Автоассоциативная нейронная сеть.

Рассмотренные ранее однослойный и многослойный персептроны представляют из себя нейронные сети, обучаемые с учителем. То есть, для их обучения необходимы заранее подготовленные ответы по каждому примеру признаков обучающего множества. Существует класс нейронных сетей, обучаемые без учителя. Чаще всего задачи таких нейронных сетей сводятся к определению наличия сложных зависимостей и корреляций среди всех признаков объекта исследования. Подобный класс сетей на практике весьма необходим для решения сложных задач. При **глубоком обучении**, прежде чем использовать признаки в качестве входных сигналов, на их основе обучается отдельная нейронная сеть для выявления наиболее важных признаков. В подобных задачах используют автоассоциативные нейронные сети. Их также называют автокодерами, автокодировщиками или автоэнкодерами.

Теоретическая часть

Автоассоциативная нейронная сеть немного отличается от многослойного персептрона в плане архитектуры, однако алгоритм прямого распространения, тип связей между слоями и алгоритм обучения на основе обратного распространения ошибки и градиентного спуска остаются неизменными. Представим, рассмотренный ранее, многослойный персептрон. Проведя небольшие изменения в его архитектуре, будет получена автоассоциативная нейронная сеть. Сеть будет содержать входной слой, состоящий из входных сигналов, один скрытый слой и один выходной слой. Количество нейронов в скрытом слое будет значительно меньше количества входных сигналов, а количество нейронов выходного слоя будет соответствовать количеству входных сигналов (рис.1).

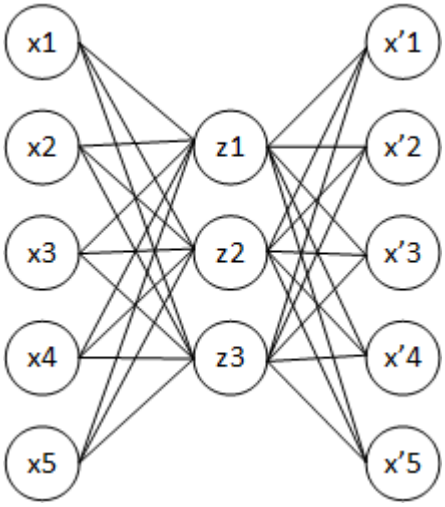

Рис 1. автоассоциативная нейронная сеть

Связываясь каждый с каждым, входные сигналы, взвешиваясь весами, поступают на вход в нейроны скрытого слоя и проходят через сигмоидальные функции активации. Нейронов скрытого слоя значительно меньше чем входных сигналов. Затем сигналы, связываясь так же каждый с каждым и взвешиваясь весами, поступают в нейроны выходного слоя. Сигналы, поступившие в нейроны выходного слоя, так же проходят через сигмоидальные функции активации. Идея достаточно проста, задача нейронной сети на выходе получить значения, максимально приближенные к значениям входных сигналов. Можно сказать, что нейронная сеть в первой половине **кодирует** сигнал, а затем во второй половине **декодирует** его, образуя таким образом в скрытом слое **код**. Именно поэтому эту сеть могут называть автокодером.

Однако, кодирование информации – это не основная задача подобной нейронной сети. В некотором смысле, автоассоциативная нейронная сеть, как и любая нейронная сеть, обучаясь, выбирает путь наименьшего сопротивления для получения результата. Если в рамках обучения от сети требуется свести сигналы в **узком горлышке**, а затем по сигналам этого узкого горлышка интерпретировать первоначальные сигналы обратно, то, по пути наименьшего сопротивления, автоассоциативная сеть в узком горлышке соберет наиболее важные сигналы, достаточные для того, чтобы, развернув, интерпретировать выходные сигналы по ним. Эта идея позволяет отслеживать наиболее важные признаки. Причем, количество наиболее важных признаков пропорционально количеству нейронов в скрытом слое.

Однако, в каждый нейрон скрытого слоя попадают линейные комбинации из всех сигналов и весов. Таким образом, нельзя однозначно утверждать, что в скрытом слое сосредоточатся именно значения наиболее важных признаков. Сеть с помощью весов усилит наиболее важные сигналы и ослабит наименее важные. Исходя из этого есть несколько методов интерпретации результата.

Метод 1. Обучая нейронную сеть, используя по очереди сочетание целевого значения с каждым признаком, оценив ошибку при тестировании каждой сети, можно по наименьшей ошибке выявить наиболее важные признаки.

Метод 2. Оценив матрицу весов и определив наибольшие значения весов, возможно определить наиболее важные признаки. Погрешность оценки будет выражаться погрешностью при тестировании нейронной сети.

Процесс, которая выполняет автоассоциативная нейронная сеть, в целом, можно интерпретировать как сравнение каждого признака с каждым. Исходя из этого, имеет смысл включить в вектор входных сигналов в том числе и целевое значение и, исходя из того, что нейронная сеть оценит это значение

как наиболее важное (так как оно коррелирует со всеми), заранее подготовить для него дополнительный нейрон в скрытом слое.

Вспомним, рассматривая многослойный персептрон, мы определили, что каждый его слой осуществляет некоторое отображение на пространство другой размерности. Исходя из этого, автоассоциативная нейронная сеть сначала формирует отображение на пространство значительно меньшей размерности чем исходное, а затем осуществляет обратное отображение в исходное пространство признаков. Дальнейшая идея заключается в том, что в процессе этого отображения в пространстве меньшей размерности по координатному расположению объектов формируются наиболее точные их характеристики с устраненным зашумлением. Устраненный шум содержится в сигналах, которые были заглушены весами при входе в скрытый слой. Если описать этими уточненными характеристиками состояние объекта исследования в каждый момент времени, то будет получена *главная нелинейная компонента* объекта исследования.

Алгоритм реализации и обучения сети аналогичен с алгоритмом многослойного персептрона, который мы разбирали ранее. Разница заключается лишь в том, что в данном случае рассматривается только один скрытый слой, с значительно меньшим количеством нейронов чем количество входных сигналов, и ошибка определяется путем сравнения двух векторов: вектора выходных сигналов сети и вектора входных сигналов. Искусственные единицы сохраняются так же во входном и скрытом слоях. Сеть обучается тем же алгоритмом обратного распространения ошибки с градиентным спуском, описанным в многослойном персептроне.

Практическая часть

Доступны данные о 7720 приложений Google Play: категории (Category), рейтинг (Rating), количество просмотров (Reviews), размер приложения (Size), тип приложения (Type), стоимость приложения (Price), контент рейтинг (Content Rating), жанр (Ganre), последнее обновление (Last Updateed) и количество скачиваний (Installs). Построим и обучим автоассоциативную нейронную сеть для выявления наиболее важных факторов, влияющих на количество скачиваний приложения.

Код и предобработанные данные: https://github.com/YSRoot/NeuralNetwork
(первоначальный источник данных: www.kaggle.com)

Итак, 7720 приложений, по каждому из которых имеем показатели по 9 признакам.

$$G_1(x_1, \dots, x_6), \dots, G_{714}(x_1, \dots, x_6)$$

Имеем 9-мерное пространство признаков с 7720 объектами. Обучающее множество составит 6000 объектов, а тестовое – 1720.

```
import numpy as np
import scipy.special as sp
import matplotlib.pyplot as plt
# библиотека для работы с csv и dataframe
import pandas as pd

# активационная функция (1/(e^(-x)))
def f(x):
    return sp.expit(x)
# производная активационной функции
def f1(x):
    return x*(1-x)
#функция инициализации весов
# на вход имеет несколько аргументов:
#    inputs - количество входных узлов
#    hiddens - количество узлов 1го скрытого слоя
#    outputs - количество узлов выходного слоя
def init_weight(inputs, hiddens, outputs):
#    матрица весов от входного слоя к 1му скрытому слою
#    принимает рандомные значения и имеет размерность [inputs x hiddens]
    np.random.seed(10)
    w1 = np.random.random((inputs,hiddens))
#    матрица весов от скрытого слоя к выходному слою слою
#    принимает рандомные значения и имеет размерность [hiddens+1 x outputs]
#    эта матрица имеет кол-во строк на единицу больше т.к. нужно учитывать
мнимую единицу
    np.random.seed(10)
    w2 =  np.random.random((hiddens+1,outputs))
    return w1,w2
#функция тренировки сети
# на вход имеет несколько аргументов:
#    inputs_list - обучающее множество (входные сигналы)
#    w1 - матрица весов от входного слоя к 1му скрытому слою
#    w2 - матрица весов от 1го скрытого слоя к 2му скрытому слою
#    targets_list - целевое множество
#    lr - скорость обучения сети
#    error - допустимая погрешность в обучении

def train(inputs_list, targets_list, w1, w2, lr, error):
 #    счетчик эпох
    era = 0
#    глобальная ошибка
    global_error = 1
#    список ошибок
    list_error = []
#   Главный цикл обучения, повторяется пока глобальная ошибка больше
погрешности
    while global_error > error:
#        локальная ошибка
        local_error = np.array([])
        # побочный цикл, прогоняющий данные с input_list
        # функция enumerate(matrix) возвращает индекс и значение строк
        # которая сохраняется в переменные i, value
        # i - индекс строки input_list
        # value - переменная которая хранит в себе строки матрицы input_list
        for i,inputs in enumerate(inputs_list):
            # переводит листа inputs в двумерный вид (для возможности
проведения операции транспонирования)
            inputs = np.array(inputs, ndmin=2)
#            targets - содержит локальный таргет для данного инпута
            targets = np.array(targets_list[i], ndmin=2)

#            прямое распространение
#            скалярное произведение строки на матрицу весов
```

```
            hidden_in = np.dot(inputs, w1)
#             применение активационной функции к вектору
            hidden_out = f(hidden_in)
#             добавление в начало вектора мнимой единицы для обучения сети
            hidden_out = np.array(np.insert(hidden_out,0,[1]), ndmin=2)

#             скалярное произведение строки на матрицу весов
            final_in = np.dot(hidden_out,w2)
            final_out = f(final_in)
#             вычисление ошибки выходного слоя
            output_error = targets - final_out
#             вычисление ошибки скрытого слоя
            hidden_error = np.dot(output_error, w2.T)
#             добавление в список локальных ошибок текущую ошибку
            local_error = np.append(local_error, output_error)
#             метод обратного распространение ошибки
#             изменение матрицы весов 2
            w2 += lr*output_error*f1(final_out)*hidden_out.T
#             в методе обратного распространения ошибки исключается мнимая
единичка для совпадения размерностей
#             hidden_error[:,1:] - означает весь вектор за исключением первого
элемента
            w1 += lr*hidden_error[:,1:]*f1(hidden_out[:,1:])*inputs.T
#         глобальная ошибка - это средняя по модулю от всех локальных ошибок
        global_error = abs(np.mean(local_error))
#         global_error = np.sqrt(((local_error) ** 2).mean())
#         эпоха увеличивается на 1
        era+=1
#         вывод в консоль текущую глобальную ошибку
        print(global_error)
#         в список ошибок добавляется глобальная ошибка
        list_error.append(global_error)
#         если при обучении количество эпох превысит порог 10000 то обучение
прекратится
        if era >10000: break
#     возвращает измененные веса, количество эпох, и список ошибок
    return w1, w2, era, list_error

# функция для проверки обученной сети и вывода результата
def query(inputs_list, w1,w2):
#     создаем список в котором будем хранить "outs" для тестового множества
    final_out = np.array([np.zeros(10)])
    for i,inputs in enumerate(inputs_list):
#         прямое распространение так же как и при обучении для получении "out"
        inputs = np.array(inputs, ndmin=2)

        hidden_in = np.dot(inputs, w1)
        hidden_out = f(hidden_in)
        hidden_out = np.array(np.insert(hidden_out,0,[1]), ndmin=2)

        final_in = np.dot(hidden_out, w2)
        final_out = np.r_[final_out,f(final_in)]
#     возвращаем значение вектора "out"
    return final_out

# Чтение датасета по приложениям в playstore или titanic
#data = pd.read_csv('titanic_data.csv', index_col='PassengerId')
data = pd.read_csv('googleplaystore_norm.csv', sep='\t',index_col='index')

#data = data.drop(columns=['Survived']).values
#data = data.drop('Survived', 1).values

# сохраняем размерности массива данных
n,m = data.shape
```

```
#количество данных, которые будут тестироваться
t = 1120
#кол-во данных на котором пройдет обучение
N = n - t

# .values - данные из dataframe конвертируются в numpy array
# составляем выборку обучающего множества из первых 600 строк датасета
inputs = data[0:N].values
# результат работы NN должен вернуть те же входные данные
# .copy - копирует данные с inputs
targets = inputs.copy()
# добавляем столбец мнимых единичек для множества
inputs = np.c_[np.ones(N),inputs]

# из оставшихся данных создается тестовая выборка
test = data[N:n].values
targets_test = test.copy()
test = np.c_[np.ones(t),test]


# скорость обучения
lr = 0.3
# допустимая погрешность обучения
eps = 0.001

# количество узлов в входном слое с учетом единичке
# т.е. кол-во столбцов датасета +1 мнимая единичка
input_layer = m+1
# количество узлов в скрытом слое
hidden_layer = 4
# кол-вол узлов в выходном слое равна количеству параметров в выбранном
датасете
output_layer = m

# инициализация весов в зависимости от количества узлов в слоях сети
w1, w2 = init_weight(input_layer, hidden_layer, output_layer)

# train network
w1, w2, era, lst = train(inputs, targets, w1, w2, lr, eps)
# вывод количества пройденных эпох
print("Количество пройденных эпох - " + str(era))
# result_test - сохранит значение "outs" NN
result_test = query(test, w1, w2)

# размерность гистограммы
#fig = plt.figure(figsize=(20,20))

# result - хранит в себе сумму строк весов 1 по модулю
result = []
# суммируются все строки весов кроме 1, т.к. первая строка содержит веса
мнимой единицы
for i in range(1,len(w1)):
    result.append(sum(abs(w1[i])))
# n - длина массива result
n= len(result)
# n - диапазон чисел от 0 до n
n = np.arange(n)
# gca() захватывает текущую ось и возвращает e'
ax = plt.gca()
#отрисовка гистограммы
ax.bar(n, result, align='edge') # align='edge' - выравнивание по границе, а
не по центру
ax.set_xticks(n)
# добавляем название столбцам гистограммы
```

```
ax.set_xticklabels(data.columns.values)
# показать график
plt.show()
```