Многослойный персептрон.

Ранее мы рассмотрели однослойный персептрон Розенблатта. Вспомним, он помогает решать задачу классификации линейно разделимых множеств на пространстве признаков. Для решения нетривиальных задач линейная разделимость является существенным ограничением. Как же решать более сложные задачи? Как решать задачи, в которых множества не являются линейно разделимым?

Теоретическая часть

Предположим, мы имеем **объект исследования**, и о нем нам известно n признаков, описывающих его свойства. Нам доступно обучающее множество из большого количества наблюдений за этим объектом, описывающих разные его состояния. В таком случае мы столкнулись с *n-мерным пространством признаков* (пространство R^n), где все состояния этого объекта будут описываться некоторым множеством объектов в пространстве признаков.

$$G_1(x_1, x_2, \dots, x_n), \dots, G_q(x_1, x_2, \dots, x_n),$$

где q – количество объектов

Описание простейшего однослойного персептрона сводилось к уравнению прямой, разделяющей множество на двумерном пространстве признаков на классы. Многослойный персептрон, по аналогии, является некоторой гиперплоскостью, разделяющей множества на классы в *n-мерном* пространстве признаков.

Таким образом, имея n признаков о каком-либо исследуемом объекте, при допущении что этих признаков достаточно, многослойный персептрон способен строить выводы о состоянии этого объекта. То есть способен свести некоторую функцию n переменных к некоторой функции минимальной размерности. Можно считать, что многослойная нейронная сеть способна представить искомую функцию в виде суперпозиции. Этот вывод был доказан Хехт-Нильсеном в 1987 году в результате переложения теоремы Колмогорова-Арнольда о суперпозиции.

Теорема Колмогорова-Арнольда (1957 год). *Любая непрерывная функция многих переменных представляется в виде суперпозиции непрерывных функций одного и двух аргументов.*

Теорема Хехт-Нильсена (1987 год). *Любая функция нескольких переменных может быть представлена двухслойной нейронной сетью с прямыми полными связями с N нейронами входного слоя, (2N+1) нейронами скрытого слоя с ограниченными функциями активации (например, сигмоидальными) и M нейронами выходного слоя с неизвестными функциями активации.*

Рассмотрим некоторую архитектуру многослойного персептрона (рис. 1). На вход нейронная сеть получает сигналы в виде признаков объекта $(x_1, \dots, x_n)$, описывающих один случай или одно состояние объекта исследования. Эти сигналы называются **входным слоем.** Сигналы входного слоя, взвешенные весами, поступают на вход следующего слоя, связываясь каждый с каждым. То есть каждый сигнал связывается с каждым нейроном следующего слоя. Этот тип связи называется **полносвязным,** а следующий слой после входного называется **скрытым.** Скрытых слоев может быть несколько. Последний слой называется **выходным** или **сумматором**, если в нем отсутствует функция активации.

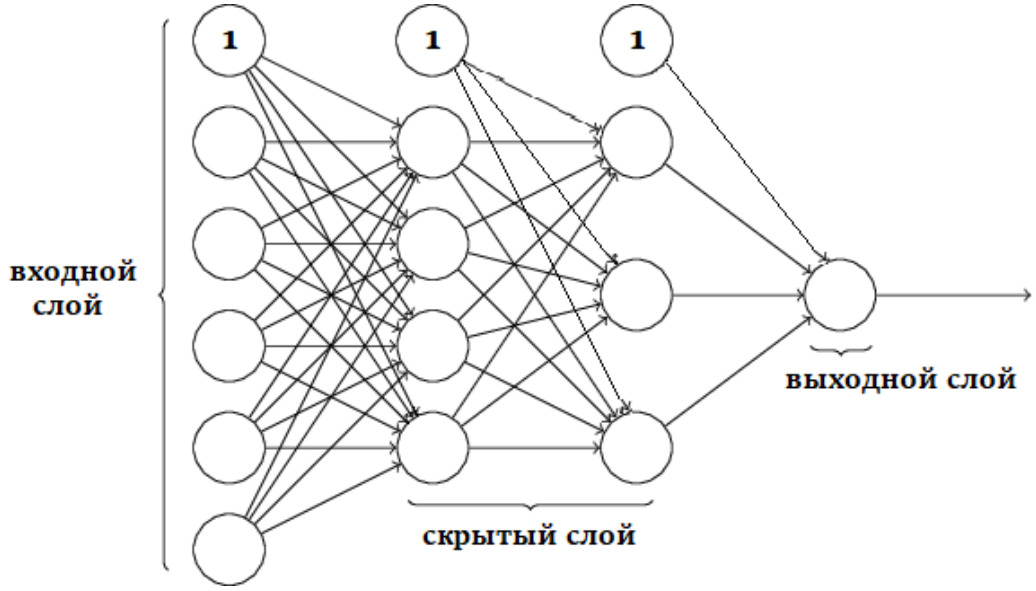


Рис.1. многослойный персептрон

Каждый нейрон содержит функцию активации. Напомним, в случае однослойного персептрона нейрон являлся прямой, разделяющей точки, описывающие объекты, на двумерном пространстве признаков на классы. На вход нейрону поступала линейная комбинация из сигналов и весов, принимающая вид прямой на пространстве признаков.

$$input = xw_x + yw_y + b$$

$$y = -x\frac{w_x}{w_y}, \qquad \text{где } input = 0$$

Затем по значению $input$ использовалась пороговая функция (рис. 2.1), присваивающая на выход нейрону значения 0 или 1 в зависимости расположения точки относительно разделяющей прямой.

$$output = 1, \qquad \text{где } input \geq 0$$

$$output = 0, \qquad \text{где } input < 0$$

Задача этой функции активации – интерпретация расположения точки относительно прямой.

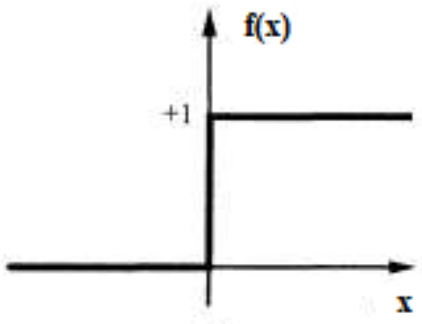


Рис 2.1. пороговая функция активации

В случае многослойного персептрона используется непрерывная сигмоидальная функция (рис. 2.2), полученная в результате сглаживания пороговой функции. Ее задача сводится к такому же распределению значения *output* относительно *input*. Необходимость использовать сглаженную функцию активации в виде сигмоиды будет раскрыта позже.

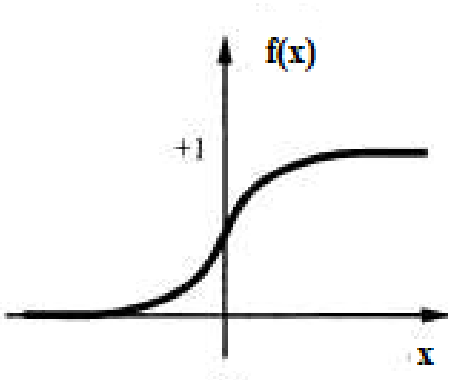


Рис 2.2. сигмоидальная функция активации

Сигмоидальная функция имеет следующий вид

$$f(x) = \frac{1}{1 + e^{-x}}$$

Линейная комбинация входных сигналов и весов, входящая в нейрон, будет распределяться в нем по сигмоиде и на выходе выдавать значение в диапазоне (0;1). Таким образом, сам нейрон есть некоторая гиперплоскость, разделяющая, как и в случае однослойного персептрона, множество на пространстве признаков на классы. Множество всех нейронов одного скрытого слоя есть совокупность таких гиперплоскостей, разделяющая множества на классы в *n-мерном* пространстве признаков. Можно считать, что каждый последующий скрытый слой по аналогии формирует некоторую гиперплоскость внутри подпространства предыдущего слоя, позволяющую более точно разделить объекты на классы внутри уже разделенных кластеров.

При этом, каждая связь двух слоев происходит по аналогии, как связь входных сигналов с первым скрытым слоем. Грубо говоря, выходные сигналы каждого последующего слоя формируют некоторое новое пространство, похожее на первоначальное пространство признаков, но с размерностью, соответствующей количеству нейронов. То есть, при переходе от одного слоя к другому происходит некоторое отображение на пространство с размерностью, равной количеству этих разделяющих плоскостей. Используя эти возможности, мы можем осуществлять отображение множества, как с уменьшением, так и увеличением размерности пространства, производя разделение исходного множества на классы внутри каждого нового пространства, характеризующегося слоем нейронной сети, а главное, свести искомую функцию многих переменных к некоторой функции минимальной размерности, достаточной для принятия решения.

Эта возможность «переходов к размерностям разных измерений» определяет существенное преимущество, благодаря которому, имея достаточное количество данных и признаков, возможно решать линейно неразделимые задачи. Если мы попытаемся описать это не математически, то каждый последующий скрытый слой позволяет как бы углубляясь в пространство, производить в нем уточняющие разделения исходного множества на классы. «Магия» нейронных сетей и ощущение сознательного принятия решения обуславливается тем, что таких разделений мы

можем произвести достаточно много. Достаточно для решения действительно нетривиальных задач, связанных с классификацией многосоставных явлений, выявления сложных зависимостей и корреляцией по неочевидным признакам, аппроксимацией, распознаванием визуальных и звуковых образов, осуществления оптимального управления и принятия решения. Однако, тем не менее, ограничения все-таки существуют, их мы рассмотрим позже.

В результате, после обучения нейронной сети, в пространстве признаков будет получена некоторая сложная форма, разделяющая множество объектов на классы с допустимой точностью. Сама нейронная сеть способна интерпретировать эту форму. Нейронная сеть будет выполнять роль функции многих переменных, где функция характеризует принятие решения, а переменные есть признаки.

Обучение

Вспомним, обучение однослойного персептрона сводилось к изменению положения разделяющей прямой на 2-мерном пространстве признаков до тех пор, пока классы не будут полностью разделены. Рассматривалось, что нейрон есть прямая. Сравнивая реальный результат с полученным, нейронная сеть определяла ошибку и принимала решение о изменении положения этой прямой в пространстве путем изменения ее коэффициентов, роль которых выполняли веса w_x и w_y, взвешивавшие входные сигналы x и y соответственно. В случае многослойного персептрона процесс обучения так же сводится к изменению весов. И в одном и в другом случае обучение есть минимизация функционала ошибки путем изменения весов.

Однако в случае многослойного персептрона принятие решения о изменении весов достигается иначе. Связано это с тем, что, в отличие от однослойного персептрона, определяя ошибку на выходе последнего слоя, нельзя применять ее для первого слоя. Так как каждый слой есть некоторое отображение на какое-то новое пространство, ошибку, вычисленную в последнем пространстве нельзя интерпретировать на искомом пространстве признаков. Однако можно распространить, в каком-то смысле, ее след. Исходя из этого, идея обучения многослойного персептрона заключается в том, что след полученной ошибки необходимо распространить от последнего слоя к первому. Этот алгоритм так и называется **обратным распространением ошибки**. След ошибки распространяется с помощью численного метода **градиентного спуска**, где градиент формируется на основе производной функции активации.

Корректировка весов необходима для изменения расположения разделяющих гиперплоскостей во всех слоях. Каждый нейрон многослойного персептрона, по аналогии с однослойным, в процессе обучения принимает решение о необходимости изменения весов исходя из расположения объекта относительно разделяющей плоскости и полученной, в последствии, ошибки. В процессе обучения, путем изменения весов, нейронной сети необходимо менять расположение разделяющей плоскости и в последствии расположить точки на сигмоиде функции активации максимально близко к 0 или 1. Таким образом, в процессе распространения ошибки необходимо учитывать удаленность точки от 0 или 1 на сигмоиде в каждом нейроне, что можно описать как удаленность от нуля значения производной функции активации. При этом сигмоидальная функция активации имеет удобный вид производной

$$f'(x) = f(x)(1 - f(x))$$

Таким образом, распространяя след ошибки с помощью градиента, нейронная есть переходит к корректировке весов во всех слоях по аналогии с однослойным персептроном.

Алгоритм.

Имеем объекты исследования G_q и признаки x_n для каждого из них. Для каждого объекта существует целевое значение T_q, описывающее его состояние.

Входные сигналы с добавлением искусственной единицы формируют вектор X. Путем произведения вектора входных сигналов X на матрицу весов W формируются вектор входных значений для нейронов скрытого слоя. Как правило стартовые значения матрицы весов W формируются случайным образом.

$$hidden\ in_j = X_q W_{i,j}\ ,\ \text{где}$$

$$X_q = [1, x_1, \dots, x_n], \quad W_{i,j} = \begin{bmatrix} w_{1,1} & \dots & w_{1,k} \\ \vdots & \ddots & \vdots \\ w_{(n+1),1} & \dots & w_{(n+1),k} \end{bmatrix},$$

$$i = \overline{1,n}, \qquad j = \overline{1,k}, \qquad k - \text{количество нейронов скрытого слоя}$$

Таким образом, на вход каждому нейрону скрытого слоя поступает линейная комбинация.

$$hidden\ in_j = w_{1,j} + x_2 w_{2,j} \dots + x_n w_{(n+1),j},\ \ j = \overline{1,k}$$

Для каждого элемента этого вектора формируется отображение по функции активации. Каждое это отображение и есть нейрон. Затем с добавлением искусственной единицы формируется выходной вектор для скрытого слоя.

$$hidden\ out = [1, f(hidden\ in_1), \dots, f(hidden\ in_k)], \qquad \text{где}\ \ f(hidden\ in) = \frac{1}{1 + e^{-hidden\ in}}$$

Далее выходной вектор скрытого слоя *hidden out* является входными для нейронов следующего скрытого слоя. Нейроны текущего слоя связываются с последующим слоем полносвязно, взвешиваясь весами – по аналогии с первым слоем. Таким образом, каждый последующий скрытый слой однотипно связывается с последующим.

Затем выходные сигналы последнего скрытого слоя связываются с выходным слоем или сумматором.

$$final\ in = hidden\ out_p\ W_p, \qquad \text{где}\ p - \text{количество слоев сети}$$

В последнем слое, по аналогии, используется сигмоидальная функция активации, однако в случае если выходной слой является просто сумматором, то используется функция активации вида

$$final\ out = final\ in$$

Затем, сравнивая выходное значение сети с целевым значением, формируется ошибка

$$e = T_q - final\ out$$

На этом заканчивается прямое распространение и начинается обратное распространение ошибки. На каждом слое, кроме первого, на основе производной функции активации, формируется градиент.

$$\delta_p = e\ W_p$$

$$\delta_{p-1} = \delta_p W_{p-1} \frac{d\ hidden\ out_{p-1}}{d\ hidden\ in_{p-1}}$$

$$\vdots$$

$$\delta_2 = \delta_3 W_2 \frac{d\ hidden\ out_2}{d\ hidden\ in_2}$$

Затем на основе градиента происходит корректировка весов

$$W_p = W_p + e\ hidden\ out_{p-1}\ \sigma$$

$$W_{p-1} = W_{p-1} + \delta_p\ hidden\ out_{p-2}\ \sigma$$

$$\vdots$$

$$W_1 = W_1 + \delta_2\ X\ \sigma,$$

где $\sigma = const$ – скорость обучения

После корректировки весов возобновляется прямое распространение.

Таким образом нейронная сеть обучается, постепенно понижая ошибку до тех пор, пока ошибка не станет меньше заданной точности. По окончанию обучения, на новых объектах путем прямого распространения, исключая обратное распространение ошибки, на обученных весах будет производиться тестирование. В результате тестирования будет определена конечная погрешность и принято решение о качестве обучения.

Практическая часть

Доступны данные о 714 пассажирах Титаника: класс проживания (Pclass), пол (Sex), возраст (Age), количество братьев и сестер на борту (SibSp), количество детей и родителей на борту (Parch), стоимость билета (Fare) и факт выживания (Survived). Построим и обучим многослойный персептрон для классификации факта выживания по доступным признакам.

Код и обработанные данные доступны на: https://github.com/YSRoot/NeuralNetwork
(первоначальный источник данных: www.kaggle.com/c/titanic)

Итак, 714 человек, по каждому из которых имеем показатели по 6 признакам.

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

Имеем 6-мерное пространство признаков с 714 объектами. Задача многослойного персептрона оптимально разделить множество этих объектов на 2 класса: выжившие пассажиры и погибшие пассажиры. Будет использоваться многослойный персептрон с 2-мя скрытыми слоями (рис. 1). Обучающее множество составит 600 объектов, а тестовое – 114.

Реализация и обучение

```
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
#    hiddens2 - количество узлов 2го скрытого слоя
#    outputs - количество узлов выходного слоя
def init_weight(inputs,hiddens,hiddens2, outputs):
#    матрица весов от входного слоя к 1му скрытому слою
#    принимает рандомные значения и имеет размерность [inputs x hiddens]
    w1 = np.random.random((inputs,hiddens))
#    матрица весов от 1го скрытого слоя к 2му скрытому слою
#    принимает рандомные значения и имеет размерность [hiddens+1 x hiddens2]
#    эта матрица имеет кол-во строк на единицу больше т.к. нужно учитывать
мнимую единицу
    w2 = np.random.random((hiddens+1,hiddens2))
#    матрица весов от 2го скрытого слоя к выходному слою
#    принимает рандомные значения и имеет размерность [hiddens2+1 x outputs]
#    эта матрица имеет кол-во строк на единицу больше т.к. нужно учитывать
мнимую единицу
    w3 =  np.random.random((hiddens2+1,outputs))
    return w1,w2,w3

#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
#    w3 - матрица весов от 2го скрытого слоя к выходному слою
#    targets_list - целевое множество
#    lr - скорость обучения сети
#    error - допустимая погрешность в обучении
def train(inputs_list,w1,w2,w3,targets_list,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
            hidden_in2 = np.dot(hidden_out,w2)
#             применение активационной функции к вектору
            hidden_out2 = f(hidden_in2)
#             добавление в начало вектора мнимой единицы
            hidden_out2 = np.array(np.insert(hidden_out2,0,[1]), ndmin=2)

#             скалярное произведение строки на матрицу весов
            final_in = np.dot(hidden_out2,w3)
#             активационная функция выходного слоя это прямая y = x, поэтому
#             здесь значение "out" равно значение "in"
            final_out = final_in
#             вычисление ошибки выходного слоя
            output_error = targets - final_out
#             вычисление ошибки второго скрытого слоя
            hidden_error2 = np.dot(output_error,w3.T)
#             вычисление ошибки первого скрытого слоя
            hidden_error = np.dot(hidden_error2[:,1:],w2.T)
#             добавление в список локальных ошибок текущую ошибку
            local_error = np.append(local_error, output_error)
#             обратного распространение ошибки
#             изменение матрицы весов 3 т.к. производная активационный функции
(y = x)
#             y` = 1 в dW = lr*output_error*hidden_out2.T не умножается на эту
производную
            w3 += lr*output_error*hidden_out2.T
#             в методе обратного распространения ошибки исключается мнимая
единичка для совпадения размерностей
#             hidden_error2[:,1:] - означает весь вектор за исключением
первого элемента
            w2 += lr*hidden_error2[:,1:]*f1(hidden_out2[:,1:])*hidden_out.T
            w1 += lr*hidden_error[:,1:]*f1(hidden_out[:,1:])*inputs.T
#         глобальная ошибка - это средняя по модуля 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,w3,era,list_error

# функция для проверки обученной сети и вывода результата
def query(inputs_list, w1,w2,w3):
#     создаем список в котором будем хранить "outs" для тестового множества
    final out = np.array([])
    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)
```

```
            hidden_in2 = np.dot(hidden_out,w2)
            hidden_out2 = f(hidden_in2)
            hidden_out2 = np.array(np.insert(hidden_out2,0,[1]), ndmin=2)

            final_in = np.dot(hidden_out2,w3)

            final_out = np.append(final_out,final_in)
#    возвращаем значение вектора "out" округленные до целого числа
    return np.around(final_out)

# считываем данные с csv с помощью библиотеки pandas
# Данные о пассажирах Титаник
# данные предоставлены Яндекс курсом
# задаем столбец по которому будет вести индексирование
index_col='PassengerId'
data = pd.read_csv('titanic_data.csv', index_col='PassengerId')
# столбец Survived из data
# .values означает что данные из dataframe конвертируются в numpy array
target_data = data['Survived'].values
# удаляем из датасета столбец Survived и конвертируем в array
#data = data.drop(columns=['Survived']).values
data = data.drop('Survived', 1).values

# составляем выборку обучающего множества из первых 600 строк датасета
inputs = data[0:600]
# добавляем столбец мнимых единичек для множества
inputs = np.c_[np.ones(600),inputs]
# составляем целевое множество
targets = target_data[0:600]

# из оставшихся 114 строк составляем тестовое множество
test = data[600:714]
test = np.c_[np.ones(114),test]
targets_test = target_data[600:714]

# скорость обучения
lr = 0.3
# допустимая погрешность обучения
eps = 10**(-8)

# количество узлов в входном слое с учетом единичке
# т.е. кол-во столбцов датасета +1 мнимая единичка
input_layer = 7
# количество узлов в скрытом слое 1
hidden_layer = 9
# количество узлов в скрытом слое 2
hidden_layer2 = 4
# количество узлов в выходном слое
output_layer = 1

# инициализация весов в зависимости от количества узлов в слоях сети
w1, w2, w3 = init_weight(input_layer, hidden_layer, hidden_layer2,
output_layer)

# тренировка сети
# train network
w1, w2, w3, era, lst = train(inputs, w1, w2, w3, targets, lr, eps)
# вывод количества пройденных эпох
print("Количество пройденных эпох - " + str(era))
# result_test - сохранит значение "outs"
result_test = query(test,w1,w2,w3)
# проверка совпадают ли значения targets_test с result_test
```

```
# Сумма всех совпадений, разделенная на количество выборки дает точность
обучения в среднем 85%
eq = sum(result_test == targets_test)/len(test)
# вывод точности
print("Точность - " + str(eq))

#отрисовка побочных графиков
#plt.plot(np.arange(114),result_test,color='r')
#plt.plot(np.arange(114),targets_test,color='b')

# отрисовка графика кривой ошибки
plt.plot(np.arange(era-1),lst)
plt.show()
```

Справочные материалы

Метод **print()** выводит в консоль объект внутри скобок

```
>>> print('Вывод текста в консоль')
Вывод текста в консоль
>>> print(10+5)
15
```

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
>>> i = 5
>>> while i < 10:
...     print(i)
...     i = i + 2
5
7
9
```

Цикл **for** уже чуточку сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла **while**. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
>>> for i in 'hello world':
...     print(i * 2, end='')
hheellllloo  wwoorrlldd
```

Условная инструкция if-elif-else - основной инструмент выбора в Python. Проще говоря, она выбирает, какое действие следует выполнить, в зависимости от значения переменных в момент проверки условия.

```
>>> if 5>2:
...     print('true')
... else:
...     print('false')
true
```

Для генерации диапазона нужно вызвать функцию range, передав ей от 1 до 3 целочисленных аргументов. В языке Python диапазон является самостоятельным объектом. Чаще всего она используется в циклах for.

```
>>> range(5)

range(0, 5)

>>> range(1,10,3)

range(1,10,3)
```

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

Метод **numpy.array()** трансформирует вложенные последовательности в многомерные массивы. Тип элементов массива зависит от типа элементов исходной последовательности

```
>>> numpy.array([1,2,3,4])

array([14, 32, 50])

>>> numpy.array([[1,2,3],[4,5,6],[7,8,9]])

array([[1, 2, 3],

       [4, 5, 6],

       [7, 8, 9]])
```

Метод **numpy.dot(a, b)**

Для двухмерных массивов это эквивалентно матричному умножению, а для 1-D массивов - скалярному произведению векторов (без комплексного сопряжения). Для размерностей N это суммирующий продукт по последней оси **“a”** и второй по последнему из **“b”**

```
>>> A = numpy.array([[1,2,3],[4,5,6],[7,8,9]])

>>> B = numpy.array([1,2,3,4])

>>> C = numpy.dot(A, B)

>>> print(C)

array([1, 2, 3])
```

$$\boldsymbol{c}[\boldsymbol{i}] = \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{n}} \boldsymbol{a}[\boldsymbol{i}] * \boldsymbol{b}[\boldsymbol{i}],$$

где $\boldsymbol{a}[\boldsymbol{i}]$ — это строка матрицы $\boldsymbol{A}, \boldsymbol{b}[\boldsymbol{i}]$ — элемент вектора $\boldsymbol{B}$

Функция **np.random.random(), np.random.random_simple()** - без аргументов возвращает просто число в промежутке [0, 1), с одним целым числом - одномерный массив, с кортежем - массив с размерами, указанными в кортеже (все числа - из промежутка [0, 1)).

```
>>> np.random.sample()

0.6336371838734877

>>> np.random.sample(3)

array([ 0.53478558,  0.1441317 ,  0.15711313])

>>> np.random.sample((2, 3))

array([[ 0.12915769,  0.09448946,  0.58778985],

       [ 0.45488207,  0.19335243,  0.22129977]])
```

Метод **numpy.random.seed()** инициализирует генератора случайных чисел. В Python, как и в любом другом языке, используется т.н. генератор псевдо случайных чисел. Т.е. random выдает не случайное число, а число которое вычисляется алгоритмом на основе другого числа, по умолчанию это текущее время. random.seed позволяет изменить число, которое передается в random для генерации случайного числа, а т.к. "случайные" числа выдаются одним и тем же алгоритмом, то при одинаковом параметре в random.seed будут и одинаковые "случайные" числа.

```
>>> np.random.random(5)

array([0.89629309, 0.12558531, 0.20724288, 0.0514672 , 0.44080984])

>>> np.random.random(5)

array([0.02987621, 0.45683322, 0.64914405, 0.27848728, 0.6762549 ])

#а если перед вызовом рандома вызвать seed()

>>> np.random.seed(3)

>>> np.random.random(5)

array([0.5507979 , 0.70814782, 0.29090474, 0.51082761, 0.89294695])

>>> np.random.seed(3)

>>> np.random.random(5)

array([0.5507979 , 0.70814782, 0.29090474, 0.51082761, 0.89294695])

#как видно последовательность рандома та же
```

Библиотека **matplotlib** — это библиотека двумерной графики для языка python с помощью которой можно создавать высококачественные рисунки различных форматов.

Эта библиотека содержит очень много возможностей для инфографики но мы воспользуемся только pyplot.plot(), pyplot.scatter(), pyplot.legend(), pyplot.show()

plt.show() – показать итоговый график

plt.scatter() - маркер или точечное рисование, аргументы:

- s - размер маркера, как для 1 значения, так и для списка
- color - цвет маркера, как для 1 значения, так и для списка
- ...

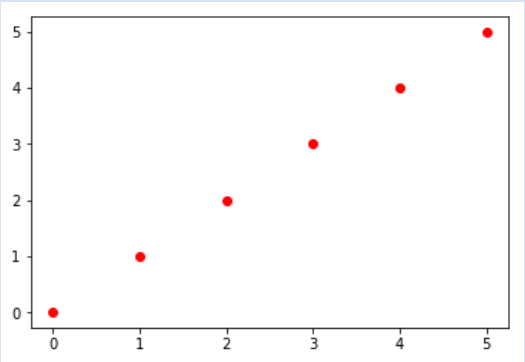
```
>>> import matplotlib.pyplot as plt

>>> x = np.arange(6)

>>> y = np.arange(6)

>>> plt.scatter(x,y,color='red')

>>> plt.show()
```



plt.plot() - ломаная линия на вход принимает аргументы:

- color - цвет линии
- label - строка легенды
- line_format - идет сразу после координат, тип линии, цвет линии, маркер точек, задается строкой
- linestyle - стиль линии
- linewidth - ширина линии
- marker - маркер точек
- markersize - размер маркера
- ...

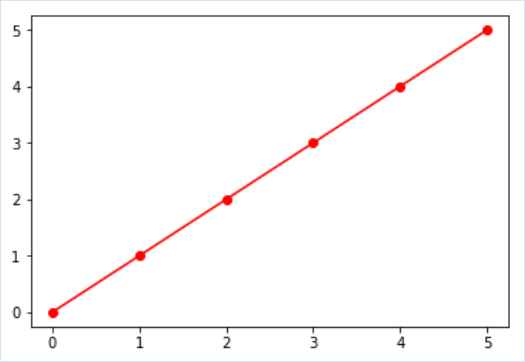
```
>>> import matplotlib.pyplot as plt

>>> x = np.arange(6)

>>> y = np.arange(6)

>>> plt.plot(x,y,color='red',marker='o')

>>> plt.show()
```



plt.legend() – добавление легенды в график, на вход принимает аргументы:

- borderaxespad - величина зазора между осями и легендой
- legend_names - список названии легенд, лучше задавать при построении графика
- loc - местоположение вывода данных легенды, можно задать как числом, так и строкой, а также кортежом позиции
- ncol - количество столбцов для легенды
- ...

```
>>> import matplotlib.pyplot as plt

>>> x = np.arange(6)

>>> y = np.arange(6)

>>> plt.plot(x,y,color='red',marker='o', label = "example" )

>>> plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.10), ncol=1)

>>> plt.show()
```