нейронов каждого скрытого слоя. В ряде заданий будет достаточно одного скрытого слоя, как в примере листинга 1. При необходимости использования двух слоев параметр *hidden* нужно определять как двухкомпонентный вектор, например, *hidden* = *c*(6,2) определяет два скрытых слоя с 6 нейронами в первом и 2 нейронами во втором слое.

Число эпох обучения задается параметром *rep*, который по умолчанию имеет значение 1. Обычно, когда сеть не может достигнуть заданной точности за одну эпоху, выдается предупреждение «*Algorithm did not converge in 1 of 1 repetition(s) within the stepmax*».

## Лабораторная работа № 1

Работа состоит из двух заданий, необходимо выполнить оба задания.

**Задание 1.** Смоделировать заданную функцию на отрезке  $[0, 8]$  с использованием библиотеки *neuralnet*. Отчет должен содержать подготовку данных для обучения сети. Обоснование выбранной архитектуры. Структурную схему обученной сети с указанием весов. Графики моделируемой и смоделированной функций. Варианты заданий приведены в табл.1 (номер варианта определяется последней цифрой номера зачетки).

Таблица 1

Варианты задания

Вариант	Моделируемая функция
0	$y = \cos x / (\sin 2x + 2)$
1	$y = \sin x(\operatorname{arctg} 0.5x) + \cos x$
2	$y = 0.5 \sin x + \cos 2x + \sqrt{x}$
3	$y = \cos x + x - \sin 2x$
4	$y = 4 \operatorname{arctg} x + \sin x$
5	$y = \sin x / (\cos 2x + 2)$
6	$y = \sin x + x - \cos 2x$
7	$y = 0.4 \cos x + \sin 3x + \sqrt{x}$
8	$y = 0.5 \cos x + 1.5 \operatorname{arctg} 0.2x + \sqrt{x}$
9	$y = \sin 0.3x^2 + \sqrt{x}$

**Задание 2.** Подобрать структуру *hidden* и порог функции ошибки *threshold* для точной аппроксимации заданных последовательностей импульсов. Отчет по работе должен содержать структурную схему обученной сети без указания весов. Графики аппроксимируемой и полученной последовательности для нескольких приближений параметров *hidden* и *threshold*. Варианты заданий приведены в табл.2.

Таблица 2

Варианты задания

Вариант	Последовательность импульсов
Четная цифра	$x = \text{seq}(\text{from} = -9.5, \text{to} = 10, \text{by} = 0.5)$ $y = \text{rep}(c(1,1,1,1,1,0,0,0,0,0), 4)$
Нечетная цифра	$x = \text{seq}(\text{from} = 0, \text{to} = 3.15, \text{by} = 0.05)$ $y = \text{rep}(c(1,1,1,1,0,0,0,0), 8)$

Графики должны быть в виде наложения полученной последовательности на аппроксимируемую. Пример построения графиков приведен на рис. 4. Для получения нескольких графиков в одном окне используйте команду типа  $par(mfrow = c(3, 2))$ , которая указывает, что окно делится на 3 части по горизонтали и на две по вертикали, где будет размещено 6 графиков.

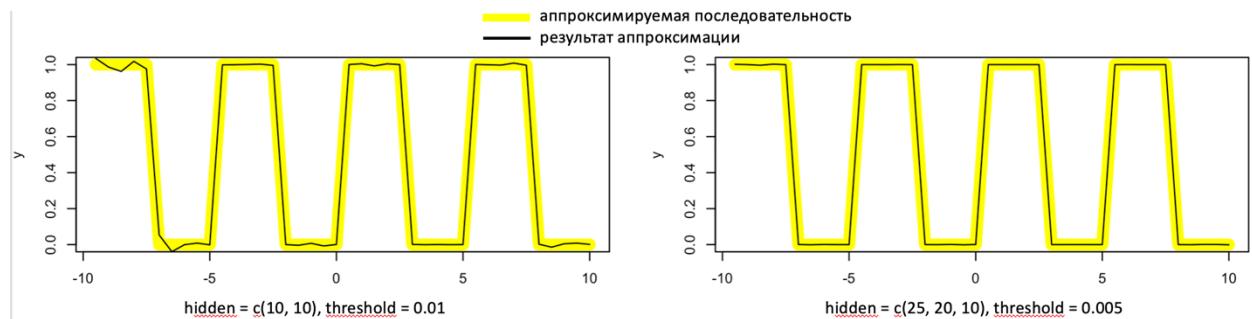


Рис. 4. Пример построения графиков

Для изображения нескольких кривых на одном графике используется две функции *plot()* и *lines()*. Функция *plot()* для первого графика, а функции *lines()* для последующих. Например, на рис. 4 для построения желтых кривых использован  $plot(x, y, type = 'l', col = "yellow", lwd = 9)$ , а для наложения черных  $lines(x, y1)$ .

### Прогнозирование временных рядов

В большинстве случаев основой для прогнозирования служат ретроспективные данные в виде временных рядов. В *Data Mining* существует специальное понятие временного ряда (*Time-Series*). Анализ временных рядов от анализа случайных выборок отличается предположением о равных промежутках времени между наблюдениями и наличием хронологического порядка. Привязка наблюдений ко времени играет здесь ключевую роль, тогда как при анализе случайной выборки она не имеет никакого значения.

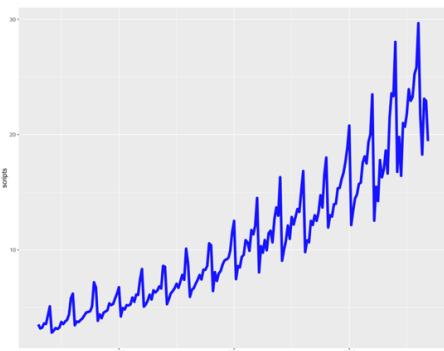


Рис. 5. Временной ряд *a10*

Возьмем в качестве примера временного ряда данные *a10* из библиотеки *fpp2* [8]. Файл *a10* включает данные о количестве выписанных ежемесячных рецептов на фармацевтические продукты, подпадающие под код А10, согласно данным Австралийской комиссии по страхованию здоровья. Данные представлены за период с июля 1991 по июнь 2008 год. Характер изменения значений этого временного ряда представлен на рис. 5.

Задача состоит в том, чтобы путем анализа временного ряда определить тенденции динамики его изменения на основе ретроспективных значений. Обучающая выборка строится на основе разбиения данных на равные периоды, для каждого из которых правильным значением на выходе устанавливается первое следующее за ним значение. На практике строится матрица наблюдений с числом столбцов, равным длине периода наблюдения плюс единица.

На рис. 6 представлено подобное разбиение для временного ряда *a10*. Периодом наблюдения является 12 месяцев, поэтому входной вектор *X* обучающей выборки имеет размерность 12, а выходной вектор *Y* является одноразмерным. На рис. 6 показано построение первых пяти обучающих наборов.

Первый набор представляет из себя 13 первых элементов ряда *a10*, где 12 элементов образуют входной вектор, а 13-й элемент выходной вектор. Второй набор образован из тринадцати элементов ряда *a10*, начиная со второго элемента и также разбит на 12+1 и так далее. Например, пятый набор сформирован из 13 элементов ряда *a10*, начиная с пятого элемента и разбит на 12+1.

	<i>X = LearnSeq[,1:12]</i>												<i>Y = LearnSeq[,13]</i>
[1,]	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]
[2,]	3.526591	3.180891	3.252221	3.611003	3.565869	4.306371	5.088335	2.814520	2.985811	3.204780	3.127578	3.270523	3.737851
[3,]	3.180891	3.252221	3.611003	3.565869	4.306371	5.088335	2.814520	2.985811	3.204780	3.127578	3.270523	3.737851	3.558776
[4,]	3.252221	3.611003	3.565869	4.306371	5.088335	2.814520	2.985811	3.204780	3.127578	3.270523	3.737851	3.558776	3.777202
[5,]	3.611003	3.565869	4.306371	5.088335	2.814520	2.985811	3.204780	3.127578	3.270523	3.737851	3.558776	3.777202	3.924490
	3.565869	4.306371	5.088335	2.814520	2.985811	3.204780	3.127578	3.270523	3.737851	3.558776	3.777202	3.924490	4.386531

Рис. 6. Заполнение матрицы наблюдений

Решение задачи прогнозирования представлено в листинге 2. Рассмотрим его особенности.

Сначала загружаются необходимые библиотеки *fpp2* и *nnet*, данные *a10*читываются в переменную *scripts* и подсчитывается их число *n*.

Далее готовится матрица наблюдений *LearnSeq* с числом столбцов 13 и строк *n* – 12. Матрица заполняется данными в соответствии с рассмотренным выше способом (строка 12, листинг 2).

В качестве модели персептрона будем использовать функцию *nnet()*. Следует отметить, что эта функция допускает наличие только одного скрытого слоя, поэтому параметр *size* может задавать число нейронов только в одном слое. Перед заданием параметров и запуском на обучение сети не

забываем зафиксировать начальную фазу генератора случайных чисел `set.seed(55)`.

Выбираем число нейронов `size = 60`, разрешаем в качестве функции активации выходного нейрона линейную функцию `linout = TRUE`. Для первоначальных случайных весов выбираем значение `rang = 0.1` и задаем максимальное число итераций `maxit = 300`. Интересно, как правильно осуществлять выбор параметров нейронной сети, значения которых, заданные по умолчанию, явно не подойдут. Прежде всего следует понимать какие параметры могут повлиять на работу сети для конкретной задачи. В данном случае решается задача прогнозирования временного ряда, поэтому параметр `decay = 0` можно оставить без внимания, так как шумы отсутствуют. Главными в этом случае являются `size`, `rang` и `maxit`. Два первых проще всего находить подбором, а число итераций чем больше, тем лучше. При обучении сети, когда по умолчанию включен параметр `trace = TRUE`, можно увидеть реальное число необходимых эпох.

#### Листинг 2. Прогнозирование временного ряда

```
1 library(fpp2)
2 library(nnet)
3
4 # Загрузка данных для анализа
5 scripts <- a10
6 n <- length(scripts) # n = 204 месяца (17 лет)
7
8 # Подготовка матрицы наблюдений заполненной нулями
9 LearnSeq <- matrix(rep(0, (n-12)*13), nrow = n-12, ncol = 13)
10
11 # Заполнение матрицы данными для обучения
12 for (i in 1:(n-12)) LearnSeq[i, ] <- scripts[i:(12 + i)]
13
14 # Обучение сети Nnet
15 set.seed(55)
16 Nnet <- nnet(LearnSeq[,1:12], LearnSeq[,13], size = 60,
17                 linout = TRUE, rang=0.1, maxit = 300)
18
19 # Прогноз составленный обученной сетью Nnet
20 prognos <- c(rep(0,12),Nnet$fitted.values)
21
22 # График фактического числа рецептов (blue) и предсказанный (yellow)
23 ggplot(scripts, aes(x)) +
24   geom_line(aes(y=scripts), color="blue", size = 2) +
25   geom_line(aes(y=prognos), color = "yellow", size = 0.5)
```

Попробуйте изменять значения параметров сети `Nnet`, заданные в листинге 2, изучая их влияние на качество прогнозирования. Будет несложно заметить, как меняется качество прогноза непосредственно на результирующем графике (рис. 7).

Здесь результирующий график построен с использованием функции `ggplot()`, которая предоставляет гораздо больше возможностей по сравнению с функцией `plot()`. Для качественной визуализации результатов любых исследований знание возможностей, которые предоставляют эти функции, и приобретение навыков их применения чрезвычайно полезно. Достаточно просто и кратко необходимые сведения об этих функциях изложены в 5 и 6 разделах веб-ресурса [5].

На графике рис. 5 прогноз, сделанный сетью представлен тонкой желтой кривой поверх синего графика временного ряда  $a10$ . Можно заметить, что его начало сдвинуто на 12 месяцев.

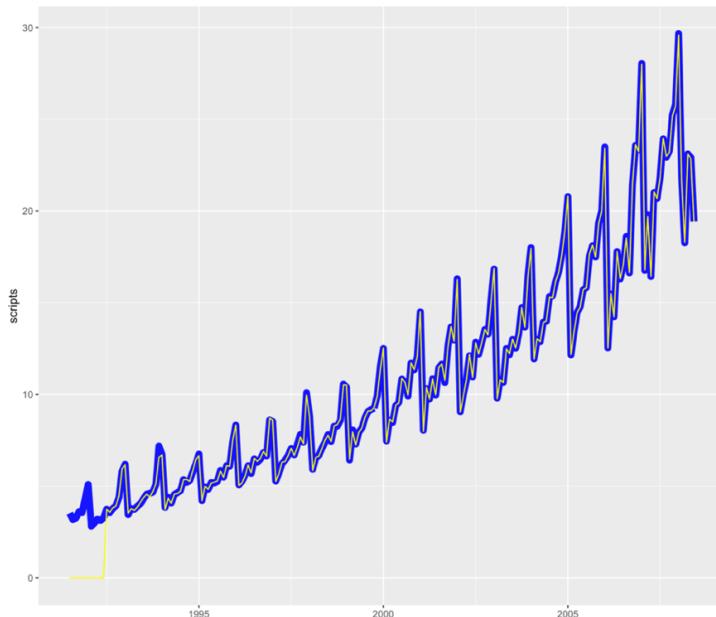


Рис. 7. График числа рецептов  $a10$  (синий) и предсказанный сетью (желтый)

Можно воспользоваться функцией `plotnet()` из библиотеки *NeuralNetTools* и отобразить архитектуру обученной сети *Nnet* (рис. 8). Синаптические связи с положительными весами выделены на рис. 8 красным цветом, а с отрицательными – синим.

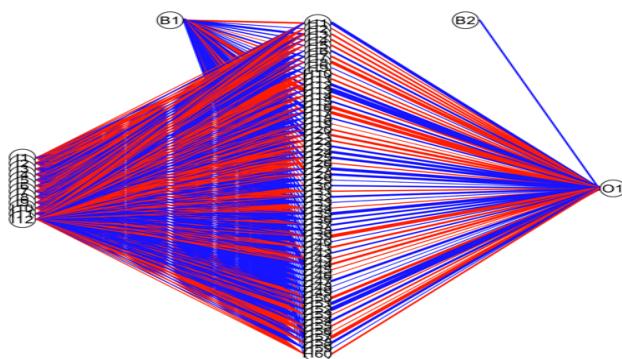


Рис. 7. Архитектура сети *Nnet*

## Лабораторная работа № 2

Составить прогноз для заданного временного ряда с использованием библиотеки *npnet*. Отчет должен содержать подготовку данных для обучения сети. Обоснование выбранной архитектуры. Структурную схему обученной сети без указания весов. Варианты заданий приведены в табл. 3, где указаны названия временных рядов, которые доступны из библиотеки *fpp2* [8].

Таблица 3

Варианты задания

Вариант	Временной ряд
0	<i>auscafe</i>
1	<i>austourists</i>
2	<i>calls</i>
3	<i>debitcards</i>
4	<i>departures (resshort)</i>
5	<i>elecequip</i>
6	<i>qauselec</i>
7	<i>qcement</i>
8	<i>qgas</i>
9	<i>departures (visshort)</i>

Графики прогнозируемого и спрогнозированного временного ряда построить с использованием функции *ggplot()*.

## Классификация объектов

Задача классификации очень часто встречается на практике. Если распознаются только два класса, то в выходном слое персептрона находится только один нейрон, который обладает двумя реакциями. Если классов больше двух, то для каждой группы устанавливается свой нейрон.

Рассмотрим основные приемы часто используемые при анализе данных для множественной классификации. Проще всего их продемонстрировать на классическом примере классификации сортов ирисов. Набор *iris* [4], который включает 150 измерений параметров длины и ширины чашелистика (*sepal*) и лепестка (*petal*) трех сортов ирисов Фишера: щетинистого (*Setosa*), виргинского (*Virginica*) и разноцветного (*Versicolor*), по 50 измерений для каждого сорта.

Функция *summary()*, запущенная в консоли *RStudio*, позволяет ознакомиться с основными параметрами этого набора данных *iris*:

```
> summary(iris)
   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width      Species
   Min. :4.300   Min. :2.000   Min. :1.000   Min. :0.100   setosa   :50
   1st Qu.:5.100 1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300  versicolor:50
   Median :5.800  Median :3.000  Median :4.350  Median :1.300  virginica :50
   Mean   :5.843  Mean   :3.057  Mean   :4.358  Mean   :1.587
   3rd Qu.:6.400 3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
   Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500
```

Отсюда видно, что имеется четыре входных параметра (*Sepal.Length*, *Sepal.Width*, *Petal.Length*, *Petal.Width*) и три категории сорта (*species*) ирисов. Исходя из этого, можно предложить архитектуру персептрона, изображенную на рис. 8, где в скрытом слое условно изображено 6 нейронов.

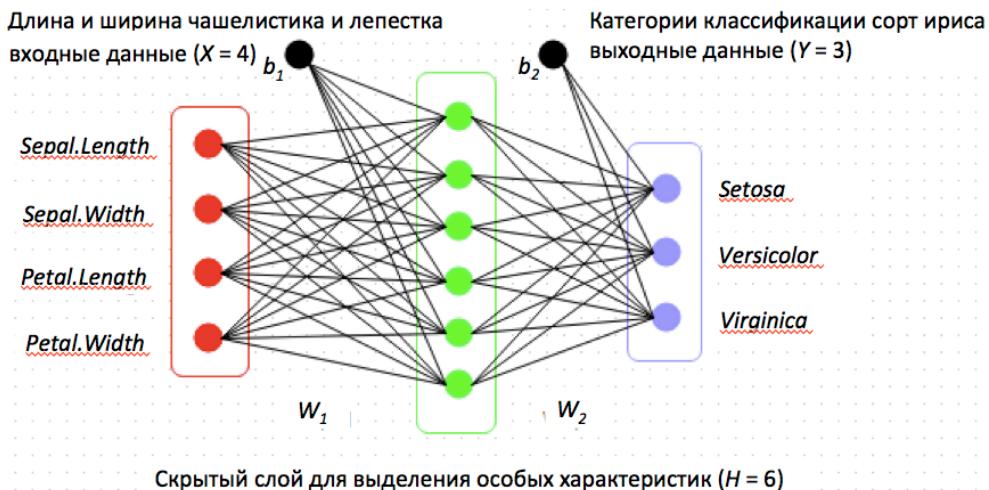


Рис. 8. Архитектура персептрона для классификации данных *iris*

При подготовке наборов данных (*data set*) весь набор делят на две части – обучающий или тренировочный набор (*train set*) и проверочный или тестовый набор (*test set*). Часто соотношение составляет 70-75% и 30-25%. Несмотря на то, что набор данных *iris* состоит всего из 150 векторов, сделаем такое разбиение. Важно здесь то, что нужно разбить весь набор на две части так, чтобы в любой из них представители из каждой категории или класса были в равных долях. В листинге 3, решающем задачу классификации данных на основе модели *pnet*, это разбиение выполняется в строке 14, где в *samp* формируется по 35 случайных номеров векторов из каждой категории для обучающего набора.

Обратите внимание на интересный прием, который используется для формирования тестового набора. Если в обучающий набор *train.set* включаются все векторы с номерами из *samp* (строка 17), то в тестовый набор *test.set* – все остальные векторы и это записывается следующим образом *data.iris[-samp, ]* (строка 18).

В листинге 3 использовано всего 2 нейрона в скрытом слое *size* = 2 и, как мы увидим их вполне достаточно для решения этой простой задачи. Кроме наборов входных *train.set* и выходных *targets[samp, ]* обучающих векторов и значения *size*, которые всегда необходимо формировать, задан только один параметр *maxit*, определенный по умолчанию. Его значение по умолчанию равно *maxit* = 100, а мы задали значение *maxit* = 500.

### Листинг 3. Классификация набора iris

```
1 library(nnet)
2
3 # Загрузка данных iris во фрейм data.iris
4 data.iris <- data.frame(iris)
5
6 # Исключение столбца Species
7 data.iris$Species <- NULL
8
9 # Формирование выходных реакций в виде трехкомпонентного вектора
10 targets <- class.ind(c(rep('s', 50), rep('c', 50), rep('v', 50)))
11
12 # Формирование номеров векторов тренировочного набора
13 set.seed(55)
14 samp <- c(sample(1:50,35), sample(51:100,35), sample(101:150,35))
15
16 # Формирование обучающего и тестового наборов
17 train.set <- data.iris[samp,]
18 test.set <- data.iris[-samp,]
19
20 # Обучение сети net.iris
21 set.seed(5)
22 net.iris <- nnet(dtrain.set,targets[samp,], size = 2, maxit = 500)
23
24 # Проверка сети на тестовом наборе
25 net.pred <- predict(net.iris, test.set)
26
27 # Построение проверочной таблицы
28 table(max.col(targets[-samp,]), max.col(net.pred))
```

Поскольку, по умолчанию параметр *trace* = TRUE, визуальное отслеживание процесса обучения сети включено и мы увидим следующее:

```
> net.iris <- nnet(train.set,targets[samp,], size = 2, maxit = 500)
# weights: 19
initial value 78.686545
iter 10 value 43.022535
iter 20 value 25.951476
iter 30 value 24.543711
iter 40 value 24.161124
iter 50 value 23.892634
iter 60 value 22.477187
iter 70 value 22.342019
iter 80 value 22.106268
iter 90 value 21.860257
iter 100 value 21.258000
iter 110 value 20.159820
iter 120 value 19.850850
iter 130 value 19.684387
iter 140 value 19.446336
iter 150 value 19.419291
final value 19.418854
converged
```

Мы видим, что сформированная функцией *nnet()* сеть *net.iris* включает 19 обучаемых весов (объясните почему) и ошибка обучения или потери уменьшаются в процессе итераций. В итоге, для сходимости процесса обучения в нашем случае потребовалось 150 итераций. Таким образом, следует уяснить, что сеть закончит обучение либо выполнив число итераций, заданное параметром *maxit*, либо достигнув заданной точности (параметрами *abs.tol* и *reltol*).

Для оценки качества классификации удобно использовать проверочную матрицу, которая показывает, как сеть справилась с задачей. Для построения такой матрицы можно использовать две функции *table()* и *max.col()*, а в качестве параметров то, что требовалось получить на выходе сети и то, что она получила. В нашем случае результат выполнения кода строки 28 будет следующим:

```
> table(max.col(targets[-samp,]), max.col(net.pred))
  1 2 3
1 14 0 1
2 0 15 0
3 0 0 15
```

Номера строк и столбцов соответствуют трем классам, на которые сеть должна разбить входные векторы. Очевидно, поскольку в тестовом наборе мы оставили по 15 векторов каждого класса, идеальным результатом является диагональная матрица с элементами 15, 15, 15. Полученный результат говорит о том, что сеть выдала только одно неверное предсказание, отнеся вектор первого класса к третьему. Попробуйте устраниТЬ этот недочет и поделитесь полученным результатом.

### Лабораторная работа № 3

Для выполнения лабораторной работы воспользуемся данными доступными на ресурсе <http://archive.ics.uci.edu/ml/datasets/Vertebral+Column>. Загрузите архив *vertebral\_column\_data.zip* и выберите из этого архива файл *column\_3C.dat*.

Файл содержит результаты рентгенологического исследования 310 пациентов. Каждый пациент представлен в наборе данных шестью параметрами, определяемыми формой и ориентацией таза и поясничного отдела позвоночника в следующем порядке: угол падения таза, наклон таза, угол поясничного лордоза, наклон крестца, радиус таза и степень спондилолистеза. Для обозначений классов используется следующее соглашение: DH (грыжа диска), спондилолистез (SL) и нормальный (NO).

Подготовить набор данных используя файл *column\_3C.dat*. Построить и обучить сеть для классификации диагноза пациентов. Отчет должен

содержать подготовку данных для обучения сети. Обоснование выбранной архитектуры. Проверочную матрицу для оценки качества классификации.

## Построение автокодировщиков

Автокодировщики представляют интересный тип нейронных сетей, задачей которых является максимально возможное сжатие входного вектора и его воссоздание из этого сжатого представления. В связи с этим, характерной особенностью автокодировщиков является совпадение количества нейронов на входном и выходном слое. Сама по себе способность сжимать данные используется редко, однако автокодировщики можно использовать для предобучения, например, когда стоит задача классификации, а размещенных пар слишком мало, либо когда просто надо научиться различать полезные свойства входного сигнала.

Рассмотрим пример построения, обучения и проверки автокодировщика для букв латинского алфавита в стилизации *art1\_letters.pat* из набора данных *snnsData* библиотеки *RSSNS*. Для построения используем модель персептрона *neuralnet* с числом входов и выходов 35 и числом нейронов в скрытом слое *hidden = 10*. Число входов и выходов сети определяется длиной вектора представления букв  $7 \times 5 = 35$ , а число нейронов скрытого слоя определялось в процессе обучения.

*Листинг 4. Автокодировщик букв латинского алфавита art1\_letters.pat*

```
1 library(RSNNS)
2 # Подготовка обучающей выборки
3 data(snnsData)
4 inputs <- snnsData$art1_letters.pat
5 print.alphabet(inputs)
6 train.data=data.frame(inputs,inputs)
7
8 # Формирование формул для входов и выходов
9 x.names <- paste0("in", 1:35, collapse = " + ")
10 y.names <- paste0("in", 1:35, collapse = " + ")
11 form <- as.formula(paste(x.names, " ~ ", y.names))
12
13 # Обучение автокодировщика
14 set.seed(55)
15 net <- neuralnet(formula = form, train.data, hidden=10,threshold=0.0006)
16
17 # Тестирование обученного автокодировщика
18 test.data=data.frame(inputs)
19 y.predict=predict(net,test.data)
20 print.alphabet(y.predict)
```

Подготовка обучающей выборки в данном случае не составляет труда, так как входной и выходной наборы совпадают и взяты

непосредственно из библиотеки (строка 4 – 6, листинг 4). Для визуализации алфавита использована функция *print.alphabet()*, ее содержание раскрыто в листинге 5.

Обратите особое внимание на способ формирования формул для входов и выходов автокодировщика. Поскольку их размерность равна 35, то вручную записывать такие последовательности трудоемко. Поэтому удобно воспользоваться функциями конкатенации *paste0()* и *as.formula()* (строка 9 – 11, листинг 4), результатом использования которых получена необходимая формула. Для проверки правильности формирования формулы посмотрим содержимое переменной *form*:

```
> form
in1 + in2 + in3 + in4 + in5 + in6 + in7 + in8 + in9 + in10 +
  in11 + in12 + in13 + in14 + in15 + in16 + in17 + in18 + in19 +
  in20 + in21 + in22 + in23 + in24 + in25 + in26 + in27 + in28 +
  in29 + in30 + in31 + in32 + in33 + in34 + in35 ~ in1 + in2 +
  in3 + in4 + in5 + in6 + in7 + in8 + in9 + in10 + in11 + in12 +
  in13 + in14 + in15 + in16 + in17 + in18 + in19 + in20 + in21 +
  in22 + in23 + in24 + in25 + in26 + in27 + in28 + in29 + in30 +
  in31 + in32 + in33 + in34 + in35
```

Как видим, сначала сформированы имена выходов, объединенные знаком «+», а затем, после знака «~», имена входов. Правда, здесь они совпадают.

Следует отметить, что функция *neuralnet()* позволяет подбирать оптимальные значения для более, чем десятка параметров. Здесь мы задаем только два: число нейронов в скрытом слое *hidden* = 10 и достаточно низкий порог функции ошибки *threshold* = 0.0006.

Обученный автокодировщик тестируем на том же входном наборе, на котором обучали с помощью функции *predict()*.

#### Листинг 5. Функция визуализации алфавита

```
1 library(RSNNS)
2 print.alphabet <- function(x) {
3   letter <- matrixToActMapList(x, nrow = 7)
4   par(mar=c(1,1,1,1)) #
5   par(mfrow=c(4,7))
6   n <- c(1:26)
7   for (i in n) plotActMap(letter[[i]])
8 }
```

Для визуализации результатов обучения удобно использовать описанную выше функцию *print.alphabet()*. На рис. 9 приведен результат работы обученной сети в сравнении с обучающей выборкой.

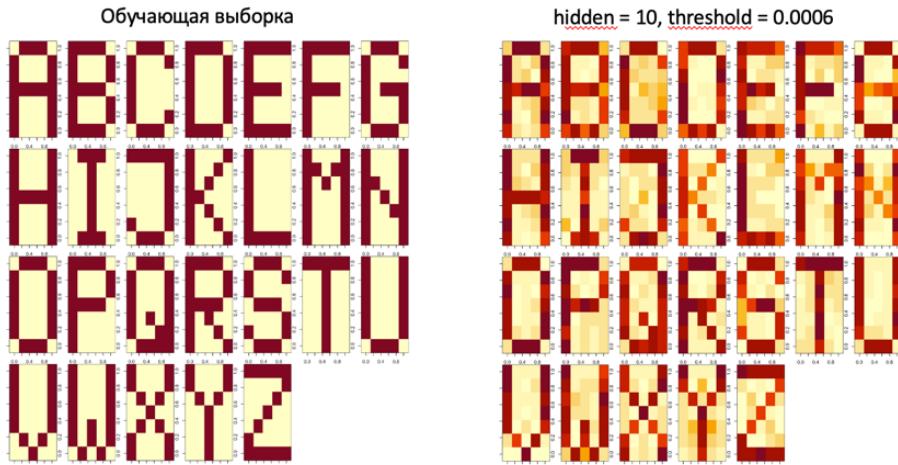


Рис. 9. Обучающая выборка и восстановленный алфавит при сжатии 3.5

В скрытом слое было использовано 10 нейронов, поэтому коэффициент сжатия входного вектора равен  $35/10 = 3.5$ . Попробуйте менять параметры сети (*hidden* и *threshold*) с тем, чтобы получить качественное восстановление для других коэффициентов сжатия.

#### Лабораторная работа № 4

Построить автокодировщик для заданной обучающей выборки. Отчет должен содержать подготовку данных для обучения сети. Формирование формул для входа и выхода. Визуализацию обучающей выборки и восстановленного образца. Структурную схему обученной сети без указания весов. Варианты заданий приведены в табл. 4, где указаны названия файлов с обучающей выборкой, которые доступны из библиотеки *RSNNS* [4].

Таблица 4

Варианты задания

Вариант	Обучающая выборка
Четная цифра	<i>snnsData\$letters_auto.pat</i>
Нечетная цифра	<i>snnsData\$artmap_test.pat</i>

Оценку максимально возможного сжатия производить визуально. Визуализировать обучающие выборки можно с использованием функции листинга 5, выполняя вызовы *print.alphabet(snnsData\$letters\_auto.pat,14)* и *print.alphabet(snnsData\$artmap\_test.pat,8)*

## 2. ОГРАНИЧЕННЫЕ МАШИНЫ БОЛЬЦМАНА

Ограниченнные машины Больцмана (*restricted Boltzmann machine - RBM*) являются особым видом генеративных стохастических нейронных сетей, которые обучаются методом градиентного спуска с контрастивной дивергенцией [12]. На практике *RBM* используются для снижения размерности данных, классификации и выделения признаков. Большой практический интерес представляют также сети глубокого доверия (*deep belief network - DBN*), которые строятся путем специального каскадирования *RBM*.

Изучение моделей *RBM* и *DBN* построим на базе функций библиотеки *Timo Matzen* из [13].

Функции *RBM()* и *StackRBM()* позволяют изучать соответствующие модели, а функции *ReconstructRBM()* и *PredictRBM()* предоставляют возможность визуализировать реконструированные векторы и оценить качество классификации.

**Листинг 6.** Функция подготовки изображений

```
1 image_conversion <- function(dir_path, width, height) {  
2   # Формирование списка файлов с изображениями из директория  
3   images_names <- list.files(dir_path)  
4   # Визуализация процесса обработки изображений  
5   print(paste("Обработка:", length(images_names), "изображений"))  
6   # Функция обработки изображения  
7   image_processing <- pblapply(images_names, function(imgname) {  
8     # Чтение изображения  
9     img <- readImage(file.path(dir_path, imgname))  
10    # Изменение размеров  
11    img_resized <- resize(img, w = width, h = height)  
12    # Преобразование в черно-белое  
13    img_gray <- channel(img_resized, "gray")  
14    # Получение матрицы размера weight x height  
15    img_matrix <- img_gray@.Data  
16    img_matrix <- rotate(img_matrix, 90)  
17    # Запись как вектор  
18    img_vector <- as.vector(t(img_matrix))  
19    return(img_vector)  
20  })  
21  # Формирование матрицы всех изображений из директория  
22  img_matrix <- do.call(rbind, image_processing)  
23  return(img_matrix)  
24}
```

Часто в качестве обучающего набора используются изображения. Когда подобные наборы формируются самостоятельно, удобно иметь автоматизированное средство для подобной подготовительной работы. В качестве подобного средства предлагается функция подготовки изображений

*image\_conversion()*, представленная в листинге 6. Функция обрабатывает все изображения размещенные в некотором директории, выполняя следующие действия: приводит размеры изображения к заданным (*width x height*), преобразует в черно-белое, представляет как вектор и формирует результирующую матрицу всех изображений. Возвращаемая матрица имеет размеры – число строк равно числу обработанных изображений, число столбцов равно произведению *width x height*.

Для использования функции *image\_conversion()* необходимо предварительно заполнить выделенный директорий файлами с изображениями. Данная функция построена на базе возможностей предоставляемых библиотеками *EBImage* и *pbapply*. На рис. 10 приведен результат использования этой функции (первый ряд – приведение к размеру 28x28, второй ряд – перевод в черно-белое изображение).

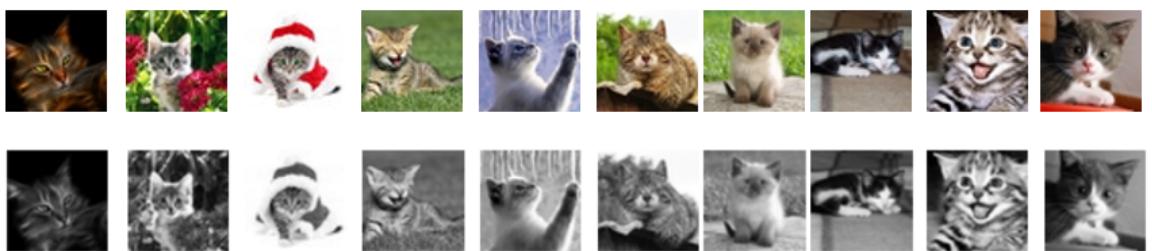


Рис. 10. Преобразование изображений

Рассмотрим варианты использования функций библиотеки *RBM* для понижения размерности обучающей выборки изображений. В качестве обучающей выборки загрузим изображения котят из веб-ресурса <https://ru.wallpaper.mob.org/> и разместим их в рабочем директории в папке *Cats* (пример части загруженных изображений на рис. 10). Указанный ресурс является свободно распространяемым и может быть использован для загрузки изображений различных категорий.

Библиотека *RBM* предоставляет как функцию *RBM()*, моделирующую отдельно взятую ограниченную машину Больцмана, так и функцию *StackRBM()*, позволяющую формировать стек из этих машин. В листинге 7 приведен пример использования обеих функций *RBM()* и *StackRBM()*.

После загрузки необходимых библиотек с помощью описанной выше функции *image\_conversion()* формируется обучающая выборка изображений размером 28x28 пикселей. Для ограниченной машины Больцмана задается скрытый слой из 196 нейронов и устанавливается 400 итераций (эпох) обучения.

Чтобы обеспечить наглядное сравнение результатов обучения отдельной ограниченной машины Больцмана и стека из этих машин вводится параметр *mflow = c(2,10)* задающий печать двух строк по 10 изображений.

Для стека ограниченных машин Больцмана задается три скрытых слоя с числом нейронов 392, 196 и 98. В качестве дополнительных параметров задается мини-батч размером 10, а также скорость обучения и коэффициенты *lambda* и *momentum*.

#### *Листинг 7. Обучение ограниченных машин Больцмана*

```

1 library(RBM)
2 library(EBImage)
3 library(pbapply)
4
5 # Подготовка обучающей выборки
6 trainData <- image_conversion("Cats", 28, 28)
7
8 # Обучение ограниченной машины Больцмана
9 set.seed(55)
10 mod <- RBM(trainData, n.iter = 400, n.hidden = 196)
11
12 # Подготовка для печати 2-х строк по 10 изображений
13 par(mar=c(1,1,1,1))
14 par(mfrow=c(2,10))
15
16 # Реконструкция 10 реконструированных изображений (hidden = 196)
17 for(i in 1:10)
18   image(ResultRBM(trainData[i, ], model = mod))
19
20 # Обучение стека ограниченных машин Больцмана
21 set.seed(55)
22 modCat <- StackRBM(x = trainData, n.iter = 400, layers = c(392, 196, 98),
23                      learning.rate = 0.1, size.minibatch = 10, lambda = 0.1,
24                      momentum = 0.5)
25 # Реконструкция 10 реконструированных изображений (hidden = 98)
26 for(i in 1:10)
27   image(ResultRBM(trainData[i, ], model = modCat, layers = 3))

```

Для того, чтобы обеспечить вывод в цикле произвольного числа реконструированных изображений оригинальная функция *ReconstructRBM()*, представленная в листинге 8, была частично изменена и обозначена *ResultRBM()* (листинг 9).

#### *Листинг 8. Оригинальная функция печати реконструированных изображений*

```

1 ReconstructRBM <- function (test, model, layers = 1)
2 {
3   if (!missing(layers)) { }
4   if (!is.null(dim(test))) { }
5   if (any(!is.numeric(test))) { }
6   if (any(!is.finite(test))) { }
7   if (length(model) != layers) { }
8   test <- matrix(test, nrow = 1)
9   V <- cbind(1, test[1, , drop = FALSE])
10  if (missing(layers)) { }
11  else { }
12    par(mfrow = c(1, 2))
13    image(matrix(V[, -1], nrow = sqrt(ncol(test))))
14    title(main = "Original Image", font.main = 4)
15    image(matrix(V.rec[, -1], nrow = sqrt(ncol(test))))
16    title(main = "Reconstruction Model", font.main = 4)
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42 }

```

Измененные части в оригинальной и измененной версии функции выделены желтым цветом.

#### Листинг 9. Измененная функция печати реконструированных изображений

```

1 ResultRBM <- function (test, model, layers = 1)
2 {
3   if (!missing(layers)) {  }
4   if (!is.null(dim(test))) {  }
5   if (any(!is.numeric(test))) {  }
6   if (any(!is.finite(test))) {  }
7   if (length(model) != layers) {  }
8   test <- matrix(test, nrow = 1)
9   V <- cbind(1, test[1, , drop = FALSE])
10  if (missing(layers)) {  }
11  else {  }
12    return(matrix(V.rec[, -1], nrow = sqrt(ncol(test))))
13 }

```

Результат работы сети *mod*, моделирующей ограниченную машину Больцмана и *modCat*, моделирующей стек ограниченных машин, приведены на рис. 11.

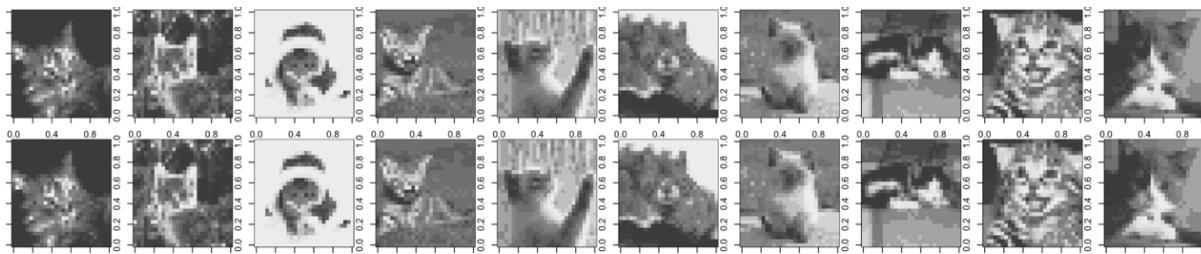


Рис. 11. Результат работы нейронной сети *mod* и *modCat*

Функция *DBN()* обучает сеть глубокого доверия, начиная с предварительно обученного стека неконтролируемых *RBM*, используя функцию *StackRBM()*, а затем добавляется контролируемый выходной слой. Результирующая сеть *DBN* настраивается по контролируемому критерию с использованием обратного распространения. Качество обучения позволяет оценить функция *PredictDBN()*.

Рассмотрим использование сети глубокого доверия для классификации образов на примере функции *DBN()*. В качестве обучающей и тестовой выборки возьмем классический набор *MNIST* (листинг 10).

#### Листинг 10. Классификация *MNIST* сетью глубокого доверия

```

1 library(RBM)
2 data(MNIST)
3 modDBN <- DBN(MNIST$trainX, MNIST$trainY, n.iter = 10000, nodes = c(500, 300, 150),
4   learning.rate = 0.5, size.minibatch = 10, n.iter.pre = 150,
5   learning.rate.pre = 0.1, verbose = TRUE)
6 PredictDBN(MNIST$testX, MNIST$testY, model = modDBN, layers = 4)

```

При вызове функции `DBN()` использовано много входных параметров, определяющих особенности обучения модели `modDBN`. Для предварительного тренинга (предобучения со скоростью 0.1) задано 150 эпох для каждого уровня `RBM`. Обучение на основе градиентного спуска определено на 10000 эпохах со скоростью 0.5. Размер батча установлен равным 10. Параметр `verbose = TRUE` разрешает визуализацию процесса обучения – печать значения потерь (`cost`) на каждой эпохе.

Для оценки результата классификации сетью глубокого доверия использована функция `PredictDBN()`, которая показывает достаточно неплохую точность 0.916:

		truth									
Preds		0	1	2	3	4	5	6	7	8	9
0	192	0	6	1	0	4	2	0	2	2	
1	0	211	0	0	0	1	0	0	3	0	
2	0	1	170	5	2	3	0	4	4	2	
3	2	1	3	177	0	6	0	0	4	2	
4	0	0	3	0	215	1	3	4	4	10	
5	1	1	1	7	1	138	2	0	6	1	
6	0	2	3	1	1	1	207	0	2	0	
7	1	1	0	1	0	0	0	192	2	4	
8	1	7	2	2	3	6	2	1	151	4	
9	0	1	2	4	4	2	0	1	1	180	

\$Accuracy
[1] 0.9165

Следует пояснить, почему при использовании функции `PredictDBN()` указан параметр `layers = 4`, хотя сеть `modDBN` включает три слоя `nodes = c(500, 300, 150)`. Это сделано с учетом добавления контролируемого выходного слоя.

## Лабораторная работа № 5

Построить ограниченную машину Больцмана для понижения размерности обучающей выборки. В качестве последней возьмите 10 произвольных изображений. Для их предварительной обработки используйте функцию листинга 6, задавая размеры 100 x 100 пикселов. Подберите параметры `RBM` для получения удовлетворительного качества. Визуализируйте полученные результаты.

Сформируйте стек ограниченных машин Больцмана с помощью функции `StackRBM()`. Проведите исследование с тем же набором изображений. Подберите параметры стека для получения удовлетворительного качества. Визуализируйте полученные результаты.

Проделайте сравнительный анализ возможностей, предоставляемых для понижения размерности обучающей выборки отдельной `RBM` и стеком из `RBM`.

### 3. СЕТИ КОХОНЕНА

Сети Кохонена – это одна из разновидностей нейросетевых алгоритмов. Алгоритм функционирования самообучающихся карт *SOM* (*Self Organizing Maps*) представляет собой один из вариантов кластеризации многомерных векторов [10].

Для моделирования сетей Кохонена используются функции *somgrid()* и *som()* из пакета *kohonen*. Для обученной сети с помощью функции *plot()* становится доступным для визуализации следующий комплект карт:

*codes* - показывается распределение по решетке соотношения долей участия отдельных исходных переменных;

*counts* - число исходных объектов в каждом узле сети;

*mapping* - координаты исходных объектов на сформированной карте;

*property*, *quality*, *dist.neighbours* - различными цветами изображается целый набор свойств каждого узла: доли участия отдельных исходных переменных, меры парных или средних расстояний между нейронами и т.д.

Рассмотрим пример, в котором используем три указанные функции. В качестве исходных данных для анализа используем фрейм данных *wines*, поставляемый с библиотекой *kohonen*. Фрейм *wines*, включает 177 строк и тридцать столбцов, где приведены результаты химического анализа вин, выращенных в одном и том же регионе Италии (Пьемонт), но полученных из трех разных сортов винограда: Неббиоло, Барбера (*Barbera*) и Гриньолино (*Grignolino*). Вино из винограда сорта Неббиоло называется Бароло (*Barolo*). Данные содержат количества нескольких компонентов, присутствующих в каждом вине, а также некоторые спектроскопические переменные.

Для каждой строки фрейма *wines* переменная *vintages* содержит метки классов, соответствующие винтажам *Barbera*, *Grignolino* и *Barolo*.

**Листинг 10.** Моделирование карт Кохонена

```
1 library(kohonen)
2 data(wines)
3 set.seed(5)
4
5 # Формирование SOM карты
6 sommap <- som(scale(wines), grid = somgrid(10, 17, "hexagonal"))
7
8 # Разбиение карты на отдельные кластеры
9 groups<-3
10 som.hc <- cutree(hclust(dist(sommap$codes[[1]])), groups)
11
12 # Построение SOM карты
13 plot(sommap, type="codes", bgcol=rainbow(groups)[som.hc],
14      main="SOM карта вин (wines)")
15
16 # Прорисовка кластеров на карте
17 add.cluster.boundaries(sommap, som.hc)
```

Как видно из листинга 10 перед кластеризацией исходные данные отмасштабированы функцией `scale()`. Для карты функцией `somgrid()` сформирована сетка размером 10 x 17 и задана структура «шестиугольник».

С помощью функции `cutree()` мы осуществили разбиение всех вин на три кластера и с помощью функции `plot()` построили карту Кохонена (рис. 12). Дополнительная прорисовка кластеров выполнена с помощью функции `add.cluster.boundaries()`.

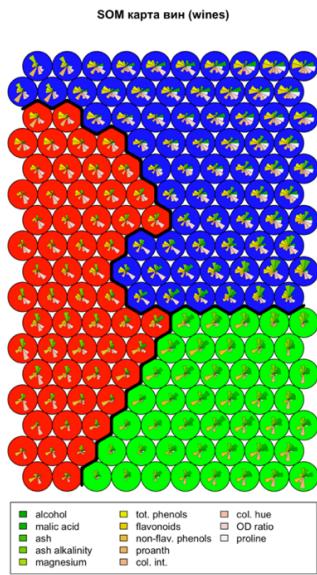


Рис. 12. Карта Кохонена для трех кластеров

Карта дополнена информацией о всех компонентах анализируемых векторов, с указанием названия компоненты и цвета отведенного для ее представления. Это позволяет визуально оценить качество кластеризации и принять решение об изменении числа кластеров для оптимизации разбиения.

### Лабораторная работа № 6

Построить карты Кохонена для тестовых наборов из ресурса *Wine Quality* (<http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/>) репозитария [6]. Варианты заданий приведены в табл. 5.

Таблица 5

Варианты задания

Вариант	Обучающая выборка
Четная цифра	<i>winequality-red.csv</i>
Нечетная цифра	<i>winequality-white.csv</i>

Отчет должен содержать описание тестового набора, подготовку обучающих данных, выбор и обоснование количества кластеров. Представленный в отчете код должен иметь подробный комментарий.

## 4. СВЕРТОЧНЫЕ СЕТИ

Сверточные нейронные сети (*CNN – Convolutional Neural Networks*) используют две основные операции – свертки и пулинга. В архитектуре сети слои, которые выполняют их, следуют друг за другом, постепенно извлекая и выделяя характерные признаки анализируемых данных. Когда признаки достаточно выделены, требуемый результат формируется с помощью полно связного слоя, в котором осуществляется анализ их взаимовлияния на итог исследования.

В отличие от классических нейронных сетей, для которых разработано большое число моделей, сверточные сети требуют «индивидуального» подхода к архитектуре. Поэтому для формирования *CNN* используют библиотеки, предоставляющие возможность формирования и объединения между собой отдельных слоев - свертки, пулинга и полно связного слоя.

Несмотря на то, что *CNN* специализирована для задач компьютерного зрения и распознавания изображений, она также оказалась полезной во многих других приложениях и часто используется на практике. Достаточно интересное погружение в сверточные сети можно найти на ресурсе [11].

Здесь мы остановимся на двух аспектах, связанных с использованием моделей *CNN* – это *переобучение* и *визуализация промежуточных активаций*.

*Переобучение* - распространенная проблема когда имеется только несколько образцов данных, на которых можно учиться. Это мешает модели хорошо обучиться. Один из методов, который помогает справиться с этой проблемой – приращение (*augmentation*) данных. Этот метод уменьшает переобучение за счет генерации большего количества обучающих данных из существующих выборок с помощью нескольких случайных преобразований, создающих правдоподобные изображения. Для модификации применяются такие операции, как сдвиг, переворачивание, масштабирование и тому подобные. Библиотека *keras* предоставляет удобные функции для реализации метода аугментации в реальном времени.

Пример использования данных функций представлен в листинге 11. В качестве образцов для аугментации выбран директорий *cats-dogs*, включающий *jpeg* изображения кошек и собак. Функция *image\_data\_generator()* позволяет определить необходимые диапазоны изменений параметров исходного изображения. В листинге 11 для этой функции использованы далеко не все доступные возможности настройки, подробно с назначением каждого параметра можно ознакомиться, например в [17].

Изображения из директория приводятся к одинаковому размеру 150x150 пикселов и преобразуются в тензор размерности 1,150,150,3. В строке 14 листинга 11, для простоты задается только одно 9-ое изображение.

Функция *flow\_images\_from\_data()* запускает работу генератора аугментации *augmentation\_generator*, который затем используется в цикле для

вывода 12 последовательных преобразований исходного изображения `fnames[[9]]`.

Листинг 11. Приращение обучающего набора путем аугментации

```
1 library(keras)
2 dir <- 'cats-dogs'
3 datagen <- image_data_generator(
4   rescale = 1/255,
5   rotation_range = 40,
6   width_shift_range = 0.2,
7   height_shift_range = 0.2,
8   shear_range = 0.2,
9   zoom_range = 0.2,
10  horizontal_flip = TRUE,
11  fill_mode = "nearest"
12 )
13 fnames <- list.files(dir, full.names = TRUE)
14 img_path <- fnames[[9]]
15 img <- image_load(img_path, target_size = c(150, 150))
16 img_array <- image_to_array(img)
17 img_array <- array_reshape(img_array, c(1, 150, 150, 3))
18
19 augmentation_generator <- flow_images_from_data(
20   img_array,
21   generator = datagen,
22   batch_size = 1
23 )
24
25 op <- par(mfrow = c(3, 4), pty = 's', mar = c(0, 0, 0, 0))
26 for (i in 1:12) {
27   batch <- generator_next(augmentation_generator)
28   plot(as.raster(batch[1,,,]))
29 }
```

Результат работы скрипта листинга 11 представлен на рис. 13. Отсюда видно, что трансформации параметров изображения осуществляются по случайному закону.



Рис. 13. Результат аугментации

Получаемые изображения существенно дополняют палитру обучающих данных, помогая справляться с проблемой переобучения нейронной сети.

*Визуализация промежуточных активаций* помогает понять как преобразуются входные данные по мере прохождения по слоям нейронной сети (вывод результата функций активации слоя называют его *активацией*). Карты признаков изучаемые сверточными нейронными сетями, легко поддаются визуализации благодаря тому, что представляют визуальные объекты. Визуализация активаций заключается в отображении карт признаков, которые выводятся разными сверточными и объединяющими слоями сети в ответ на определенные входные данные.

Рассмотрим особенности использования функций, предоставляемых библиотекой *keras* для визуализации активаций. В качестве объекта исследования можно загрузить любую сохраненную модель *CNN*. Это выполняется с помощью функции *load\_model\_hdf5()*. Загрузим обученную модель из [19] и создадим экземпляр модели из входного тензора и списка выходных тензоров (листинг 12 ).

**Листинг 12.** Загрузка модели и создание экземпляра

```
model <- load_model_hdf5("cats_and_dogs_small_1.h5")
layer_outputs <- lapply(model$layers[1:8], function(layer) layer$output)
activation_model <- keras_model(inputs = model$input, outputs = layer_outputs)
activations <- activation_model %>% predict(img_tensor)
```

Характеристика слоев используемой нейронной сети приведены в табл. 6. По мере продвижения от входа к выходу размеры карт признаков постепенно уменьшаются, а их число увеличивается.

Таблица 6  
Характеристика слоев нейронной сети *cats\_and\_dogs\_small\_1*

Номер и тип слоя	Размер карты признаков	Число карт
1 Свертка	148 x 148	32
2 Пулинг	74 x 74	32
3 Свертка	72 x 72	64
4 Пулинг	36 x 36	64
5 Свертка	34 x 34	128
6 Пулинг	17 x 17	128
7 Свертка	15 x 15	128
8 Пулинг	7 x 7	128

Созданный с помощью функции *keras\_model()* экземпляр модели *activation\_model* позволяет сформировать активацию для любой карты признаков в каждом из восьми слоев. Для ее визуализации удобно использовать функцию *plot\_channel()* представленную в листинге 13.

Для анализа получаемых послойных активаций выберем произвольное входное изображение  $img$  из тестовой выборки, преобразуем его в тензор  $img\_tensor$  размерности 1, 150, 150, 3 и нормализуем  $img\_tensor = img\_tensor / 255$ . Используя полученный  $img\_tensor$  в качестве входного параметра функции  $predict()$  будем в цикле использовать функцию  $plot\_channel()$ .

**Листинг 13.** Функция визуализации карты признаков

```
plot_channel <- function(channel) {
  rotate <- function(x) t(apply(x, 2, rev))
  image(rotate(channel), axes = FALSE, asp = 1,
        col = terrain.colors(12))
}
```

На рис. 14 приведены примеры визуализации одной, случайно выбранной карты признаков на каждом слое.

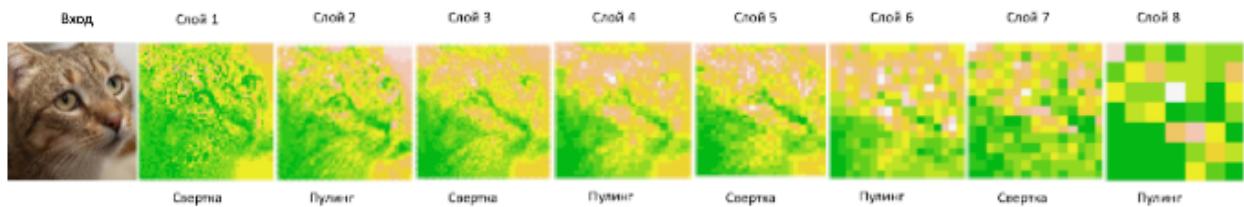


Рис. 14. Примеры визуализации карты признаков по слоям

Как видно из рисунка, по мере продвижения по слоям активации изображения становятся все более абстрактными, а их визуальная интерпретация становится сложней. По сути они начинают кодировать высокоуровневые представления и несут все меньше информации об исходном изображении и все больше — о классе изображения. Последняя карта напоминает QR-код, в котором закодирована информация о классе принадлежности изображения поданного на вход.

## Лабораторная работа № 7

Изучите особенности использования функции  $image\_data\_generator()$ . Продемонстрируйте возможности управления всеми ее параметрами для формирования обучающих примеров. Какие параметры являются наиболее эффективными?

Выберите несколько произвольных изображений относящихся к двум классам, сформируйте достаточный обучающий набор на базе функции  $flow\_images\_from\_data()$  и постройте простую сверточную нейронную сеть решающую задачу классификации.

Дополнительно: используя информацию из [19], визуализируйте все карты признаков построенной сети для произвольного входного изображения.

## 5. ВАРИАЦИОННЫЕ АВТОКОДИРОВЩИКИ

Классический автокодировщик учится представлять входные данные в скрытом пространстве уменьшенных размеров. Он изучает произвольную функцию для выражения входных данных в сжатом скрытом представлении. Вариационный автокодировщик (*VAE*) вместо изучения произвольной функции изучает параметры распределения вероятностей сжатого представления. Если мы сможем выбирать точки из этого распределения, мы сможем сгенерировать новые данные. В этом смысле *VAE* относится к генеративным моделям нейросетей.

Вариационный автокодировщик состоит из двух сетей: сети кодера и сети декодера (рис. 15). Кодер — это нейронная сеть, которая принимает входные данные  $x$  и выдает скрытое представление  $z$ . Целью сети кодера является для каждого  $x$  предсказать два значения — среднее значение  $\mu$  и отклонение (дисперсию)  $\sigma$  нормального распределения, из которого сэмплируются случайные образцы  $z$ . По сути, кодер в *VAE* изучает распределение вероятностей  $P_\theta(z|x)$ , где  $\theta$  — параметр сети кодера.

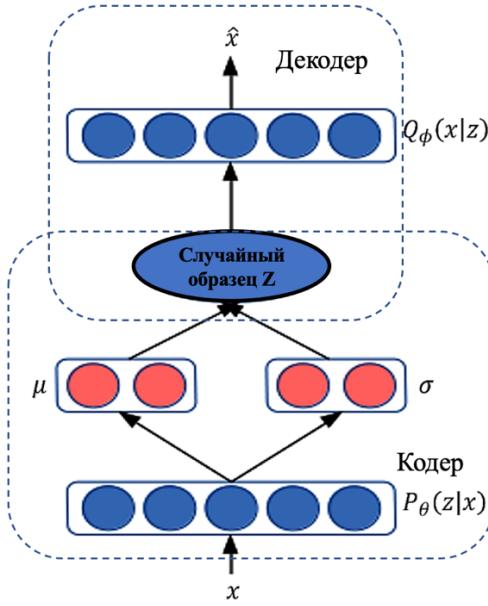


Рис. 15. Структура вариационного автокодировщика

Целью сети декодера является восстановление входных данных  $x$  из случайного образца  $z$  ( $z$  принадлежит распределению с параметрами  $\mu$  и  $\sigma$ ). Его задача — предсказать распределение вероятностей  $Q_\phi(x|z)$ , где  $\phi$  — параметр сети декодера.

В типичном *VAE* функция потерь состоит из двух частей: потери восстановления и потери регуляризатора. Для одного обучающего примера функция потерь задается следующим уравнением:

$$l(\theta, \phi) = E_{z \sim P_\theta(z|x)} [\log Q_\phi(x|z)] - KL[P_\theta(z|x) | N(0, I)]$$

Первый член уравнения — потери реконструкции, а второй — это расхождение Кульбака-Лейблера ( $KL$ ) между изученным распределением вероятностей, и истинным распределением скрытой переменной,. В  $VAE$  предполагается, что скрытые переменные происходят из стандартного нормального распределения, то есть  $P(z)$  есть  $N(0, 1)$ .

Рассмотрим пример реализации вариационного автокодировщика для генерации изображений [18]. В качестве изображений используется набор данных *Fashion MNIST*, который поставляется вместе с библиотекой *keras*. В листинге 14 загружаются необходимые библиотеки и набор данных разбивается на две части: для обучения  $x\_train$  и тестирования  $x\_test$  (строки 1 - 8).

#### Листинг 14. Формирование вариационного автокодировщика

```

1 library(keras)
2 library(abind)
3 library(grid)
4 mnist <- dataset_fashion_mnist()
5 x_train <- mnist$train$x/255
6 x_test <- mnist$test$x/255
7 x_train <- array_reshape(x_train, c(nrow(x_train), 784), order = "F")
8 x_test <- array_reshape(x_test, c(nrow(x_test), 784), order = "F")
9 # Шаг 1: параметры обучения
10 batch_size <- 100L
11 input_dim <- 784L
12 latent_dim <- 2L
13 epochs <- 10
14 # Шаг 2: входной и скрытый слои
15 input <- layer_input(shape = c(input_dim))
16 x <- input %>% layer_dense(units = 256, activation = "relu")
17 # Шаг 3: среднее и логарифмическая дисперсия скрытого распределения
18 z_mean <- x %>% layer_dense(units = latent_dim, name = "mean")
19 z_log_sigma <- x %>% layer_dense(units = latent_dim, name = "sigma")
20 # Шаг 4: функция сэмплирования
21 sampling <- function(arg) {
22   z_mean <- arg[, 1:(latent_dim)]
23   z_log_var <- arg[, (latent_dim + 1):(2 * latent_dim)]
24   epsilon <- k_random_normal(shape = list(k_shape(z_mean)[1],
25                               latent_dim), mean = 0, stddev = 1)
26   z_mean + k_exp(z_log_sigma) * epsilon
27 }
28 # Шаг 5: случайная точка из скрытого распределения
29 z <- layer.concatenate(list(z_mean, z_log_sigma)) %>%
30 layer_lambda(sampling)
31 # Шаг 6: скрытые слои и выход декодера
32 x_1 <- layer_dense(units = 256, activation = "relu")
33 x_2 <- layer_dense(units = input_dim, activation = "sigmoid")
34 vae_output <- x_2(x_1(z))
35 # Шаг 7: вариационный автокодировщик VAE
36 vae <- keras_model(input, vae_output)

```

В листинге 14 представлены последовательные шаги формирования вариационного автокодировщика.

На шаге 1 задаются параметры обучения – размер батча, размерности входа, параметров  $\mu$  и  $\sigma$  и количество эпох.

На шаге 2 определяется входной и скрытый слой сети кодера. Далее, на шаге 3 формируется скрытый слой для представления параметров  $\mu$  и  $\sigma$ .

На шаге 4 определяется функция сэмплирования, которая позволяет генерировать новую точку из скрытого пространства, а на шаге 5 формируется слой, в котором из параметров  $\mu$  и  $\sigma$  скрытого распределения генерируется случайный образец  $z$ .

На шаге 6 мы создаем два скрытых слоя для части декодера  $VAE$  и объединяем их, чтобы создать выходной слой. Наконец, на шаге 7 мы формируем вариационный автокодировщик. Используя функцию *summary()* можно изучить подробную структуру модели *vae*.

#### Листинг 15. Обучение и тестирование

```
37 - #####
38 # Шаг 8: создание отдельной сети кодера
39 encoder <- keras_model(input, c(z_mean,z_log_sigma))
40 # Шаг 9: создание отдельной модели декодера
41 decoder_input <- layer_input(k_int_shape(z)[-1])
42 decoder_output <- x_2(x_1(decoder_input))
43 decoder <- keras_model(decoder_input,decoder_output)
44 # Шаг 10: определим свою функцию потерь (ошибок)
45 vae_loss <- function(x, decoded_output){
46   rec_loss <- (input_dim/1.0)*loss_binary_crossentropy(x,
47                                         decoded_output)
48   kl_loss <- -0.5*k_mean(1 + z_log_sigma - k_square(z_mean) -
49                         k_exp(z_log_sigma), axis = -1L)
50   rec_loss + kl_loss
51 }
52 # Шаг 11: Компиляция и обучение модели VAE
53 # компиляция VAE
54 vae %>% compile(optimizer = "rmsprop", loss = vae_loss)
55 # обучение VAE
56 vae %>% fit(
57   x_train, x_train,
58   shuffle = TRUE,
59   epochs = epochs,
60   batch_size = batch_size,
61   validation_data = list(x_test, x_test)
62 )
63 # Шаг 11: визуализация пяти изображений сгенерированных VAE
64 random_distribution = array(rnorm(n = 100,mean = 0,sd = 10),dim =
65                           c(5,2))
66 predicted = NULL
67 for(i in seq(1,nrow(random_distribution))){
68   one_pred = predict(decoder,matrix(random_distribution[i,],ncol=2))
69   predicted = abind(predicted,array_reshape(one_pred,dim =
70                     c(28,28)),along = 2)
71 }
72 options(repr.plot.width=10, repr.plot.height=1)
73 grid.raster(predicted,interpolate=FALSE)
```

В листинге 15 описаны шаги, необходимые для обучения и тестирования сформированного *VAE*. На шаге 8 и 9 создаются отдельные модели кодера и декодера, которые используются при реконструкции изображения из точки скрытого пространства случайного распределения. На шаге 10 определяется функция потерь *vae\_loss()*, учитывающая как потери восстановления, так и потери регуляризатора.

На шаге 11, после компиляции, определяющей стандартный оптимизатор *rmsprop* и свою функцию потерь *vae\_loss()*, проводится обучение *VAE*. График изменения потерь при обучении *loss* и тестировании *val\_loss* от 2-ой до 10-ой эпохи приведен на рис. 16.

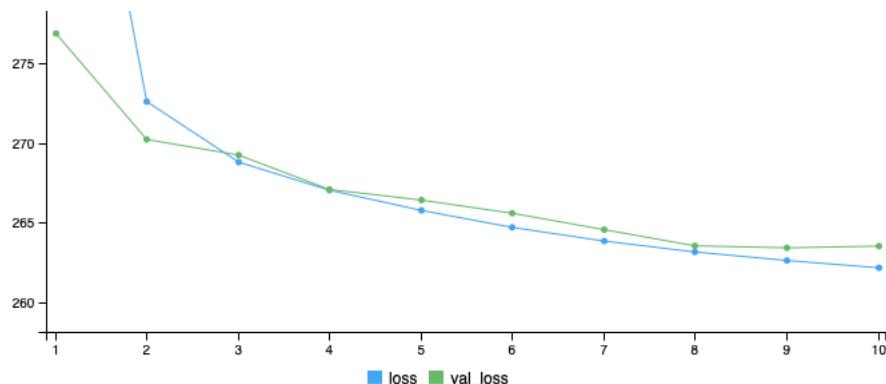


Рис. 16. Уменьшение потерь на 10 эпохах

Проверка работоспособности *VAE* выполняется на шаге 11. Для пяти случайных точек скрытого пространства из *random\_distribution* декодер в цикле формирует предсказания, то есть выходные векторы размерности 784 в переменной *one\_pred*. Эти векторы преобразуются в матрицы размерности 28 x 28, которые сохраняются в переменной *predicted*.



Рис. 17. Изображения сгенерированные *VAE*

С помощью функции *grid.raster()* сгенерированные *VAE* изображения визуализируются (рис. 17).

## Лабораторная работа № 8

Изучить особенности работы вариационного авто кодировщика. Построить *VAE* для генерации рукописных цифр, используя набор MNIST. Подобрать параметры обучения для получения качественной реконструкции.

## 6. РЕКУРРЕНТНЫЕ СЕТИ НА БАЗЕ LSTM

Рекуррентные сети часто используют для решения креативных задач. Рассмотрим пример использования модуля *LSTM* для реализации рекуррентной сети, которая будет генерировать текст из заданного начального набора, состоящего из трех слов. Исходный текст или целый набор текстов при работе с языковыми моделями принято называть корпусом.

Чтобы построить языковую модель, нужно «очистить» текст, используемый для обучения, и разбить его на токены. Для этих целей удобно использовать токенизатор *keras*, который разбивает корпус на список токенов, удаляет все знаки препинания, преобразует слова в нижний регистр и строит внутренний словарь на основе введенного текста. Словарь генерируемый токенизатором, представляет собой индексированный список, в котором слова индексируются по их общей частоте в наборе данных.

Возьмем в качестве исходного текста стихотворение К. Чуковского «Муха Цокотуха» и посмотрим насколько хорошо сеть может обучаться стихосложению. Подготовка исходных данных выполняется в листинге 16.

Листинг 16. Подготовка обучающих данных

```
1 library(keras)
2 library(readr)
3 library(stringr)
4 # чтение исходного текста
5 data <- read_file("fly_c.txt") %>% str_to_lower()
6 # формирование токенов
7 toks = text_tokenizer(num_words = 350, char_level = F)
8 toks %>% fit_text_tokenizer(data)
9 # представление исходного текста в виде строки токенов
10 text <- texts_to_sequences(toks, data)
11 text <- text[[1]]
12
13 # преобразование строки токенов в обучающую выборку
14 inp_len <- 3
15 X <- matrix(ncol = inp_len)
16 Y <- matrix(ncol = 1)
17 for(i in seq(inp_len, length(text))){}
18   if(i >= length(text)){break()}
19   start_idx <- (i - inp_len) + 1
20   end_idx <- i + 1
21   new_seq <- text[start_idx:end_idx]
22   X <- rbind(X, new_seq[1:inp_len])
23   Y <- rbind(Y, new_seq[inp_len+1])
24 }
25 X <- X[-1,]
26 Y <- Y[-1,]
27 Y <- to_categorical(Y, num_classes = toks$num_words)
```

Здесь исходный текст преобразовывается в последовательность токенов. Обратите внимание, что аргумент *num\_words* функции *text\_tokenizer()* определяет максимальное количество сохраняемых слов в зависимости от их частоты. Это означает, что в закодированной последовательности сохраняются только первые *n* часто встречающихся слов, в нашем случае 350.

Далее обучающая выборка формируется в виде матриц *X* и *Y*. Каждая строка входной матрицы *X* содержит три последовательные слова из текста, представленные соответствующими им токенами, а в матрице *Y* записывается токен слова, следующего за этими тремя. Длина начального набора *inp\_len* выбрана нами равной трем, но может быть любой.

Скрипт формирования и обучения *LSTM* модели представлен в листинге 17. Структура модели строится с помощью четырех функций – встраивания *layer\_embedding()*, определения числа модулей *layer\_lstm()*, формирования полносвязного слоя *layer\_dense()* и задания функции активации *layer\_activation()*. Сначала мы инициализировали последовательную модель, а затем добавили к ней слой внедрения. Далее мы добавили слой *LSTM* с 50 модулями. Поскольку предсказание слова – это проблема классификации, при которой следующее слово выбирается из словаря, мы добавили полносвязный слой с числом нейронов, равным количеству слов в словаре и с функцией активации softmax.

#### Листинг 17. Формирование и обучение *LSTM* модели

```
28 # Формирование и обучение модели нейронной сети
29 # формирование модели LSTM сети
30 model <- keras_model_sequential()
31 model %>%
32   layer_embedding(input_dim = toks$num_words,
33                   output_dim = 10, input_length = inp_len) %>%
34   layer_lstm(units = 50) %>%
35   layer_dense(toks$num_words) %>%
36   layer_activation("softmax")
37 summary(model)
38 # компиляция модели LSTM сети
39 model %>% compile(
40   loss = "categorical_crossentropy",
41   optimizer = optimizer_rmsprop(lr = 0.002),
42   metrics = c('accuracy')
43 )
44 # обучение модели LSTM сети
45 model %>% fit(X, Y, batch_size = 10, epochs = 100)
```

Для компиляции модели заданы параметры определяющие функцию потерь, алгоритм оптимизации со скоростью обучения *lr* = 0.002 и функцию, используемую для оценки производительности обученной модели.

Для функции обучения *fit()* задан размер пакета *batch\_size* = 10 и число эпох *epochs* = 100.

Проверить качество работы обученной модели можно с использованием специальной функции листинга 18. В качестве параметров выступает обученная модель, словарь токенов, длина начального набора, начальный набор и длина текста, который нужно сгенерировать.

*Листинг 18. Функция генерации текста с использованием обученной модели*

```
gen_text <-function(model, toks, n_begin,
                     beg_words, n_next_words){
  form_text <- beg_words
  for(i in seq(n_next_words)){
    encoded <- texts_to_sequences(toks,form_text)[[1]]
    encoded <- pad_sequences(sequences = list(encoded),
                            maxlen = n_begin,padding = 'pre')
    what_next <- predict_classes(model,encoded, verbose=0)
    next_word <- toks$index_word[[as.character(what_next)]]
    form_text <- paste(form_text, next_word)
  }
  return(form_text)
}
```

В листинге 19 представлена апробация обученной сети *model* с помощью функции *gen\_text()*. Наша модель предсказывает следующее слово из исходных трех предыдущих слов. В первом примере начальный набор – это «приходила к мухе», а предсказанное слово – «бабушка», таким образом создается последовательность из трех слов. В следующей берется последние три слова предложения «к мухе бабушка» и предсказывается следующее слово «пчела». Функция продолжает генерировать текст, пока не будет получено *n\_next\_words* = 16 слов.

*Листинг 19. Апробация обученной модели*

```
> begin = "приходила к мухе"
> cat("Текст 1: ",gen_text(model,toks,inp_len,begin,16),"\n ")
Текст 1: приходила к мухе бабушка пчела мухе цокотухе меду принесла... «бабочка красавица кушайте
варенье или вам не нравится наше угощенье
>
> begin = "эй сороконожки бегите"
> cat("Текст 2: ",gen_text(model,toks,inp_len,begin,16),"\n ")
Текст 2: эй сороконожки бегите по дорожке зовите музыкантов будем танцевать музыканты прибежали
в барабаны застучали бом бом бом бом
```

Как видно из листинга 19, сеть отлично справилась с первым заданием. Для второго примера, с начальным набором «эй сороконожки бегите», в конце сгенерированной фразы получен лишний «бом». Объясните почему и как это исправить?

## **Лабораторная работа № 9**

Выполните любой вариант работы.

### **Вариант 1**

Сформируйте произвольный корпус и постройте рекуррентную сеть на базе модуля *LSTM*. Добейтесь максимальной точности предсказаний, путем подбора оптимальных параметров архитектуры и обучения.

Отчет должен содержать структуру сети, сформированную функцией *summary()*, графики изменения потерь (*loss*) и точности (*acc*). Примеры апробации работы сети, с использованием функции *gen\_text()*.

### **Вариант 2**

Сформируйте произвольный корпус и постройте рекуррентную сеть на базе модуля *GRU*. Добейтесь максимальной точности предсказаний, путем подбора оптимальных параметров архитектуры и обучения.

Отчет должен содержать структуру сети, сформированную функцией *summary()*, графики изменения потерь (*loss*) и точности (*acc*). Примеры апробации работы сети, с использованием функции *gen\_text()*.

## **ЗАДАНИЯ ДЛЯ ПРАКТИКИ**

Особенностью всех заданий для практики является самостоятельный выбор способа решения поставленной задачи. Совсем не обязательно получить наилучшее решение, тем более, что никогда в этом нельзя быть уверенными при решении подобных задач.

### **Задание № 1: Моделирование логической функции**

Научиться формировать обучающую выборку и тестовый набор данных. Освоить способы создания, модификации и обучения простейшей нейронной сети. Создать нейронную сеть, моделирующую логическую функцию по предложенным вариантам (табл. 7).

Таблица 7

Варианты задания

Номер варианта	Логическая функция (цифра в начале означает число входов)
0	"4И"
1	"4И-НЕ"
2	"4ИЛИ"
3	"4ИЛИ-НЕ"
4	исключающее "4ИЛИ"
5	исключающее "4ИЛИ-НЕ"
6	"4И" с инверсией по первому входу
7	"4И-НЕ" с инверсией по первому входу
8	"4ИЛИ" с инверсией по первому входу
9	"4ИЛИ-НЕ" с инверсией по первому входу

Отчет должен содержать скрипт подготовки обучающих данных и обучения, топологию сети, входные данные, эталонные выходные данные, полученные выходные данные, выводы по результатам работы.

### **Задание № 2: Классификация входных данных**

Входными данными является восьмиразрядный двоичный вектор. Первые четыре разряда представляют исходный код, а последние четыре – исходный код, циклически сдвинутый на один разряд вправо или влево. Необходимо построить нейронную сеть для определения направления двоичного сдвига. Для решения поставленной задачи, следует построить и обучить нейронную сеть, которая должна будет определять направление циклического двоичного сдвига. Сущность операции циклического двоичного сдвига заключается в том, что число представляется в двоичном коде, а

затем с полученной последовательностью производится операция циклического сдвига вправо или влево (табл. 8).

Таблица 8

Представление чисел при сдвигах

Число	Двоичный код	Сдвиг влево	Сдвиг вправо
1	0001	0010	1000
2	0010	0100	0001
3	0011	0110	1001
4	0100	1000	0010
5	0101	1010	1010
6	0110	1100	0011
7	0111	1110	1011
8	1000	0001	0100
9	1001	0011	1100
10	1010	0101	0101
11	1011	0111	1101
12	1100	1001	0110
13	1101	1011	1110
14	1110	1101	0111
15	1111	1111	1111

Очевидно, что для определения направления сдвига на входы нейронной сети необходимо представить исходную четырехпозиционную двоичную последовательность и четырехпозиционную двоичную последовательность, которая получилась в результате сдвига.

Отчет должен содержать скрипт подготовки обучающих данных и обучения, топологию сети, входные данные, эталонные выходные данные, полученные выходные данные, выводы по результатам работы.

### Задание № 3: Построение карты Кохонена

Изучить приемы работы с самообучающимися нейронными сетями и особенности их использования на примере самоорганизующейся сети Кохонена. Выполните оба варианта задания.

#### Вариант 1

Постройте сети Кохонена для кластеризации данных *yeast* (дрожжи), которые после загрузки библиотеки *kohonen* можно просто загрузить командой *data(yeast)*. Воспользуйтесь функцией *supersom()*. Объясните полученные результаты.

#### Вариант 2

На ресурсе <http://www.instantr.com/wp-content/uploads/2013/01/europe.csv> предоставлены данные о странах Европы. Исследуйте эти данные с помощью сети Кохонена. Объясните полученные результаты.

## **Задание № 4: Распознавание и классификация объектов**

### ***Изучение стандартной модели DNN***

#### ***Цель работы***

Научиться создавать и обучать многослойные нейронные сети.

#### ***Задание***

Создать нейронную сеть со структурой многослойный персептрон, и обучить ее распознаванию цифр, заданных пиксельной матрицей размером 5\*7 клеток, в соответствии с номером своего варианта (табл. 9). В каждом варианте задания указана своя дополнительная функция.

В качестве входных данных предлагается 5 файлов в формате *csv*, каждый из которых содержит по 10 изображений цифр в виде матрицы 5 на 7 пикселов, развернутых в векторы длиной по 35 элементов каждый:

- *set1.csv* - изображения 10 цифр (от 0 до 9) в нормальной ориентации.
- *set2.csv* - изображения 5 цифр (от 0 до 4) в нормальной ориентации (первые 5 векторов) и в перевернутом виде (следующие 5 векторов).
- *set3.csv* - изображения 5 цифр (от 5 до 9) в нормальной ориентации (первые 5 векторов) и в «зеркальном» виде по вертикали (следующие 5 векторов).
- *set4.csv* - изображения 10 цифр (от 0 до 9) в «зеркальном» виде по горизонтали.
- *set5.csv* - изображения 5 цифр (от 5 до 9) в перевернутом виде (первые 5 векторов) и в «зеркальном» виде по горизонтали (следующие 5 векторов).

Таблица 9  
Варианты заданий

Номер варианта	Набор данных	Дополнительная функция распознавания
0	<i>set1.csv</i>	Распознать четность числа
1	<i>set3.csv</i>	Распознать нечетность числа
2	<i>set4.csv</i>	Распознать числа, делящиеся на 3 без остатка
3	<i>set1.csv</i>	Распознать простые числа
4	<i>set4.csv</i>	Распознать числа кратные 4
5	<i>set1.csv</i>	Распознать нечетность числа
6	<i>set3.csv</i>	Распознать четность числа
7	<i>set5.csv</i>	Распознать простые числа
8	<i>set4.csv</i>	Распознать числа кратные 4
9	<i>set1.csv</i>	Распознать числа, делящиеся на 3 без остатка

#### ***Отчет по работе***

Оформить отчет в электронном виде.

Состав отчета: задание на лабораторную работу, описание топологии сети, входные данные, эталонные выходные данные, полученные выходные данные, выводы.

### Задание № 5: Сжатие данных

Выберите любой вариант задания.

#### Вариант 1

Определите минимальный размер скрытого слоя в ограниченной машине Больцмана, достаточный для удовлетворительного представления 25 изображений из набора *FasionMNIST*. При загруженной библиотеке *keras*, этот набор загружается командой *dataset\_fasion\_mnist()*.

Оценку качества представления производить визуально, конкретные изображения (25 штук) для эксперимента выбрать произвольно.

Повторите эксперимент с использованием стека из трех ограниченных машин Больцмана, где последний скрытый слой состоит из того же количества нейронов, которое получено для однослойной машины.

Сравните полученные результаты.

#### Вариант 2

Построить сеть глубокого доверия *DBN* для классификации данных из набора *FashionMNIST*. Набор состоит из обучающего множества 60000 изображений и тестового множества из 10000 примеров.

### Задание № 6: Составление прогноза

Задача прогноза - определить, заработает ли человек более 50 тысяч долларов в год на основе 14 характеристик. Обучение сети следует провести на основе информации, взятой из базы данных бюро переписи населения США на ресурсе <http://archive.ics.uci.edu/ml/datasets/Adult>.

Данные разделены в отношении 2/3 и 1/3 случайным образом на обучающий набор из 32561 вектора и тестовый набор из 16281 вектора.

Данные включают 15 следующих характеристик:

- *Age*: возраст в годах.
- *WorkClass*: (вид трудовой занятости) *Private*, *Self-emp-not-inc*, *Self-emp-inc*, *Federal-gov*, *Local-gov*, *State-gov*, *Without-pay*, *Never-worked*.
- *fnlwgt*: финальный оценочный вес.
- *Education*: *Bachelors*, *Some-college*, *11th*, *HS-grad*, *Prof-school*, *Assoc-acdm*, *Assoc-voc*, *9th*, *7th-8th*, *12th*, *Masters*, *1st-4th*, *10th*, *Doctorate*, *5th-6th*, *Preschool*.
- *EducationNum*: продолжительность образования в годах.
- *MaritalStatus*: *Married-civ-spouse*, *Divorced*, *Never-married*, *Separated*, *Widowed*, *Married-spouse-absent*, *Married-AF-spouse*.

- Occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- Relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- Race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- Sex: Female, Male.
- CapitalGain: прирост капитала.
- CapitalLoss: убыль капитала.
- HoursPerWeek: занятость в рабочих часах в неделю.
- NativeCountry: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
- IncomeLevel: годовой доход.

### Задание № 7: Классификация объектов

Загрузите набор данных из ресурса <http://archive.ics.uci.edu/ml/datasets/Abalone>. Набор *abalone* (морское ушко) включает 4177 векторов (первые 3133 используются для обучения, остальные 1044 для тестирования). Входной вектор включает 8 первых параметров приведенных в табл. 10.

Таблица 10

Характеристика параметров набора данных *abalone*

Параметр		Описание	
<i>Sex</i>	Пол	<i>M, F, I</i>	<i>I</i> - ребенок
<i>Length</i>	Длина	мм	максимальный
<i>Diameter</i>	Диаметр	мм	перпендикулярно длине
<i>Height</i>	Высота	мм	с мясом в скорлупе
<i>WholeWeight</i>	Полный вес	г	целого морского ушка
<i>ShuckedWeight</i>	Очищенный вес	г	вес мяса
<i>VisceraWeight</i>	Масса внутренностей	г	мясо кишki
<i>ShellWeight</i>	Масса скорлупы	г	после сушки
<i>Rings</i>	Число колец		дает возраст в годах

Выходной вектор должен быть построен на основе данных о возрасте морского ушка (параметр *Rings*). Для этого все векторы набора нужно разделить на три класса:

- класс 1 – *Rings* = 1-8,
- класс 2 – *Rings* = 9 и 10,
- класс 3 – *Rings* > 10.

Задача состоит в том, чтобы построить и обучить нейронную сеть как можно наиболее точно классифицировать векторы из набора данных *abalone*, то есть определять возраст морского ушка по семи параметрам.

### Задание № 8: Распознавание изображений

Ознакомьтесь с набором данных *CIFAR-10* и сформируйте нейронную сеть для распознавания изображений.

*CIFAR-10* состоит из 60000 цветных изображений 32x32 в 10 классах, по 6000 изображений в классе. Есть 50000 учебных изображений и 10000 тестовых изображений. На рис. 18 представлены классы в наборе данных, а также 10 случайных изображений из каждого:

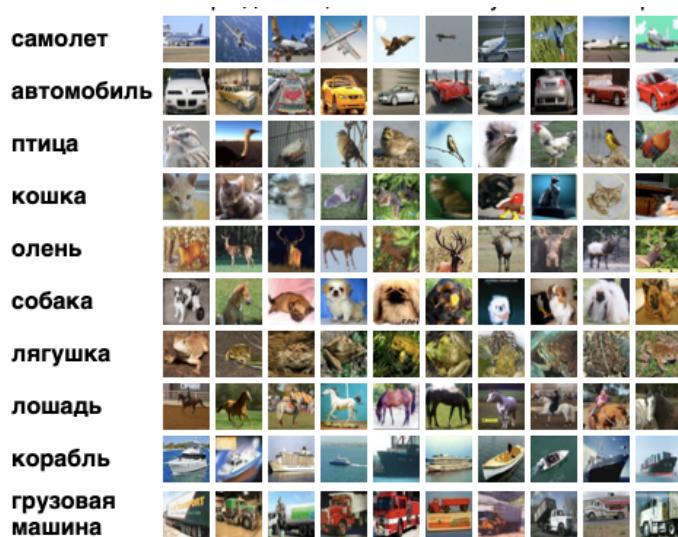


Рис. 18. Классы из набора *CIFAR-10*

Классы полностью взаимоисключающие. Между легковыми и грузовыми автомобилями нет совпадений. «Автомобиль» включает в себя седаны, внедорожники и тому подобное. «Грузовик» включает в себя только большие грузовики. Ни один не включает пикапы.

При формировании нейронной сети можно воспользоваться материалами ресурса <https://keras.rstudio.com/articles/examples/index.html>.

## Задание № 9: Классификация изображений

*FashionMNIST* это набор данных в виде изображений в градациях серого 28x28, связанных с меткой из 10 классов. Набор состоит из обучающего множества 60000 примеров и тестового набора из 10000 примеров. По сути, набор *FashionMNIST* служит прямой заменой классического набора *MNIST* для сравнительного анализа алгоритмов машинного обучения. Он имеет одинаковый размер изображения и структуру разделений обучения и тестирования. На рис. 22 приведен пример того, как выглядят данные (каждый класс занимает три строки).

При выполнении задания можно воспользоваться материалами ресурса [https://keras.rstudio.com/articles/tutorial\\_basic\\_classification.html](https://keras.rstudio.com/articles/tutorial_basic_classification.html).

Каждое изображение необходимо сопоставить с одной меткой. Поскольку имена классов не включены в набор данных, следует сформировать их в виде вектора *class\_names* для использования при построении изображений: *class\_names* = c('Футболка', 'Брюки', 'Пуловер', 'Платье', 'Пальто', 'Сандали', 'Рубашка', 'Тапки', 'Сумка', 'Ботильоны')

Необходимо подобрать параметры нейронной сети так, чтобы точность распознавания объектов из набора данных составляла не менее 95%.

Результат тестирования сети представьте в виде изображений тестового набора из 25 объектов с наименованием распознанных объектов (рис. 19).

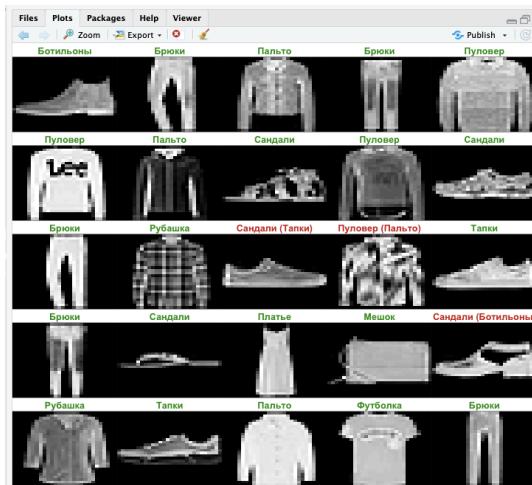


Рис. 19. Результат распознавания объектов *Fashion-MNIST*

Наименования правильно распознанных объектов подпишите зеленым цветом, неправильно распознанные объекты подпишите красным цветом, в скобках укажите правильное название.

## **Задание № 10: Прогнозирование аварий сетевого оборудования**

В качестве задания будет служить постановочная статья [15], в которой изложена задача и методика подготовки обучающих данных. Исходные данные предоставлены на ресурсе [16].

Необходимо построить нейронную сеть качественно выявляющую отклонения в работе коммуникационной сети, приводящие к сбоям.

## **ЗАКЛЮЧЕНИЕ**

Выполнение лабораторных работ и практических заданий позволяет приобрести начальный опыт применения нейронных сетей разной архитектуры для решения классических задач.

Совершенствование практических навыков использования нейросетей удобно с использованием хороших наборов обучающих данных (датасетов). Универсальным ресурсом позволяющим отыскать подходящие наборы является <https://datasetsearch.research.google.com>. С помощью этого сервиса можно находить уникальные датасеты в тысячах хранилищ по всему Интернету, используя ключевые слова.

Поиск оптимальных вариантов решения задач с помощью нейросетей возможен только при наличии глубоких знаний по классическим методам оптимизации, регуляризации, нормировке, использованию предобученных моделей [18 - 23]. Изучение этих и других источников позволит выбрать нужную архитектуру сети, определить наиболее эффективные алгоритмы ее обучения и параметры настройки.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Филиппов Ф.В. Нейросетевые технологии: учебное пособие/ Ф.В.Филиппов, СПбГУТ, – СПб., 2020. – 126 с.
2. Филиппов Ф.В. Программирование на языке R: практикум/ Ф.В.Филиппов, СПбГУТ, – СПб., 2018. – 78 с.
3. Welcome to Colaboratory, <https://colab.research.google.com/notebooks/intro.ipynb> (дата обращения 02.08.2020).
4. Филиппов Ф.В. Моделирование нейронных сетей на R: учебное пособие/ Ф.В.Филиппов, СПбГУТ, – СПб., 2016. – 84 с.
5. Самсонов Т. Визуализация и анализ географических данных на языке R, <https://tsamsonov.github.io/r-geo-course/graphics.html> (дата обращения 02.08.2020).
6. Репозитарий машинного обучения, <http://archive.ics.uci.edu/ml/index.php> (дата обращения 02.08.2020).
7. Шитиков В. К., Мастицкий С. Э. Классификация, регрессия, алгоритмы Data Mining с использованием R, 2017, e-book: <https://github.com/ranalytics/data-mining>.
8. <https://cran.r-project.org/web/packages/fpp2/fpp2.pdf> (дата обращения 02.08.2020).
9. Hyndman R.,Athanasopoulos G. Forecasting: Principles and Practice, Monash University, Australia, <https://otexts.com/fpp2/residuals.html>, (дата обращения 02.08.2020).
10. Филиппов Ф.В., Губин А.Н. Обработка информации в среде *RStudio*: учебное пособие/ Ф.В.Филиппов, А.Н. Губин, СПбГУТ, – СПб., 2015. – 99 с.
11. <https://habr.com/ru/post/456740/>
12. <https://craftappmobile.com/ограниченные-машины-больцмана> (дата обращения 02.08.2020).
13. <https://rdrr.io/github/TimoMatzen/RBM/man/> (дата обращения 02.08.2020).
14. Филиппов Ф.В. Моделирование нейронных сетей глубокого обучения: учебное пособие / Ф. В. Филиппов; СПбГУТ, – СПб., 2017. – 84 с.
15. Тайманов Д.С. Прогнозирование аварий сетевого оборудования с использованием нейронных сетей, Информационные технологии и нанотехнологии (ИТНТ-2017) Самара, 25-27 апреля 2017 г. – с. 1812-1816
16. Соревнование «Telstra Network Disruptions» [Электронный ресурс]. – Режим доступа: <https://www.kaggle.com/c/telstra-recruiting-network> (дата обращения 02.08.2020).
17. <https://cran.r-project.org/web/packages/keras/keras.pdf> (дата обращения 02.08.2020).
18. Gupta S. Deep Learning with R. Pact, 2020, p. 309.
19. Шолле Ф. Глубокое обучение на R. Manning, 2018, 400 с.
20. Микелуччи у. Прикладное глубокое обучение: Пер. с англ. – СПб.: БХВ-Петербург, 2020. – 386 с.
21. Джулли А., Пал С. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow: Пер. с англ. А.А. Слинкина – М.: ДМК Пресс, 2018. – 294 с.
22. Фостер Д. Генеративное глубокое обучение. Творческий потенциал нейронных сетей. — СПб.: Питер, 2020. — 336 с.
23. Гудфеллоу Я., Бенджио И., Курвиль А. Глубокое обучение: Пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.